EULYNX

EULYNX Initiative

Modelling Standard

Document number: Eu.Doc.30
Version: 4.2 (1.A)

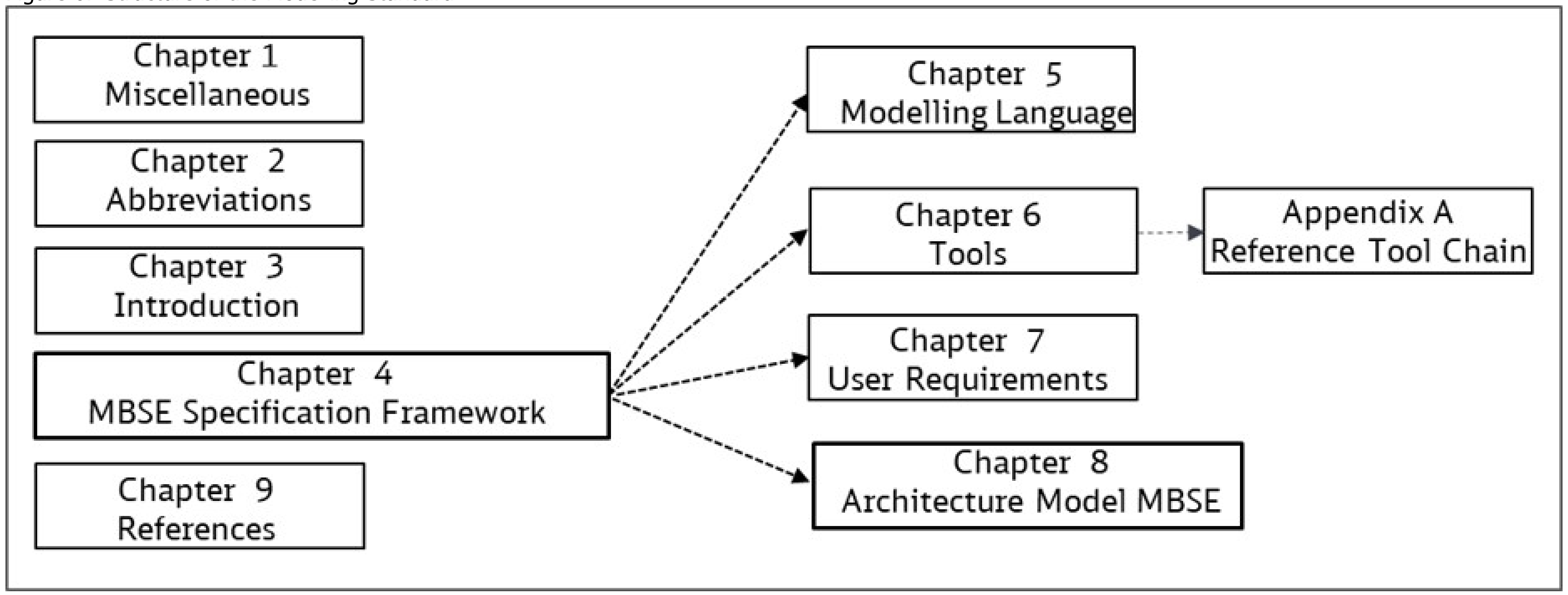
Contents

ID	Requirement
Eu.ModSt.1	# 1 Introduction
Eu.ModSt.2	## 1.1 Release information
Eu.ModSt.3	[Eu.Doc.30] Modelling Standard CENELEC Phase: 4-5 Version: 4.2 (1.A) Approval date: 02.06.2025
Eu.ModSt.1177	**Version history**
Eu.ModSt.7908	version number: 4.0 (0.A) date: 02.05.2022 author: Randolf Berglehner review: CCB changes: CCB comments incorporated. Baseline approved by CCB.
Eu.ModSt.7932	version number: 4.1 (0.A) date: 08.12.2023 author: Randolf Berglehner review: M&T changes: EUMT-61, EUMT-62, EUMT-63, EUMT-64, EUMT-65, EUMT-66, EUMT-70, EUMT-71, EUMT-75, EUMT-76, EUMT-78, EUMT-79, EUP-497
Eu.ModSt.7960	version number: 4.2 (0.A) date: 05.05.2025 author: Nico Huurman, Philipp Wolber review: M&T changes: EUMT-85, EUMT-88
Eu.ModSt.7962	version number: 4.2 (1.A) date: 20.06.2025 author: Nico Huurman review: CCB changes: EUMT-81, EUMT-89, EUMT-90
Eu.ModSt.4	## 1.2 Impressum
Eu.ModSt.5	Publisher: **EULYNX Initiative** A full list of the EULYNX Partners can be found on https://eulynx.eu/.
Eu.ModSt.7	Responsible for this document: EULYNX Project Management Office www.eulynx.eu
Eu.ModSt.1178	
Eu.ModSt.6	## 1.3 Purpose
Eu.ModSt.49	### 1.3.1 About this Modelling Standard
Eu.ModSt.50	The goal of this Modelling Standard is to provide a mandatory guideline for Model-based Systems Engineering (MBSE) of digital Command Control and Signalling systems (CCS) in the railway domain.
Eu.ModSt.52	According to MBSE introduced in this Modelling Standard the structure and functionality of digital CCS are specified using the engineering-oriented and standardised Systems Modeling Language (SysML) [1].
Eu.ModSt.1463	Furthermore, the Systems Modelling Language is embedded in a specification framework compliant to the European standards on functional safety (EN 50126, EN 50128, EN 50129, EN 50159).
Eu.ModSt.53	Based on the notion of a seamless development approach that heavily facilitates reuse, automation and innovation, an advanced and comprehensive modelling theory is used with the MBSE Specification Framework (MBSE SF) as core component. It enables a stepwise specification of digital CCS in a configurable, extendable, modular and reusable way.

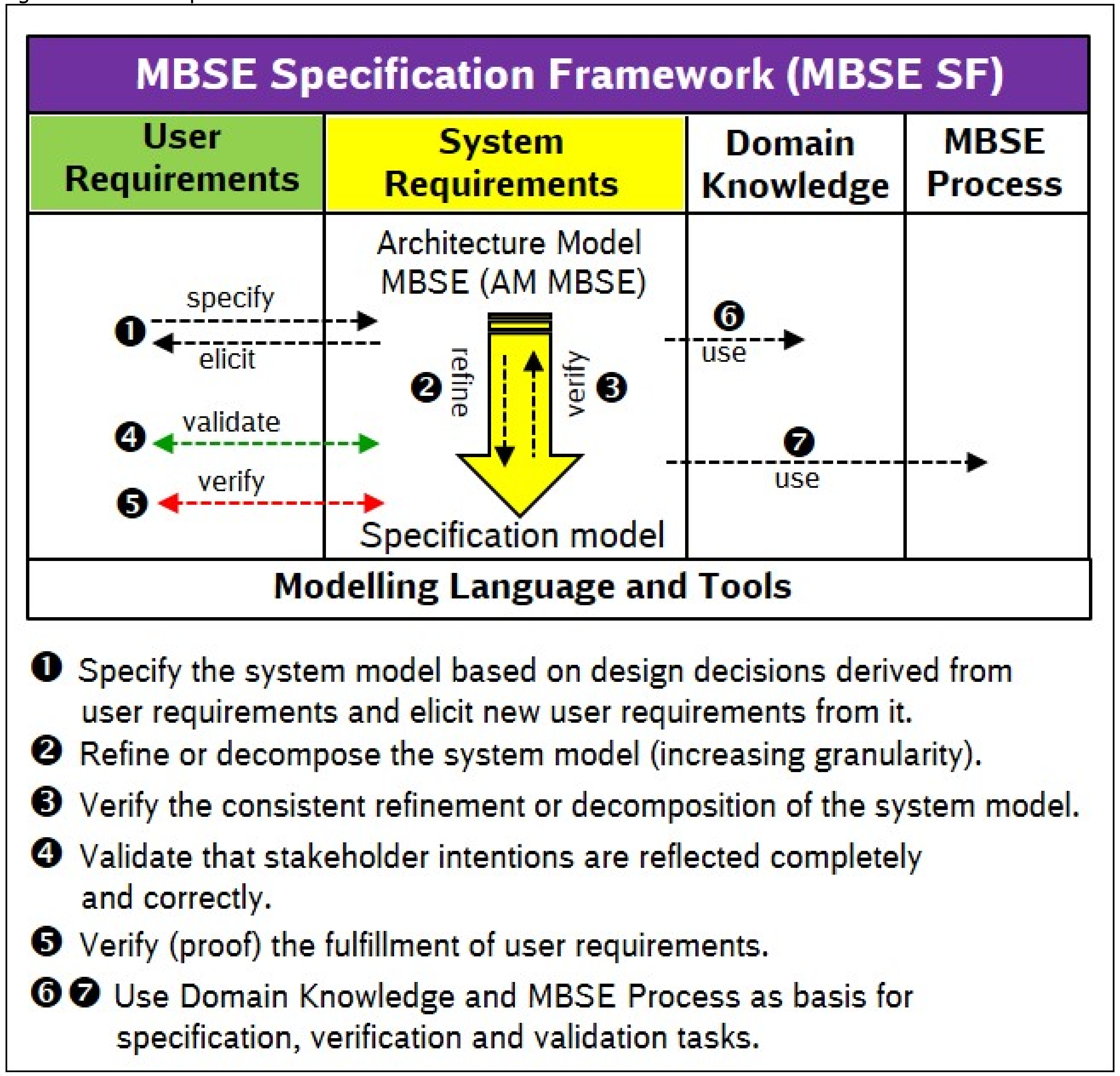
ID	Requirement	
Eu.ModSt.1975	The MBSE Specification Framework (MBSE SF) contains, among others, an Architecture Model MBSE (AM MBSE) that facilitates the description of a digital CCS from different viewpoints capturing different stakeholder concerns and with varying degrees of granularity (different abstraction levels).	
Eu.ModSt.54	It should be noted that this document is a „living document", i.e. it will evolve over time. The present version reflects the procedures that are currently being applied and evaluated in the EULYNX Initiative. Future versions of the Modelling Standard will contain the topics left out in this version.	
Eu.ModSt.864	Correspondingly, as this standard is based on standard SysML, some example diagrams and pictures obtained from diverse sources, which show enhanced graphical features such as colours, shadows, 3D or embedded pictures, shall not be considered normative.	
Eu.ModSt.7959	It should also be noted that the inserted diagrams are only to be understood as examples for methodological explanation and, although there are similarities to the content of current specifications, are not intended to convey any specification-specific content. The relevant specifications should be consulted for specification-specific content.	
Eu.ModSt.55	**1.3.2 Audience**	
Eu.ModSt.56	The audience targeted by this Modelling Standard comprises engineers being familiar with CCS, modellers creating specification models in this domain, and parties interested in understanding the MBSE approach followed in EULYNX. Fundamental knowledge about requirements- and systems engineering methodology and the modelling language SysML, as, for example introduced in [24], is recommended.	
Eu.ModSt.8	**1.4 Terms and abbreviations**	
Eu.ModSt.9	The terms and abbreviations are listed in the EULYNX Glossary [Eu.Doc.9].	
Eu.ModSt.853	The present version of the Modelling Standard contains the abbreviations listed in *Chapter 2* of it.	
Eu.ModSt.849	**1.5 Related documents**	
Eu.ModSt.850	The current versions of documents related to this document are listed in the EULYNX Documentation plan [Eu.Doc.11].	
Eu.ModSt.851	• System Engineering Process [Eu.Doc.27]	
Eu.ModSt.852	• Interpretation rules for model-based requirements [Eu.Doc.29]	
Eu.ModSt.10	**2 Abbreviations**	
Eu.ModSt.1262	Abbr.	Abbreviation
Eu.ModSt.11	ASAL	Atego Structured Action Language
Eu.ModSt.1254	AL	Abstraction level
Eu.ModSt.865	AM	Architecture Model
Eu.ModSt.12	bdd	Block definition diagram (SysML)
Eu.ModSt.13	C	Command & Control layer
Eu.ModSt.1974	CCS	Command Control and Signalling
Eu.ModSt.14	Cd	Command
Eu.ModSt.7848	CD	Connection Domain
Eu.ModSt.15	CENELEC	European standards on functional safety (EN 50126, EN 50128, EN 50129, EN 50159)
Eu.ModSt.16	Con	Configuration data
Eu.ModSt.1159	DiaNo	Diagram number
Eu.ModSt.866	D	Data
Eu.ModSt.17	D-Port	Data port
Eu.ModSt.7879	ESE	Environmental Structural Entity

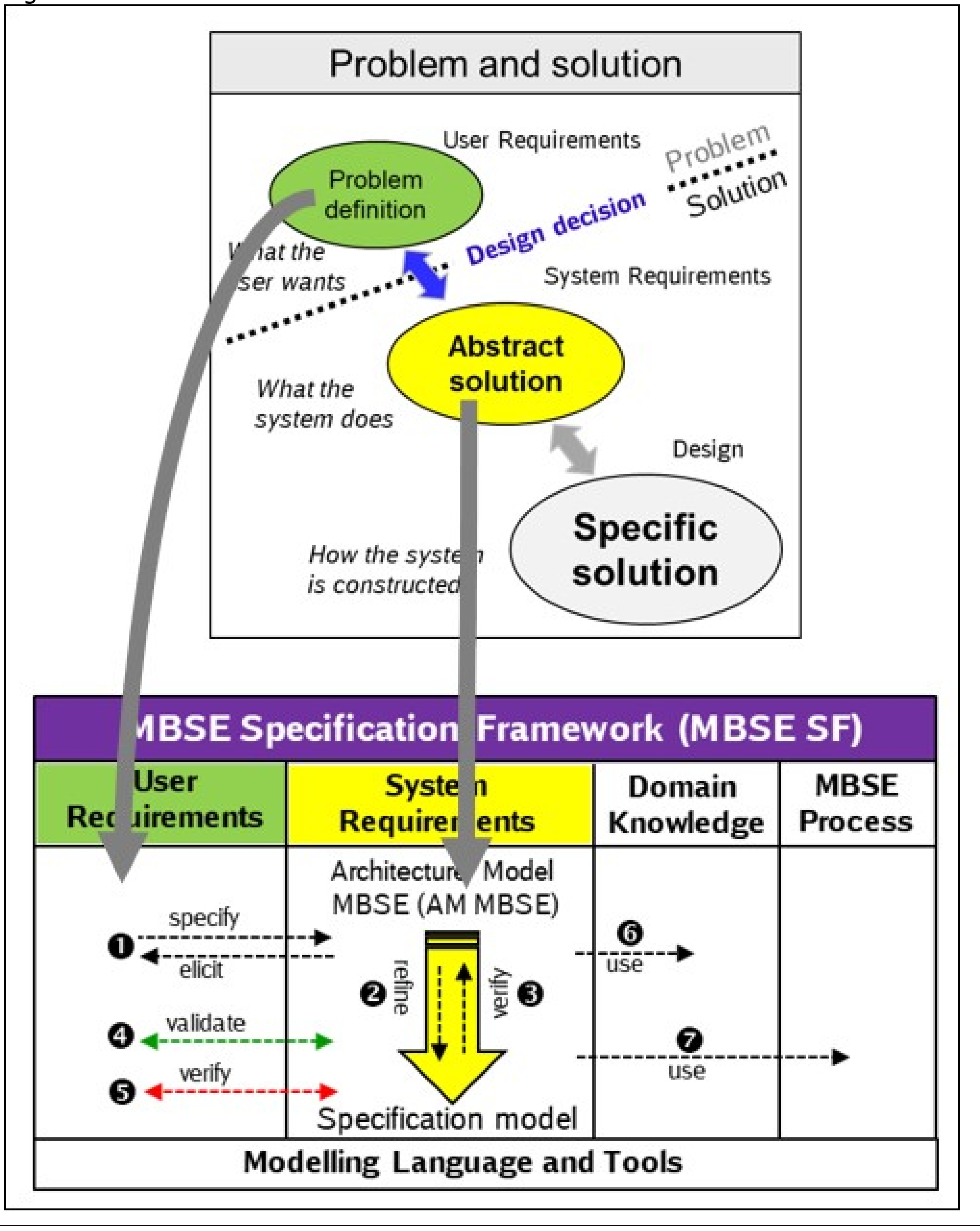
ID	Requirement	
Eu.ModSt.868	EIL	Electronic interlocking
Eu.ModSt.20	F	Field layer
Eu.ModSt.7874	FA	Functional Architecture
Eu.ModSt.7875	FE	Functional Entity
Eu.ModSt.22	Gen	Generic
Eu.ModSt.23	ibd	Internal Block Diagram (SysML)
Eu.ModSt.1976	ILS	Interlocking System
Eu.ModSt.1522	IM	Infrastructure Manager
Eu.ModSt.869	ISE	Infrastructure Elements
Eu.ModSt.24	LA	Logical Architecture
Eu.ModSt.27	LS	Light Signal
Eu.ModSt.7876	LSE	Logical Structural Entity
Eu.ModSt.28	MBSE	Model-based systems engineering
Eu.ModSt.30	MBSE SF	MBSE Specification Framework
Eu.ModSt.31	MBSEP	MBSE Process
Eu.ModSt.32	Msg	Message
Eu.ModSt.1299	OE	Operational Entity
Eu.ModSt.1521	ON	Operational Needs
Eu.ModSt.1266	PDI	Process Data Interface
Eu.ModSt.1265	PTC	Parametric Technology Corporation
Eu.ModSt.870	RA	Risk Analysis and Evaluation
Eu.ModSt.34	RAMS	Reliability, Availability, Maintainability, and Safety
Eu.ModSt.1977	RCA	Reference CCS Architecture
Eu.ModSt.36	S	Safety layer
Eu.ModSt.38	SCI	Standard communication interface
Eu.ModSt.1450	SCP	Safe Communication Protocol
Eu.ModSt.887	SIUS	System Interface under Specification
Eu.ModSt.1982	SoS	Systems of Systems
Eu.ModSt.7929	SP	System Pillar
Eu.ModSt.875	std	State diagram (SysML)
Eu.ModSt.1448	stm	State machine
Eu.ModSt.37	Sys	System
Eu.ModSt.873	SysDef	System Definition

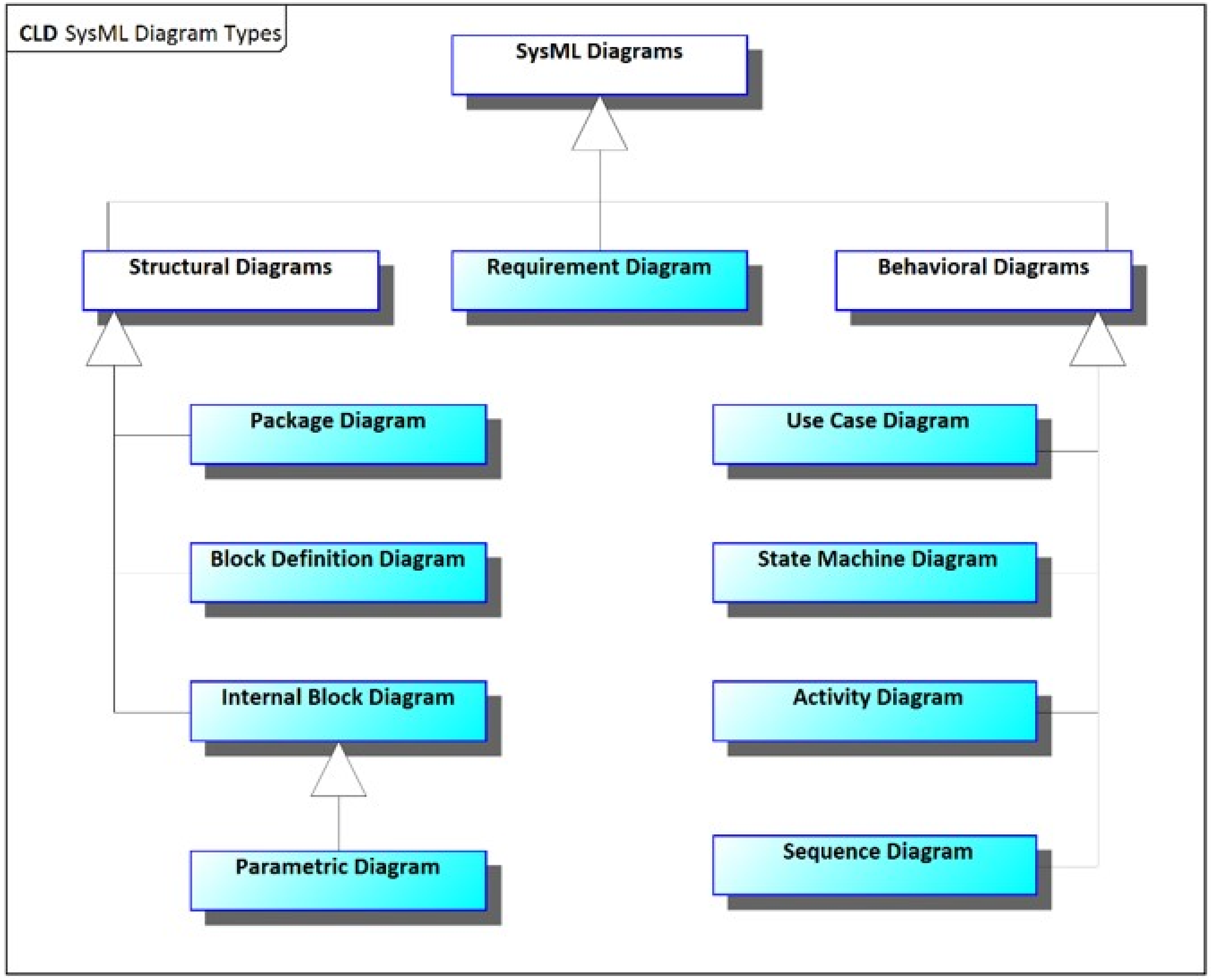
ID	Requirement
Eu.ModSt.44	SubS Subsystem
Eu.ModSt.874	SUS System under Specification
Eu.ModSt.41	SysML Systems Modeling Language
Eu.ModSt.42	SySim System simulation
Eu.ModSt.876	T Trigger
Eu.ModSt.7898	TFA Technical Functional Architecture
Eu.ModSt.7877	TFE Technical Functional Entity
Eu.ModSt.7878	TSE Technical Structural Entity
Eu.ModSt.43	T-Port Trigger port
Eu.ModSt.877	ucd UseCase diagram
Eu.ModSt.45	UML Unified modeling language
Eu.ModSt.46	VAL Validation
Eu.ModSt.47	VER Verification
Eu.ModSt.48	**3 Introduction**
Eu.ModSt.76	**3.1 Motivation**
Eu.ModSt.77	Historically, operators of rail infrastructures were supplied with monolithic systems, based on proprietary interfaces. A few years ago, a re-orientation of the means of production of future systems was initiated. This entails purchasing modular systems. For example, an interlocking system (ILS) comprises an electronic interlocking (EIL), a command control system and field elements such as points, signals, and so forth. The fundamental concept of this new approach is having these parts supplied separately [12].
Eu.ModSt.1465	The new approach requires the development of standardised interfaces between the subsystems of a digital CCS such as a digital interlocking system. This will enable the different suppliers to supply compatible modules. This requires high quality specifications, as suppliers will be working with these blueprints and the operators of rail infrastructures will carry out the system integration tasks.
Eu.ModSt.78	Furthermore, the design of a harmonised railway system with the objective of a broad EU-wide implementation, as striven for in the System Pillar (SP), requires improving specification techniques**.** Thus, it is an important issue among infrastructure managers, the railway industry and researchers to find appropriate forms to specify the architectures of complex component systems right up to huge systems of systems (SoS).
Eu.ModSt.1464	Different forms, like natural languages and graphical representations of system requirements, have been used and raised a number of criticisms. On the other hand, formal methods are considered to be one of the correct ways to specify and verify system requirements. They have been addressed in the railway domain for a number of years. To apply these formal methods, one needs a strong mathematical background.
Eu.ModSt.1978	Thus, following the goal to create high quality specifications understandable also for people without a strong mathematical background, the popular systems modeling language (SysML) [1] is used as specification language in the MBSE approach introduced in this Modelling Standard.
Eu.ModSt.79	The use of standardised interfaces and highly detailed system specifications creates a need for safety to be part of the specifications. The adoption of MBSE has therefore been part of this transformation, by proving through modelling and simulation that system specifications meet safety critical requirements.
Eu.ModSt.80	Studies of system developments show that the capture of requirements is one of the most decisive and critical steps in system development. There are many problematic aspects connected to the identification and description of requirements in software-intensive projects. The following three form the most important aspects as mentioned in [4]: • requirements are not completely and accurately identified and understood by the application expert; • requirements are not correctly specified, although completely and accurately identified and understood; • requirements are correctly specified using informal techniques, that are not properly interpreted and conceived by the system designer or the implementer. All three problems may lead to a considerably more expensive and time consuming system development.
Eu.ModSt.81	Based on these observations, an engineering-oriented model-based method for the stepwise specification of digital CCS using the Systems Modeling Language (SysML) [1] has been developed to support different professionals, especially railway engineers, to specify, validate and verify the corresponding system requirements.

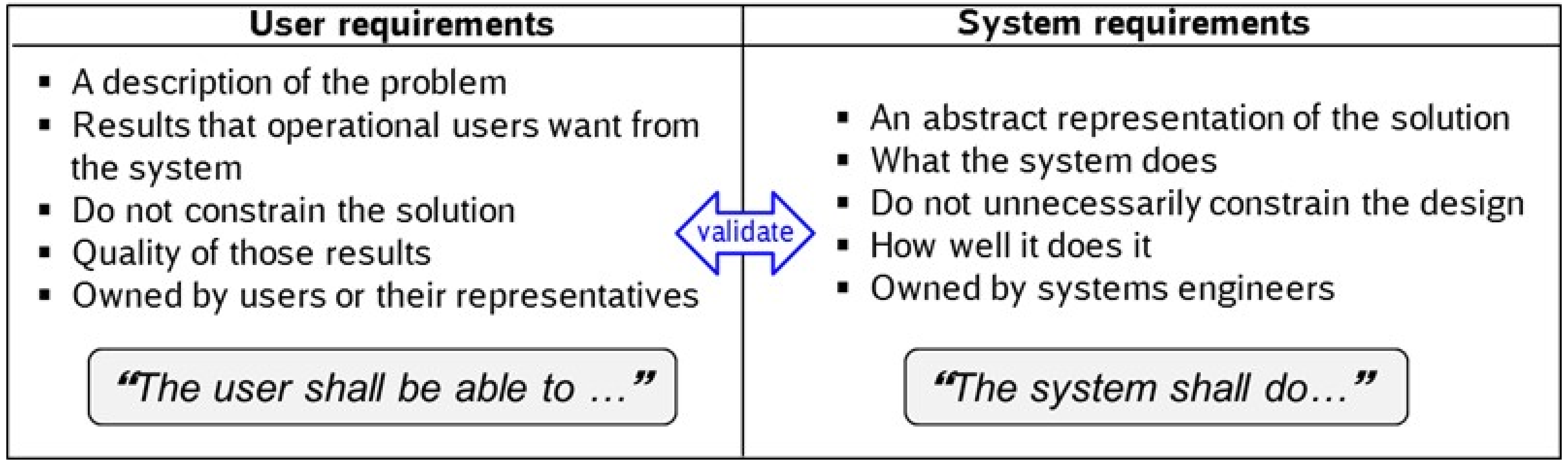
ID	Requirement
Eu.ModSt.2010	The model-based requirements definition is used to: • enable a continuous CENELEC-compatible top-down specification of a (sub)system (refinement of the requirements across different abstraction levels) • describe the functional requirements of a (sub)system or an interface operationally and therefore suitable for simulation, i.e. testable in a uniform format • support achieving consistency, non-ambiguity and completeness of the requirements as far as possible • allow for the testing by simulation of the functional requirements of a (sub)system or an interface already during the specification phase (moving error detection to the specification phase) • support the generation of (sub)system or interface test cases from the requirements specification
Eu.ModSt.2012	The system requirements are described in a consistent, non-ambiguous and compact form using the standardised semiformal language SysML. It should be noted that the SysML model elements and their interaction are to be understood as a means of describing the system requirements and not as implementation specifications. They are to be implemented with regard to their semantics.
Eu.ModSt.7899	The type of representation and the underlying methodology sometimes differs from common text-based specifications. However, the requirements can be further processed into functional specifications and products in accordance with the tested processes.
Eu.ModSt.65	**3.2 Structure of the Modelling Standard**
Eu.ModSt.66	The Modelling Standard is structured as depicted in *Figure 67*.
Eu.ModSt.67	Figure 67 Structure of the Modelling Standard 
Eu.ModSt.68	The main contents of the Modelling Standard are covered in *Chapters 3 - 10*.
Eu.ModSt.69	In ***Chapter 3***, an introduction to the Modelling Standard is given.
Eu.ModSt.71	In ***Chapter 4***, an introduction to the structure of the MBSE Specification Framework (MBSE SF) is given. The MBSE SF is the basis for the development of a stepwise model-based specification of all the design decisions that are made during the different needed engineering activities.
Eu.ModSt.72	In ***Chapter 5***, the modelling language being used is introduced.
Eu.ModSt.7928	In ***Chapter 6,*** the requirements for supporting tools necessary to implement the EULYNX MBSE process are outlined. To complement this, the tool chain currently used in EULYNX is described in ***Appendix A***. It fully supports the EULYNX MBSE process and serves as a reference for the use of alternative tool chains.
Eu.ModSt.73	In ***Chapter 7***, the area „User Requirements" of the MBSE SF is described.
Eu.ModSt.74	In ***Chapter 8***, the Architecture Model MBSE (AM MBSE) is introduced and the constituent model views are described. The characteristics of the EULYNX subsystems are highlighted and the principles of model-based requirements definition are explained. Furthermore, the MBSE process is presented in a simplified way. The main part of the chapter is dedicated to the description of the model views and the corresponding modelling rules: 8.1 Overview of the EULYNX MBSE methodology 8.1.1 Characteristics of EULYNX subsystems 8.1.2 Principle of model-based definition of requirements

ID	Requirement
	8.1.3 Overview introduction to the EULYNX MBSE Process **8.2 Model views - General modelling rules** 8.2.1 Binding of requirements 8.2.2 Modelling Pattern for interlocking systems 8.2.3 Introduction to basic structural model elements 8.2.4 Interface centric specification **8.3 Model views used to specify EULYNX subsystems** **8.4 Model views used to specify EULYNX interfaces** **8.5 Model views "Functional Entity" and "Technical Functional Entity" - Description** **8.6 Model views "Functional Entity" and "Technical Functional Entity" - Modelling rules**
Eu.ModSt.70	In ***Chapter 9***, the references are listed.
Eu.ModSt.7933	***Appendix A (chapter 10)*** describes a reference tool chain that enables the implementation of the EULYNX process.
Eu.ModSt.236	**4 MBSE Specification Framework**
Eu.ModSt.1492	Today's and, even more so, the future development of CCS systems in the railway domain faces a variety of challenges. Key success factors to meeting these challanges are suitable architecture description concepts for abstraction and structure CCS architectures at different levels of granularity. The result of these concepts is a seamless development approach that heavily facilitates reuse and automation. As stated in [25], a basic requirement for such a seamless approach is a clear notion of a system that is formalised by a comprehensive modelling theory. According to this modelling theory, a modelling framework has to provide appropriate models and description techniques for modelling the different aspects and artefacts of system development.
Eu.ModSt.237	Inspired by [25] and [26], this Modelling Standard introduces the MBSE Specification Framework (MBSE SF) in order to meet those aforementioned challenges. Focusing on system requirements specification and interface requirements specification tasks to be carried out at the infrastructure manager side, it facilitates the seamless model-based specification of • **EULYNX subsystems under Specification (SUS)** or • **EULYNX adjacent System interfaces and subsystem Interfaces under Specification (SIUS)** as well as the verification and validation of the resulting specification artefacts.
Eu.ModSt.1493	The MBSE SF consists of five areas (see *Figure 238*), namely • **User Requirements,** • **System Requirements,** • **Domain Knowledge,** • **MBSE Process** and • **Modelling Language and Tools.**
Eu.ModSt.1494	Guided by a MBSE process and based on Domain Knowledge, these areas strictly distinguish between the **problem domain (User Requirements)** and the **solution domain (System Requirements)**.

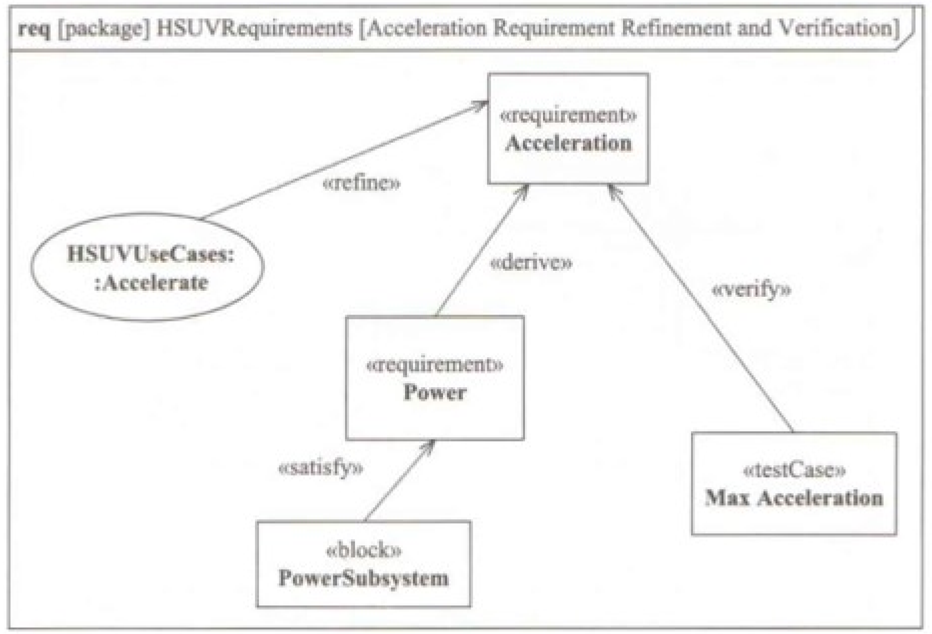
ID	Requirement
Eu.ModSt.238	Figure 238 MBSE Specification Framework 
Eu.ModSt.239	**User Requirements** The area "User Requirements" contains the model of the problem domain (problem definition) in the form of user requirements (see Fig. 1484). User requirements allow the different stakeholders to explicitly state what is expected from the future system. They are the main source for the derivation of design decisions as basis for the creation of the artefacts of an abstract system solution (system model), which itself may be again the source for the elicitation of new (possibly more granular) user requirements.
Eu.ModSt.245	It has to be verified that the design decisions derived from the user requirements are incorporated in the system model completely and correctly. In other words, it has to be proved that the system model fulfils all defined user requirements.
Eu.ModSt.1468	Furthermore, user requirements are among others (e.g. domain knowledge), the source for the validating that the system model reflects the stakeholder intentions completely and correctly.
Eu.ModSt.1486	The area "User Requirements" is described in more detail in *chapter 7*.
Eu.ModSt.240	**System Requirements** The area "System Requirements" contains the model of the solution domain in the form of a system model representing an abstract solution of the system (see *Figure 1484*). There, the design decisions derived from the user requirements are documented (specified) traceable with varying degrees of granularity (different abstraction levels) based on the Architecture Model MBSE (AM MBSE). Each abstraction level represents design decisions about the refined or decomposed implementation of its predecessor (refine dependency).
Eu.ModSt.244	The correct, complete and consistent refinement or decomposition has to be approved in verification steps (verify dependency).

ID	Requirement
Eu.ModSt.1487	The Architecture Model MBSE is described in more detail in *chapter 8*.
Eu.ModSt.243	**Domain Knowledge** The Domain Knowledge model comprises the available knowledge of the problem domain, similar to a project glossary. It hence makes up part of the context of knowledge of the system and can be used to mitigate misinterpretation, to reduce ambiguity, and to provide a possibility for early verification and validation of the system model [25].
Eu.ModSt.1488	The domain knowledge relevant for EULYNX is defined in [Eu.Doc.9] EULYNX Glossary and [Eu.Doc.10] EULYNX Domain Knowledge. The documents are available on the EULYNX website [31].
Eu.ModSt.242	**MBSE Process** The relationships between artifacts of the system model are specified by relations. Such a relation can be expressed by a process activity that defines a general technique for artefact creation and analysis. In the MBSE Process, multiple of these process activities are combined to a sequence. The output of one process activity can be input of another process activity. Furthermore, one process activity's postcondition might ensure that another process activity's precondition is met.
Eu.ModSt.1489	The EULYNX MBSE process is described in principle in *chapter 8.1*. A detailed description of the process steps will be given in a separate document in the future. The EULYNX System Engineering process is currently documented in [Eu.Doc.27] and the procedure for verification and validation of the specification models in the EULYNX verification and validation plan [Eu.Doc.31]. The documents are available on the EULYNX website [31].
Eu.ModSt.1467	**Modelling Language and Tools** The suggested modelling language and the requirements for supporting tools necessary to implement the EULYNX MBSE process. are introduced in *chapter 5* and *chapter 6* respectively.
Eu.ModSt.1484	Figure 1484 Problem definition and abstract solution in the MBSE SF 

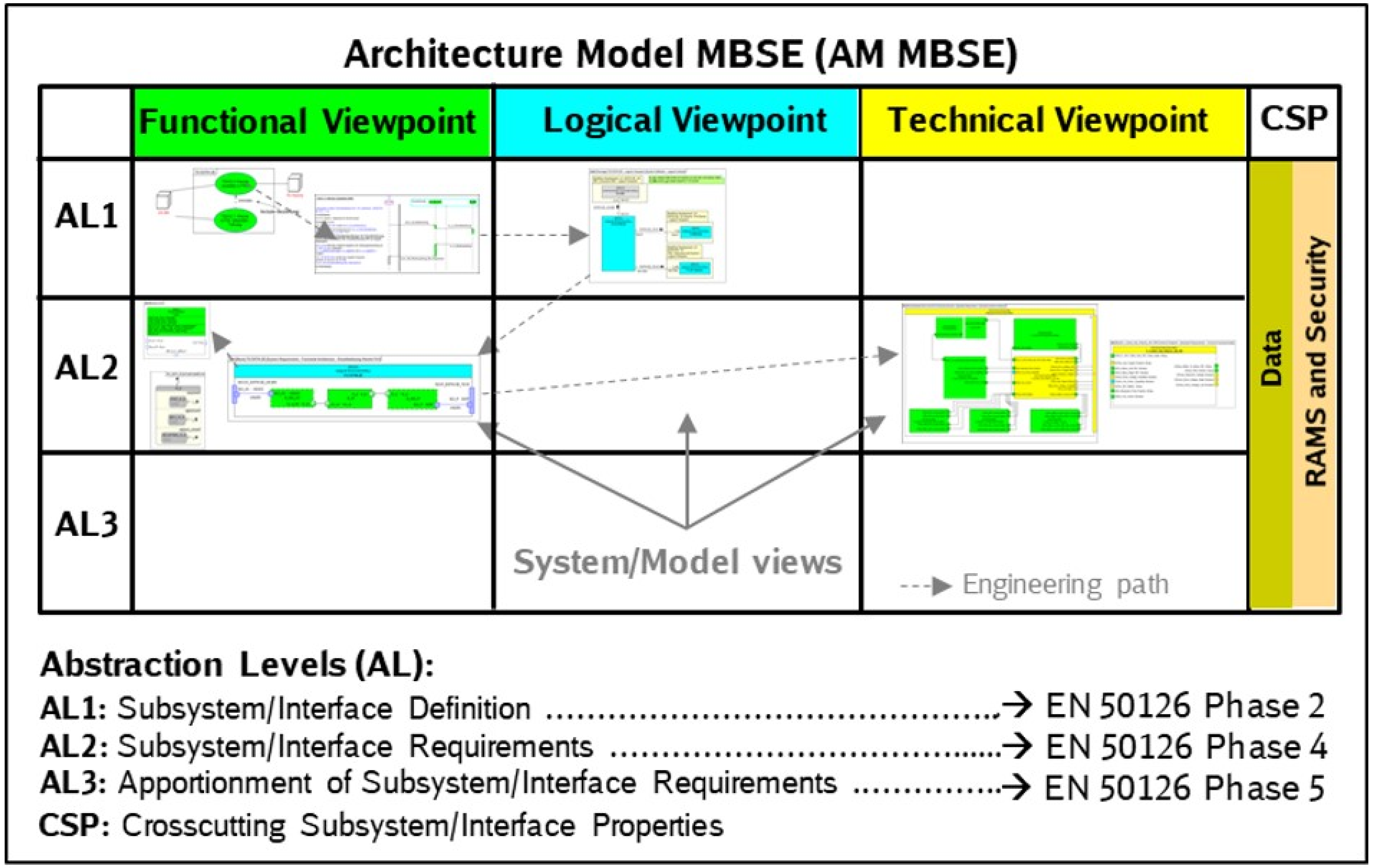
ID	Requirement
Eu.ModSt.246	**5 Modelling Language**
Eu.ModSt.247	**5.1 Systems Modeling Language (SysML)**
Eu.ModSt.248	The Systems Modeling Language [1] is used with the objective to document requirements and to specify artefacts in a standardised, correct, complete and consistent way within the framework of the MBSE specification structure, as outlined above.
Eu.ModSt.249	SysML is a standardised modeling language dedicated to systems engineering applications. It is a UML profile that not only reuses a subset of UML 2.5 [2], but also provides additional extensions to better satisfy Systems Engineering's specific needs. It is intended to help to specify and design complex systems and their subsystems and enable their analysis, verification and validation. These systems may consist of heterogeneous components such as hardware, software, information, processes, personal and facilities [1].
Eu.ModSt.250	Nine SysML diagrams (see Fig.251) define a concrete syntax that describes how SysML concepts are visualized graphically or textually. Each diagram represents a specific view of the model of the SUS or SIUS. In the SysML specification [1], this notation is described in tables that show the mapping of the language concepts into graphical symbols on diagrams. Diagrams used in this Modelling Standard will be outlined in the following chapters. For a detailed description, however, the SysML specification [1] shall be referred to.
Eu.ModSt.251	Figure 251 SysML diagram taxonomy [1] 
Eu.ModSt.252	**5.2 Action Language**
Eu.ModSt.253	The specification approach described in this modeling standard follows the objective of creating executable specification models. In order to specify the necessary executable behaviours in SysML, such as block operations or transition effects on state machines the Atego Structured Action Language (ASAL) is used.
Eu.ModSt.254	ASAL is an UML Action Language suitable for specifying executable algorithms in a target language independent way. It is used to specify the Event Action Blocks in SysML models that use state machine diagrams describing the stimulus-response behaviour of a SUS or a SIUS.

ID	Requirement
Eu.ModSt.255	Furthermore, ASAL is used to describe the transformational aspects of a SUS or SIUS (data flow). The logical structure of the input and output data, and the algorithm that computes the transformation are specified in the body of corresponding block operations.
Eu.ModSt.256	A description of ASAL is provided in *chapter 8.6.8* (see also [13]).
Eu.ModSt.7697	**5.2.1 The role of data types**
Eu.ModSt.161	According to the specification approach described in this Modelling Standard, a data type is a classification based on identification of one of the various types of data (e.g. the type of a message sent along a SUS interface). The data type such as Boolean, Integer or String restrict the possible values corresponding to that type, the meaning of data, the way values of that type can be stored and how a state machine receiving such data reacts.
Eu.ModSt.162	A data type may be refined in the tradition of data refinement [4]. We may, for example, type a message in the specification model as string, and after implementation level design of the SUS or SIUS instead of sending strings, bits are sent. Thus, a data type used in the specification model may be refined and an implementation-oriented data type may be used by the supplier of the SUS or SIUS. However, it must be ensured that the new data type complies with its predecessor (verification of the refinement).
Eu.ModSt.301	**6 Tools**
Eu.ModSt.7912	The EULYNX MBSE process shall be supported by a toolchain that enables the creation of SysML specification models, their static checking for completeness, correctness and consistency and the simulation-based validation of the models. It should be noted here that the creation of the executable models (virtual prototypes) can take place directly, i.e. without the need for the intermediate step of a model transformation.
Eu.ModSt.7913	Furthermore, the application of formal methods must be made possible (e.g. formal proof of safety properties, model checking, etc.).
Eu.ModSt.7914	The modelling tool shall provide a link to a requirements management tool that allows the representation of specification model elements in the form of atomic requirements. These must be able to be transformed into the standardised Requirements Interchange Format (ReqIF) and exchanged with suppliers.
Eu.ModSt.7915	The tool chain currently used in EULYNX is shown serving as a reference for the use of alternative tool chains in Appendix A - Reference tool chain.
Eu.ModSt.312	**7 User Requirements**
Eu.ModSt.313	**7.1 Overview**
Eu.ModSt.107	As many standards such as the EN 50126 [17] do not distinguish between a user requirements and system requirements definition phase, this has to be clarified in order to meet the objective of this Modelling Standard. The MBSE Specification Framework introduced in *chapter 4* takes account of this providing a structure to explicitly define user requirements separated from system requirements.
Eu.ModSt.314	As already stated, user requirements are depicted in the area „User Requirements“ of the MBSE SF and describe the problem domain (problem definition). They allow the stakeholders (users) to explicitly state what is expected from the SUS/SIUS. They should define the results wanted by the stakeholders i.e. what the stakeholders want to be able to do with the SUS/SIUS and the expected quality. However, they should not make any comments or statements about how the SUS/SIUS is to be created or provided.
Eu.ModSt.108	User requirements define the results that the users want, irrespective of any functional breakdown (see *Figure 112*). They must be separate from system requirements and must be defined first.
Eu.ModSt.110	The system requirements must solve the problem of the user, i.e. they must satisfy the user requirements. This has to be approved by means of validation.
Eu.ModSt.112	Figure 112 Differentiating user and system requirements 
Eu.ModSt.1485	The task of defining user requirements encompasses the whole MBSE Process. They are the main source for the creation of the model of an abstract system solution which represents the system requirements.

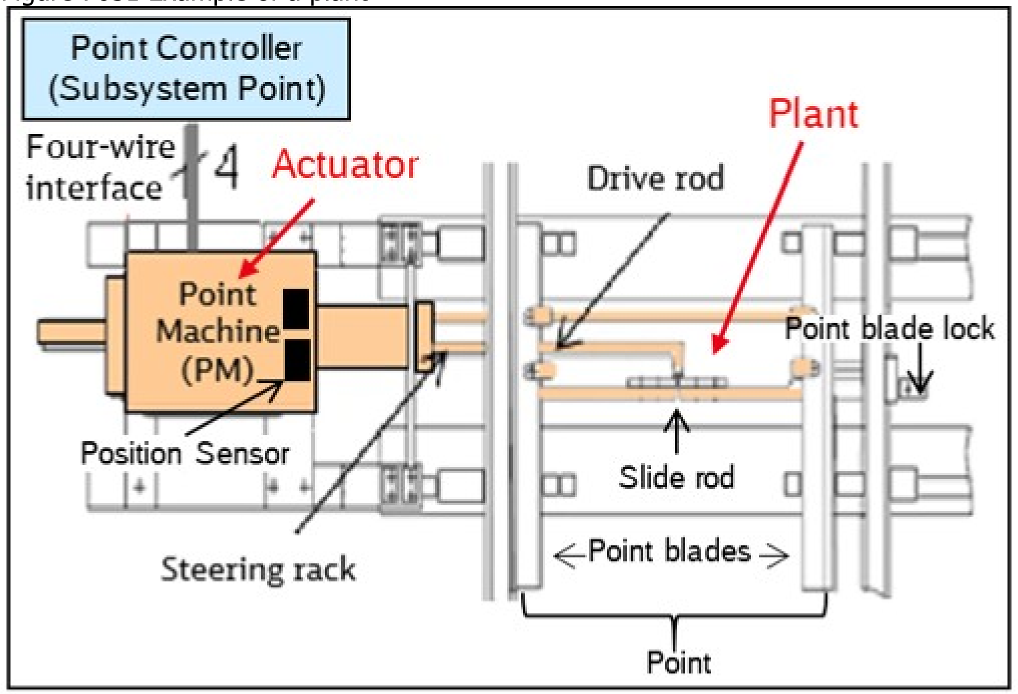
ID	Requirement
Eu.ModSt.316	User requirements should be stated by (or on behalf of) the stakeholders for whom the SUS/SIUS is being developed. Even if the stakeholders do not actually write the user requirements, they should review and when they are happy "endorse" them, and hence take an "ownership" of them.
Eu.ModSt.1473	User requirements may be divided into different classes such as operational requirements, architectural requirements, technical constraints, quality requirements, safety requirements and so on. Safety requirements are an important class of user requirements and thus shortly introduced in *chapter 7.2*. As the main focus of this Modelling Standard is not the elicitation of user requirements, the other different types are not further described.
Eu.ModSt.1474	**7.2 Safety requirements**
Eu.ModSt.1475	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 the following two categories [9]: - Safety invariants: What may not happen under any circumstances, - Safety overrides: Who may do what under which circumstances.
Eu.ModSt.1476	The origin or approach for defining safety requirements can vary. In this section, characteristics of three different methods [26] to create safety requirements are outlined.
Eu.ModSt.1478	**Ad-hoc elicitation** The first it is referred to as **ad-hoc**. Such requirements are specific to a particular system and are based on the design principles for that system. One such requirement for a relay-based interlocking may state that “Front coil of relay L may have current only if relay Ljg has dropped”.
Eu.ModSt.1481	**Regulations-based elicitation** The second is referred to as **regulations-based**. Requirements are based on safety standards, e.g. based on formalising requirements in applicable rules and regulations. One such requirement for an interlocking may state that “a main signal may clear only if there is an established flank protection”, together with appropriate definitions of what "clear" means and what the requirements on flank protection means.
Eu.ModSt.1480	**Hazard-based elicitation** The third is referred to as **hazard-based**. Requirements are based on making an analysis (hazard analysis) of the different types of possible hazards (e.g. frontal collision of trains, derailment and so on) and for each possible hazard, require that it is impossible. Essentially, the purpose of hazard analyses is to identify operational conditions of the SUS's functionality that could lead to harm. The main outputs of such an analysis are hazards and safety goals (i.e. safety requirements).
Eu.ModSt.1482	Safety requirements should be documented separately from other user requirements and incorporated into the system`s requirements artefacts. The complete and correct incorporation of the safety requirements has to be assured using verification methods such as simulation-based falsification methods or formal verification methods [25].
Eu.ModSt.1490	**Simulation-based falsification methods** can work directly on simulation models such as executable SysML state machines. In general, given a safety requirement in some form of logic, these methods leverage mathematical methods, trying to falsify the requirement. This means that the algorithms are geared towards identifying the "worst possible" simulation run with respect to the given requirement. If the method succeeds in producing a run which violates the requirement, it is falsified and the counterexample can be used to refine either the requirement or the simulation model. If it does not, no formal guarantees about the fulfillment of the requirement can be made.
Eu.ModSt.1491	In contrast, **formal verification methods** aim to provide formal proof of the correctness of the requirement for the given model of the SUS/SIUS. Because this proof cannot be provided by simulation alone, a strictly formal model is required.
Eu.ModSt.317	**7.3 Formulation of user requirements**
Eu.ModSt.318	This Modelling Standard does not have the intention to impose obligations how user requirements have to be formulated, but suggests a formulation as textual requirements according to the SysML specification [1].
Eu.ModSt.319	SysML introduces the requirement diagram which provides the means to depict requirements and to relate them to other specification, design or verification models. The requirements can be represented in graphical, tabular, or tree structure formats.
Eu.ModSt.320	The strength and usefulness of a requirement diagram consists in the fact that it allows to easily understand the relations between the requirements and other model elements. The semantics of these relationships and other diagram elements are explained in [1].
Eu.ModSt.321	A requirement can be decomposed into sub-requirements in order to organize multiple requirements as a tree of compound requirements. Moreover, a requirement can be related to other requirements as well as to other elements, such as analysis, implementation, and testing elements (see *Figure 323*).
Eu.ModSt.322	Therefore, a requirement can be generated or extracted from another requirement by using the *derive* relationship. Furthermore, requirements can be fulfilled by certain model elements using the *satisfy* relationship. The verify relationship is used to verify a requirement by applying different test cases.
Eu.ModSt.1479	User requirements (especially safety requirements) should be verifiable, so that it is possible to distinguish a system model satisfying the user requirements from one that does not do. Typical reasons for user requirements not being verifiable include: - The user requirement is incomplete. - The user requirement is poorly written. - The user requirement is not described at the level it will be verified.

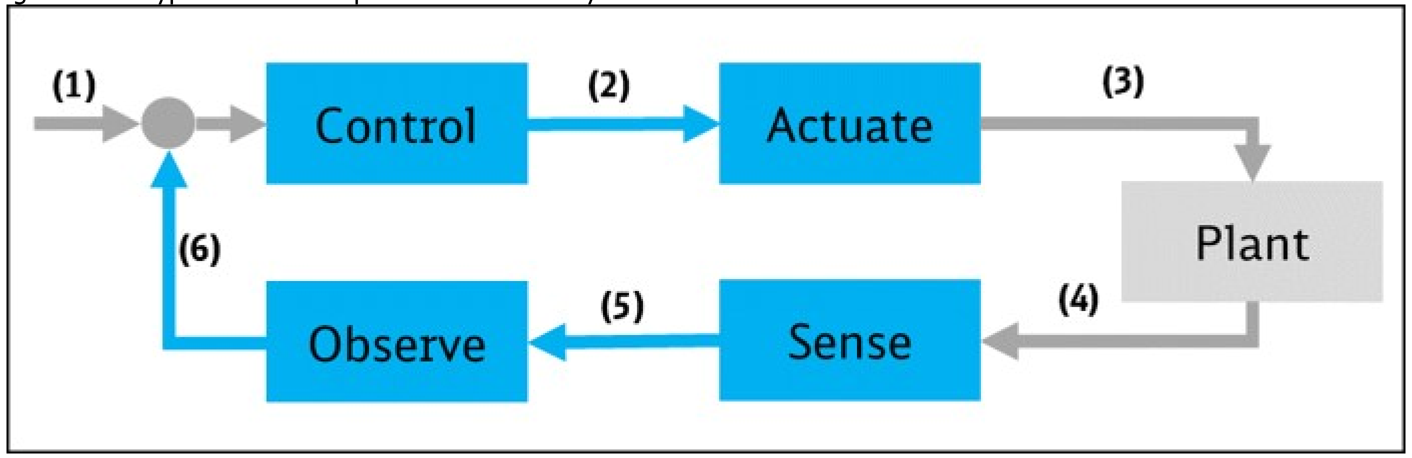
ID	Requirement
Eu.ModSt.323	Figure 323 Requirement diagram example [1] 
Eu.ModSt.332	**8 Architecture Model MBSE**
Eu.ModSt.330	The design decisions derived from the user requirements are documented traceable in the area "Architecture Model MBSE" of the SF MBSE in the form of a model of the abstract solution of a SUS or a SIUS.
Eu.ModSt.335	Focusing on specification tasks to be carried out at infrastructure manager (IM) side, the Architecture Model MBSE (see *Figure 340*) facilitates the description of a SUS or a SIUS from different viewpoints capturing different stakeholder concerns and with varying degrees of granularity (different abstraction levels).
Eu.ModSt.1516	**Viewpoint** A viewpoint is a specification of the conventions for constructing and using a view. Viewpoints comprise patterns or templates from which to develop individual views by establishing the purpose and audience for a view and the techniques for its creation and analysis (based on [29]).
Eu.ModSt.342	**Abstraction level** An abstraction level defines a specific level of abstraction and granularity at which the SUS/SIUS is examined. The level of granularity of the respective abstraction level is in turn determined by a structural characteristic that stems from the layer above. Initially we consider the SUS/SIUS as a whole [25]. In other words, an abstraction level describes the whole of a SUS/SIUS under a certain degree of abstraction, i.e. it represents the amount of complexity by which a SUS/SIUS is viewed. The higher the level, the less detail. Any abstraction level contains several appropriate views.
Eu.ModSt.1561	To change the degree of granularity for a given view to a higher degree, a low degree view is refined into a number of more detailed SUS/SIUS views following the principle of divide and conquer. This step can basically be performed from any viewpoint.
Eu.ModSt.357	**Refinement** Refinement refers to the process of detailing an analysis or design element while preserving its semantics [25]. The degree of abstraction decreases from top to bottom, i.e. the lower the degree of abstraction the higher the degree of refinement of corresponding views.
Eu.ModSt.358	The EULYNX MBSE methodology is based on two basic refinement relations, namely, behavioural and interface refinement. These relations are described as follows [4].
Eu.ModSt.360	**Behavioural refinement** Behavioural refinement relates to specifications of the same syntactic interface. The refined (more precise) specification may impose further functional and non-functional requirements in addition to those imposed by the given (more abstract) specification.
Eu.ModSt.362	**Interface refinement** Interface refinement relates to specifications of different syntactic interfaces. The refined specification is a „behavioural refinement" of the given specification with respect to a translation of its input/output histories. For example, interface refinement allows to replace a message by several messages, and vice versa or instead of transmitting natural numbers, bits may be sent (data refinement).
Eu.ModSt.1520	**Decomposition** In contrast to refinement, decomposition denotes the partitioning of an analysis element or design element, or a logical/technical component into parts [25].
Eu.ModSt.336	**View** A view is a representation of a whole SUS/SIUS from the perspective of a related set of concerns (based on [29]. In other words, a SUS/SIUS description from a specific viewpoint and with a specific degree of granularity is called a view [25]. Within the scope of this Modelling Standard, a view is synonymously referred to as "view", "model view" or "system view".

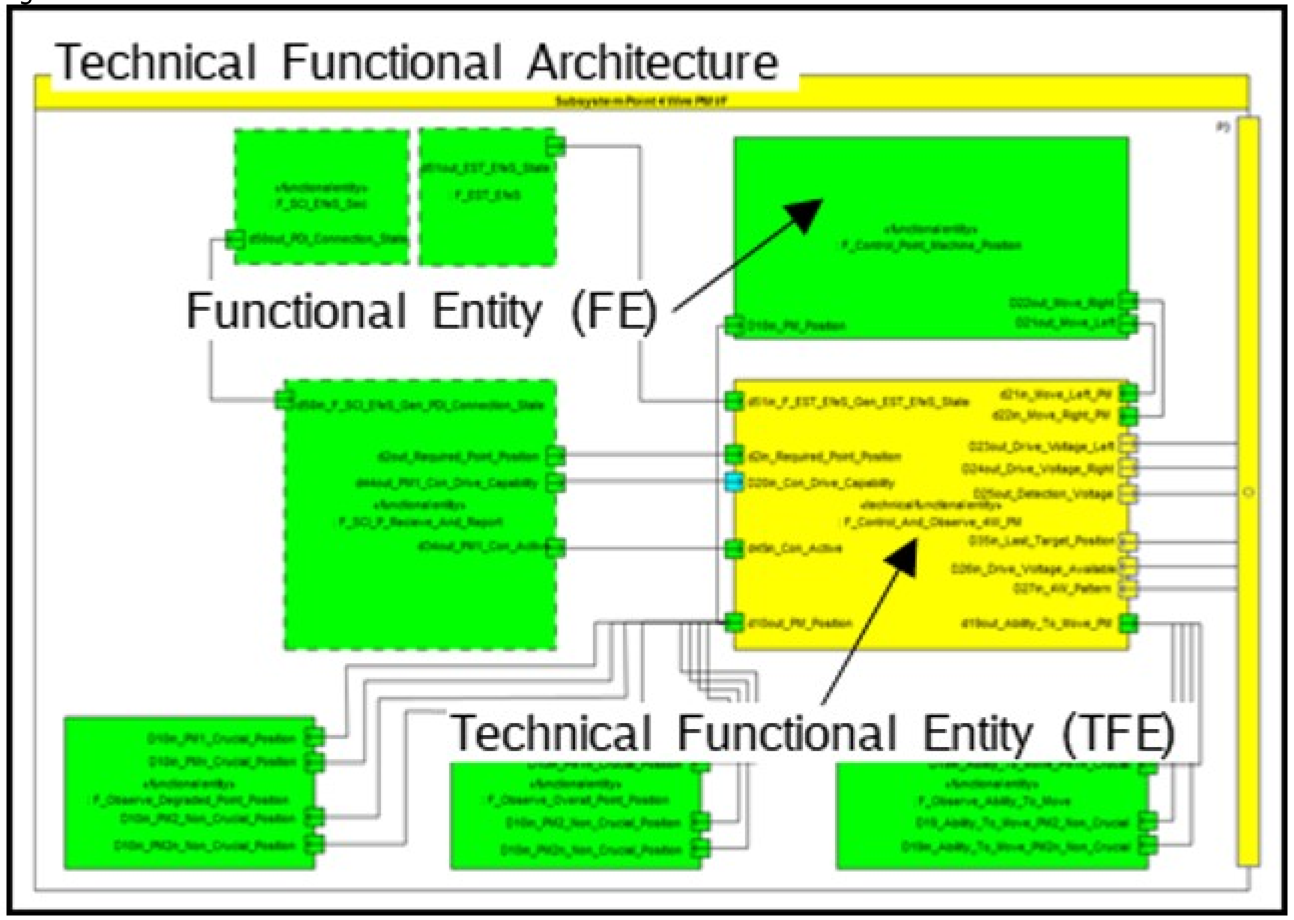
ID	Requirement
Eu.ModSt.1336	**Engineering path** As illustrated in *Figure 340* the development of views for a SUS or SIUS with a specific degree of granularity is summarised in an engineering path.
Eu.ModSt.334	The AM MBSE facilitates the seamless, model based specification of digital CCS in the railway domain with three core IM-related viewpoints namely • **Functional Viewpoint**, • **Logical Viewpoint** and • **Technical Viewpoint**.
Eu.ModSt.331	The viewpoints describe a SUS or a SIUS with respect to different concerns. However, these descriptions may vary in their degree of granularity. For complex SUS/SIUS in particular, it is reasonable to start with rather high-level descriptions. Once these high-level descriptions have been created, these views are typically refined and detailed step by step. Therefore, the AM MBSE supports views with different degrees of granularity i.e. views at different abstraction levels.
Eu.ModSt.333	Following EN 50126 [17] the AM MBSE consists of three core IM-related abstraction levels (AL) namely **AL1: Subsystem/Interface Definition**, **AL2: Subsystem/Interface Requirements** and **AL3: Apportionment of Subsystem/Interface Requirements**.
Eu.ModSt.3561	The AM MBSE can also be applied to specify an overall system, which is not the case in EULYNX at the moment. In this case, the abstraction levels are named as follows: AL1: System Definition, AL2: System Requirements and AL3: Apportionment of System Requirements.
Eu.ModSt.1526	Each of the IM-related core AL may again be decomposed in further AL such as AL1.1, AL1.2 and so on as appropriate. Any AL represents design decisions about the refined or decomposed implementation of its predecessor and the specification of the outcome of this decisions by means of appropriate views.
Eu.ModSt.1525	**Crosscutting system properties (CSP)** One of the principles of the AM MBSE is the continuous engineering of crosscutting system properties. This principle aims at establishing the ability to consider crosscutting properties of the SUS/SIUS. Typical crosscutting properties are RAMS [17], security and real-time properties of the SUS/SIUS: they must be considered in any engineering activity and the corresponding artefacts [25].
Eu.ModSt.337	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.
Eu.ModSt.1242	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 have to be integrated as tightly as possible into the development process and its models [25].

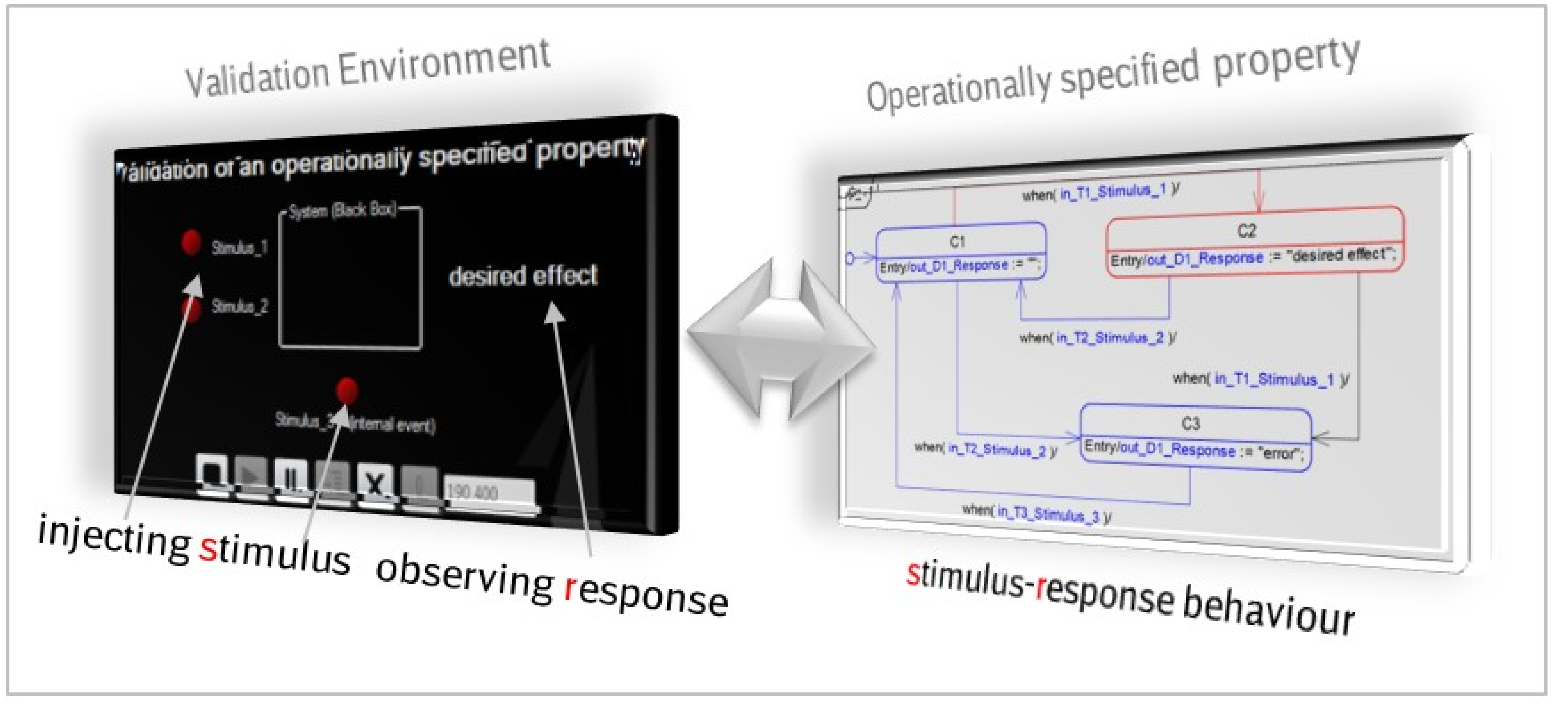
ID	Requirement
Eu.ModSt.340	Figure 340 Architecture Model MBSE 
Eu.ModSt.879	**8.1 Overview of the EULYNX MBSE methodology**
Eu.ModSt.2110	The EULYNX initiative is aiming at specifying EULYNX subsystems and standardising their interfaces (SCI, SMI, SDI) and the interfaces between adjacent systems.
Eu.ModSt.1663	This chapter provides an overview of the used MBSE methodology. The EULYNX MBSE methodology assumes that a definition of the EULYNX architecture is known. Thus, it is currently not designed to describe system architectures but black-box specification models of EULYNX subsystems, their standardised interfaces and standardised interfaces between adjacent systems.
Eu.ModSt.7012	**8.1.1 Characteristics of EULYNX subsystems**
Eu.ModSt.7014	Command control and signalling (CCS) systems such as EULYNX subsystems are reactive control systems [32] and most of them safety-critical [11]. They are characterized by the constant interaction and synchronisation between the system and its environment.
Eu.ModSt.88	The terms "system" and "reactive system" shall be explained first.
Eu.ModSt.7702	**8.1.1.1 System**
Eu.ModSt.84	A system is a technical or a sociological structure consisting of a group of entities combined to form a whole that can work, function, or move interdependently and harmoniously. A system may consist of various system elements called subsystems, that can be understood as systems on their own. Systems are thus hierarchically divided into subsystems [4]. Since the single system is, in turn, a part of a larger system, one may speak of an embedded system [5].

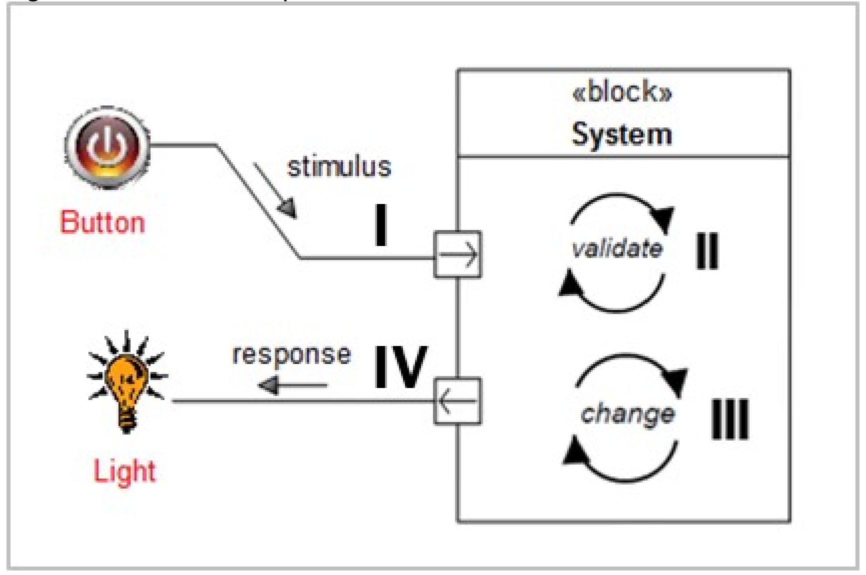
ID	Requirement
Eu.ModSt.86	EULYNX follows the objective of structuring the EULYNX overall CCS system hierarchically into subsystems in a way, that the resulting subsystems, referred to as modules, can be supplied by different suppliers and then integrated independent of a particular vendor [12]. As far as the specification of those modules, such as a Subsystem Light Signal, a Subsystem Point, a Subsystem LX and so on is concerned, they are fitted with standardised interfaces and seen as black boxes without any further decomposition.
Eu.ModSt.7059	**8.1.1.2 Reactive system**
Eu.ModSt.1496	A reactive system is a system that, when switched on, is able to create desired effects in its environment by enabling, enforcing, or preventing events in the environment.
Eu.ModSt.89	Following the deterministic paradigm which is a key requirement for a safety-critical railway system, in contrast to non-deterministic systems, the same sequence of system inputs always produces the same sequence of system outputs.
Eu.ModSt.1497	Safety is a major quality of safety-critical railway systems that must be considered in any activity during engineering. Safety can be characterized as the extent to which the SUS will not have effects on its environment that result in harm to people, significant monetary losses, or any other negative impacts to its environment [25].
Eu.ModSt.90	Reactive systems have a number of characteristics [8]: • The system is in continuous interaction with its environment. • The process by which the reactive system interacts with its environment is usually nonterminating. If a reactive system terminates during its availability time, this is usually considered a failure. • In its interaction with the environment, the system will respond to external stimuli as and when they occur. The system must therefore be able to respond to interrupts, even if it is doing something else. • The response of a reactive system depends on its current state and the external event that it responds to. The response may leave the system in a different state than it was before. • The response consists of enabling, enforcing, or preventing interaction with its environment. • The behaviour of a reactive system often consists of a number of interacting processes that operate in parallel. • Often a reactive system must operate in real time and under stringent time requirements.
Eu.ModSt.91	Although reactive systems may provide manifold functionality, they all engage in stimulus-response behaviour. Thus, for the specification of a reactive system appropriate techniques are needed for specifying stimulus-response behaviour.
Eu.ModSt.1499	For the specification of the stimulus-response behaviour of a safety-critical railway system such as an interlocking system that may be described by discret states, finite state machines such as SysML state machines may be used.
Eu.ModSt.1498	Similar to the characteristics of reactive systems are the characteristics of interactive systems. While for reactive systems the stimulus-response behaviour is determined by the physical-technical environment, the stimulus-response behaviour of interactive systems is determined by the system.
Eu.ModSt.93	Reactive systems or interactive systems can be contrasted with transformational systems [8], which exist to transform an input into an output. A diagnostic expert system, for example, is a transformational system; it may enter an interactive dialogue to acquire all relevant data about a malfunctioning system, but when all data is provided, the expert system will produce its diagnosis as output and terminates.
Eu.ModSt.7015	Since a EULYNX subsystem also has the characteristic of a control system, this term shall be explained next.
Eu.ModSt.7016	**8.1.1.3 Control system**
Eu.ModSt.7017	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.
Eu.ModSt.7018	If a lamp is switched ON or OFF using a switch, according to the example shown in *chapter 8.1.3*, 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.
Eu.ModSt.7019	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.
Eu.ModSt.7020	**8.1.1.4 Typical control loop of a EULYNX subsystem**
Eu.ModSt.7021	*Figure 7022* shows a typical control loop of a CCS system such as a EULYNX subsystem. The "Plant" is the system being controlled such as the point in the environment of the control system consisting of point controller and point machine (see *Figure 705*1).

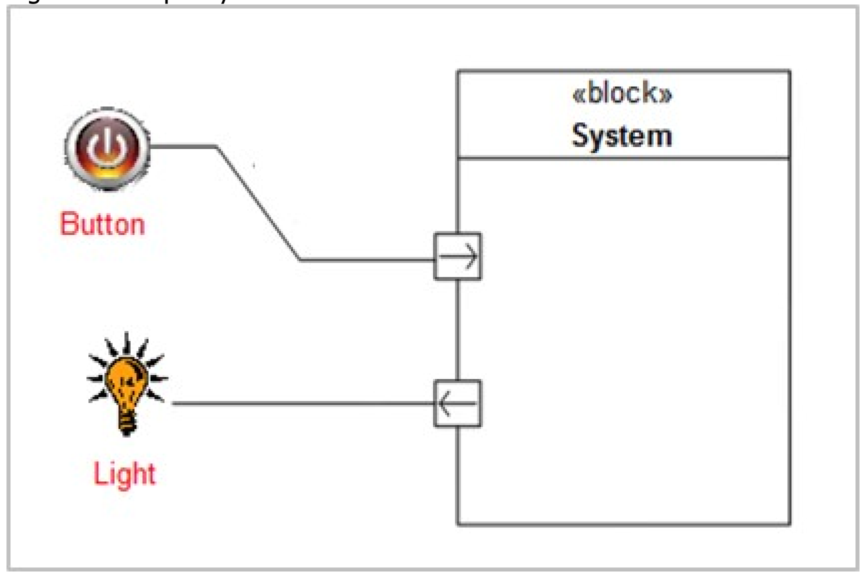
ID	Requirement
Eu.ModSt.7051	Figure 7051 Example of a plant 
Eu.ModSt.7023	Most core control system functions can be assigned to one of the four categories listed below: • **Control**: the purpose of a control function is to transform information about a needed change of the plant's state into instructions 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**.**
Eu.ModSt.7024	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.
Eu.ModSt.7025	The point state (LEFT, RIGHT or TRANSITION) of a railroad turnout, for example, is an internal state. It can be inferred by sensing the current flow via the point machine position sensor contacts. From these sensed current flow, we can infer the internal state that is the point state of the turnout.
Eu.ModSt.7026	*Figure 7022* shows the flow of information between the functions [(2), (5), (6)] within the control system and between them and an external reference (1) and the "Plant" [(3), (4)] using a railroad turnout as an example. The information flows (4), (5) and (6) correspond to the "feedback" of a closed loop control system as described in [32]. The information flows are described below: (1) Required internal state of "Plant": e.g. required point state "LEFT", (2) Required external state of "Plant": e.g. connected voltage for moving the point machine to the left (four-wire interface), (3) Actual external input state of plant: e.g. movement of the point machine drive rod to bring the point into the left position, (4) Actual external output state of plant: e.g. switching position of the point machine position sensor contacts depending on the point machine drive rod position, (5) Sensed external output state of plant: e.g. current flow via the point machine position sensor contacts (four-wire interface), (6) Estimated internal state of plant: e.g. estimated point state "RIGHT" or "TRANSITION.

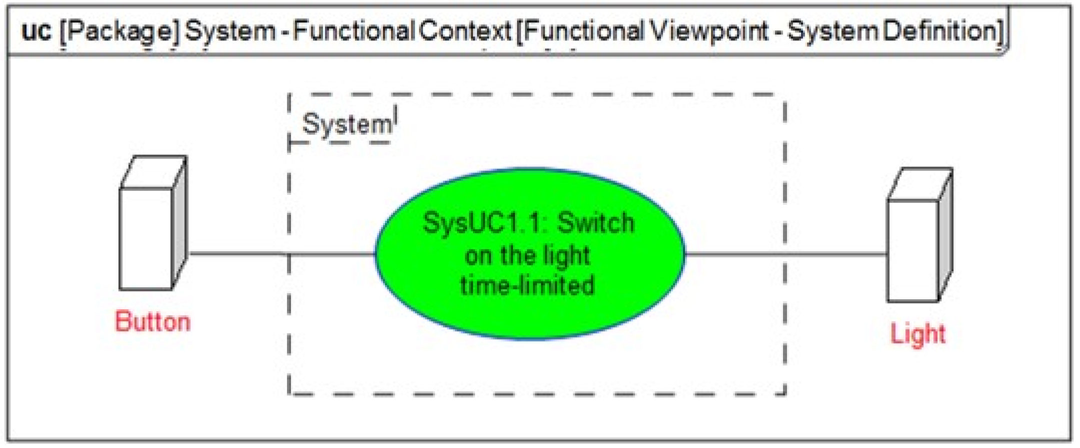
ID	Requirement
Eu.ModSt.7022	Figure 7022 Typical control loop of a EULYNX subsystem 
Eu.ModSt.7052	**8.1.1.5 Interpretation of the concept of "Function"**
Eu.ModSt.201	According to the EULYNX MBSE approach, use cases form the basis for the functions to be provided by a SUS at the highest level of abstraction, i.e. at abstraction level AL1 of the AM MBSE. They describe the functionality of a SUS in terms of how it is used to achieve the goals of its various users (see *chapter 8.1.2.2.3*). In other words, use cases create desired effects in the SUS environment.
Eu.ModSt.7699	In contrast to a use case, a function is the ability of a SUS to create a desired effect in the system environment. So all use cases of a SUS are functions and each function realises one or more UseCases [8].
Eu.ModSt.7053	At abstraction level AL2 of the AM MBSE, a function is represented by a Functional Entity (FE) or a Technical Functional Entity (TFE). Both encapsulate subsets of functional requirements of EULYNX SUSs or SIUSs in the form of function modules. They delimit the function modules from their environments and define the inputs and outputs.
Eu.ModSt.7058	While FEs define technology-independent functional requirements derived from corresponding use cases defined on abstraction level AL1, TFEs describe technology-dependent ones.
Eu.ModSt.7056	FEs and TFEs have SysML state machines and SysML block operations to describe behaviour. 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. 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).
Eu.ModSt.7057	The EULYNX specification approach allows the description of functional control system architectures and their governing control loops through the "Functional Architecture" and "Technical Functional Architecture" model views of AM MBSE. As exemplified in Figure 7055, the functions of a control system are represented by interconnected FEs or TFEs.
Eu.ModSt.7321	**Please note:** FEs and TFEs are used for the structured description of a SUS or SIUS and are not in themselves architectural specifications for the manufacturer. In other words, a manufacturer does not have to prove that it implements a particular FE or TFE. Proof is only required for the overall behaviour defined by the interconnected FEs or TFEs in a functional or technical functional architecture.

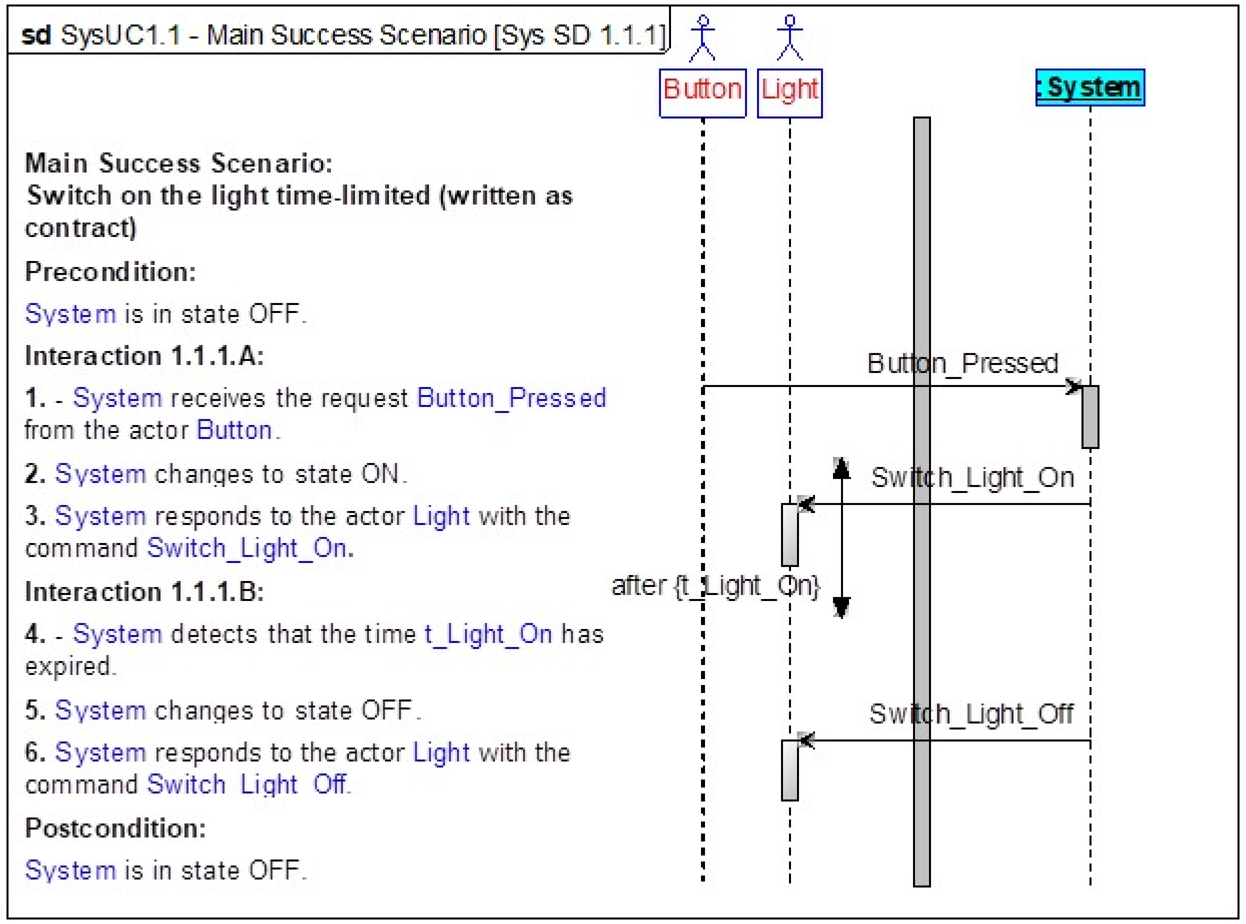
ID	Requirement
Eu.ModSt.7055	Figure 7055 FE and TFE in a Technical Functional Architecture 
Eu.ModSt.2041	**8.1.2 Principle of model-based definition of requirements**
Eu.ModSt.2061	**8.1.2.1 Applied description methods for model-based requirements**
Eu.ModSt.2044	To best support the verification and validation effort of specified SUS/SIUS requirements and to keep the specification understandable for engineers, the EULYNX specification approach aims to describe the functional SUS/SIUS requirements in the form of operational specifications.
Eu.ModSt.2047	As mentioned above, the CCS systems currently specified in EULYNX are reactive control systems and characterised by the constant interaction and synchronisation between the system and its environment.
Eu.ModSt.2048	A reactive control system, when switched on, engages in stimulus-response-behaviour in order to create desirable effects in its environment. For that reason, the EULYNX methodology proposes the specification of the functional system requirements in stimulus-response form.
Eu.ModSt.2042	As the focus of EULYNX is on the specification of interfaces, the behaviours of EULYNX systems are specified using an interface centric approach.
Eu.ModSt.2111	In the following sections, the concepts of "operational specification", "stimulus-response specification" and "interface centric approach" are explained.
Eu.ModSt.2043	**8.1.2.1.1 Operational specification**
Eu.ModSt.2045	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.
Eu.ModSt.2046	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).
Eu.ModSt.7067	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.
Eu.ModSt.114	For an operationally specified functional system property, there is a test that they can all perform and agree on the outcome - either the SUS/SIUS to be specified does or does not satisfy this property (see *Figure 115*).

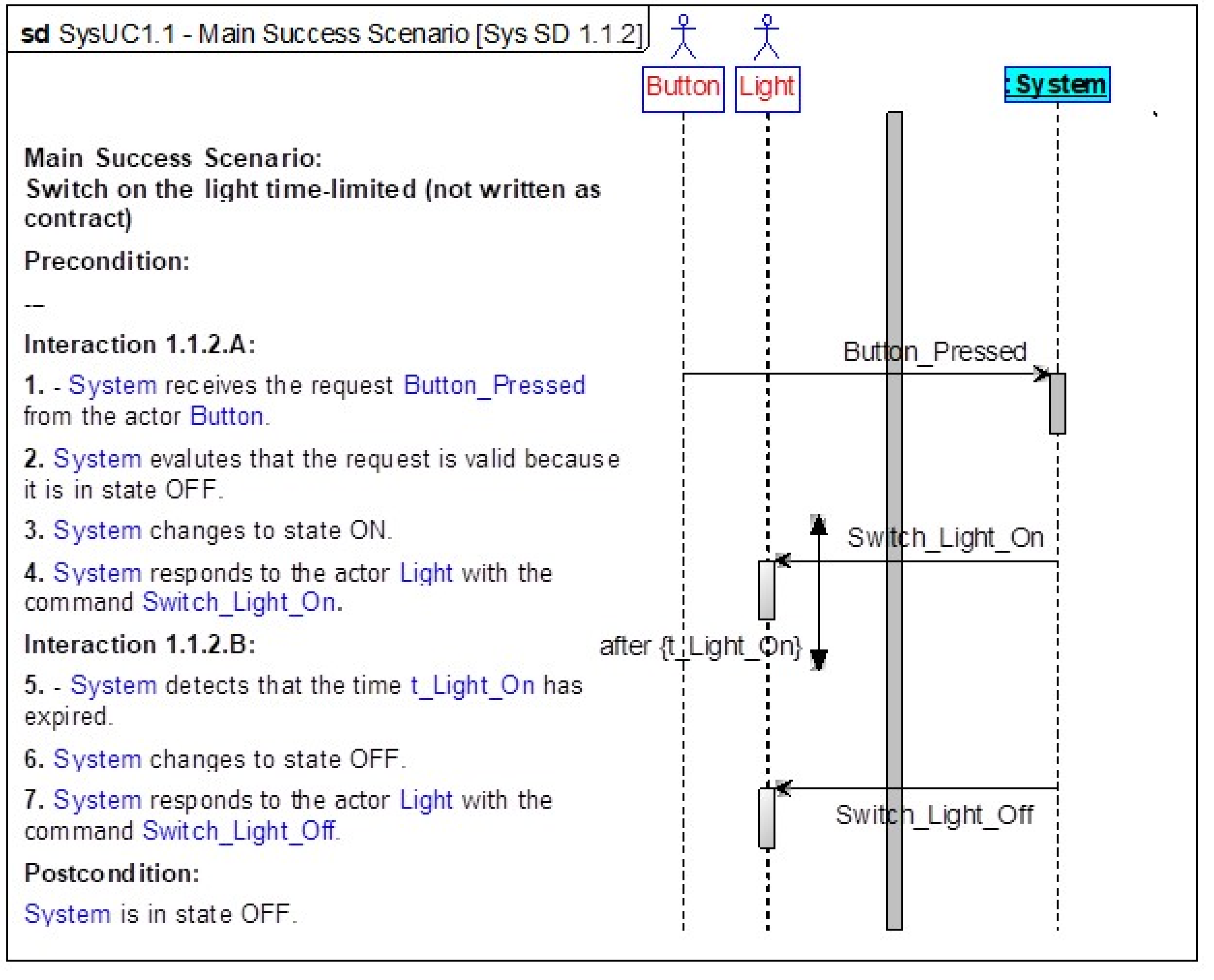
ID	Requirement
Eu.ModSt.7068	Whether an operational specification exhibits a specific property may often-case be easy to determine but it may also offer a challenge, for various reasons. To determine if a property holds or not can be non-trivial due to e.g., specification complexity that may prevent inspection alone, state-space explosion impacting the results attainable in automated analysis, and semantics for interpretation that can complicate analyses.
Eu.ModSt.7069	In general, it is desirable to have an implementation-independent operational specification, so that all stakeholders can agree on and use the same specification. The reason for this is to avoid, when the SUS/SIUS is delivered, that supplier and customer dispute about whether SUS/SIUS meet the desired properties or not. In general, it is recommended that SUS/SIUS specifications are operationalised as much as possible [8].
Eu.ModSt.115	Figure 115 Test of an operationally specified system property 
Eu.ModSt.7066	**8.1.2.1.2 Stimulus-response specification**
Eu.ModSt.7070	Stimulus-response specifications are an important class of operational specifications.
Eu.ModSt.2049	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**.
Eu.ModSt.2050	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. Please note: this is also said to be a response if a stimulus occurs and the system "keeps quiet".
Eu.ModSt.2051	A single stimulus-response pair is henceforth also referred to as an interaction.
Eu.ModSt.2052	An **interaction** is generally formulated according to the following action block schema comprising four action steps (see *Figure 173*): **Interaction:** **I.** - The SUS or SIUS receives a stimulus. **II.** The SUS or SIUS validates the stimulus. **III.** The SUS or SIUS changes its internal state (or not). **IV.** The SUS or SIUS responds with the result (Please note: a result may also be that the SUS or SIUS "keeps quiet"). However, there may be more than four action steps applied or fewer.

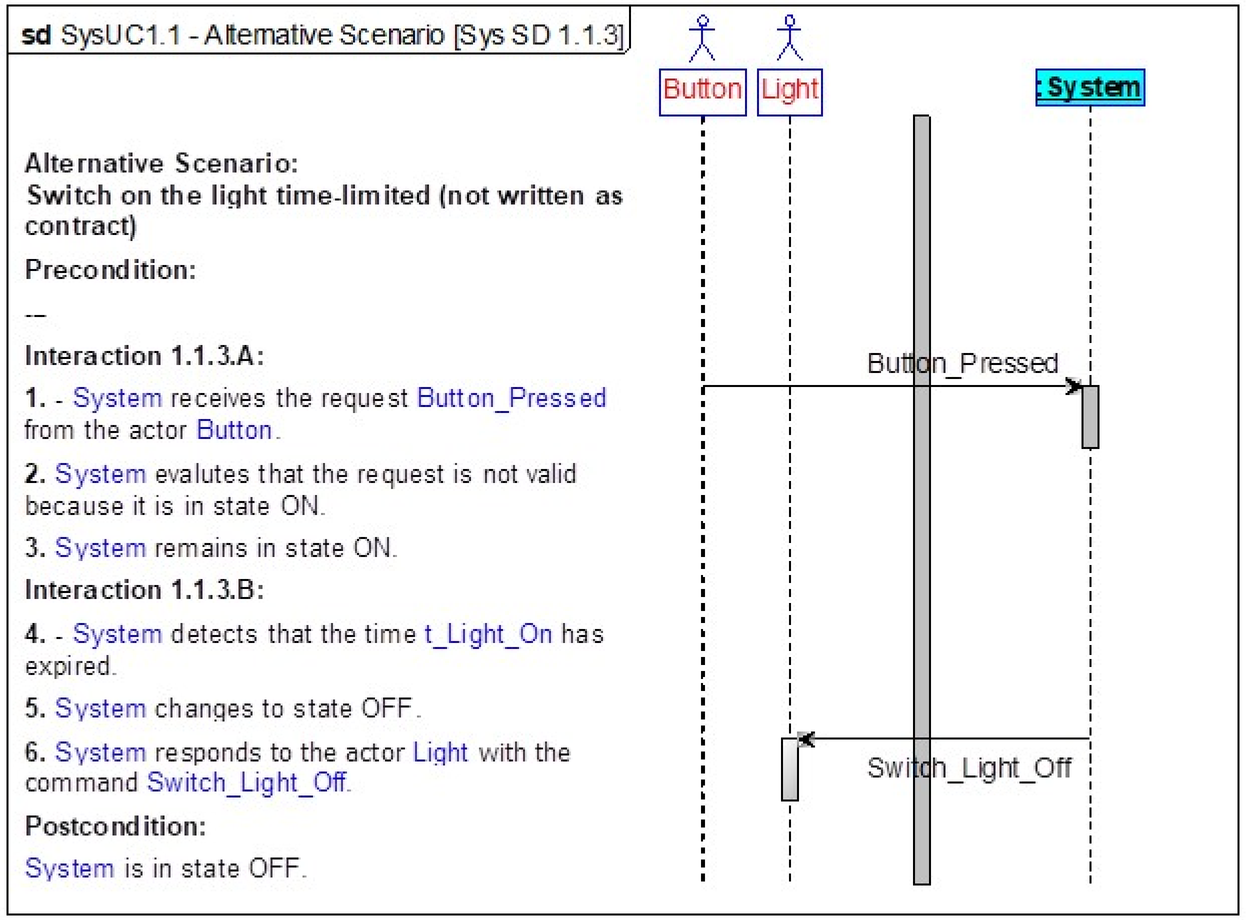
ID	Requirement
Eu.ModSt.173	Figure 173 The four steps of an action block 
Eu.ModSt.2053	An interaction always starts with the stimulus identified by a dash "-" (see step I in ID 355 above). A stimulus may have its origin • in the **request of a primary actor** (a primary actor is an actor in the environment of the SUS or SIUS who requires a service from it), • in a **timed trigger**, • in an **internal trigger** (that is, an event that occurs in the system) or • in the **entering or leaving a system state**.
Eu.ModSt.2054	**Interactions** may be extended to **contracts**.
Eu.ModSt.2055	The central idea of contracts is a metaphor on how the SUS or SIUS and the actors collaborate on the basis of mutual obligations and benefits. Having written functional requirements in the style of interactions, those contracts can easily be obtained - interactions together with pre- and postconditions.
Eu.ModSt.2056	If a SUS or SIUS provides a certain functionality, it may **a)** expect a certain condition to be guaranteed on entry by an actor that sends the request: the precondition of the interaction - an obligation for the actor, and a benefit for the SUS or SIUS, as it relieves it from having to handle the cases outside of the precondition. **b)** guarantee a certain property on exit: the postcondition of the interaction - an obligation for the system, and obviously a benefit (the main benefit of the request) for the actor.
Eu.ModSt.2057	The following applies for preconditions and postconditions in this context: **a)** The interaction may only be triggered by the actor if the precondition is met; this presupposes that the actor knows the current system condition, **b)** The system must ensure in turn that the postcondition is met after the completion of the interaction. If no explicit postcondition has been defined (indicated by three dashes "---"), the requirement applies that the postcondition is identical to the precondition.
Eu.ModSt.2058	A **contract** is formulated according to the following schema: **Precondition**: Definition of the precondition **Interaction:** **I.** - The SUS or SIUS receives a stimulus. **III.** The SUS or SIUS changes its internal state (or not). **IV.** The SUS or SIUS responds with the result (Please note: a result may also be that the SUS or SIUS "keeps quiet"). **Postcondition:** Definition of the postconditions
Eu.ModSt.2059	Alternatively to this, functional system requirements may be written without using **contracts**. In these cases it can not be assumed that the actor knows the current SUS or SIUS condition and complies with the precondition. The preconditions of the interactions are empty and the SUS or SIUS must first check on itself whether the preconditions are met before responding to the stimulus. The above schema is modified as follows (see text in italics): **Precondition**: ---

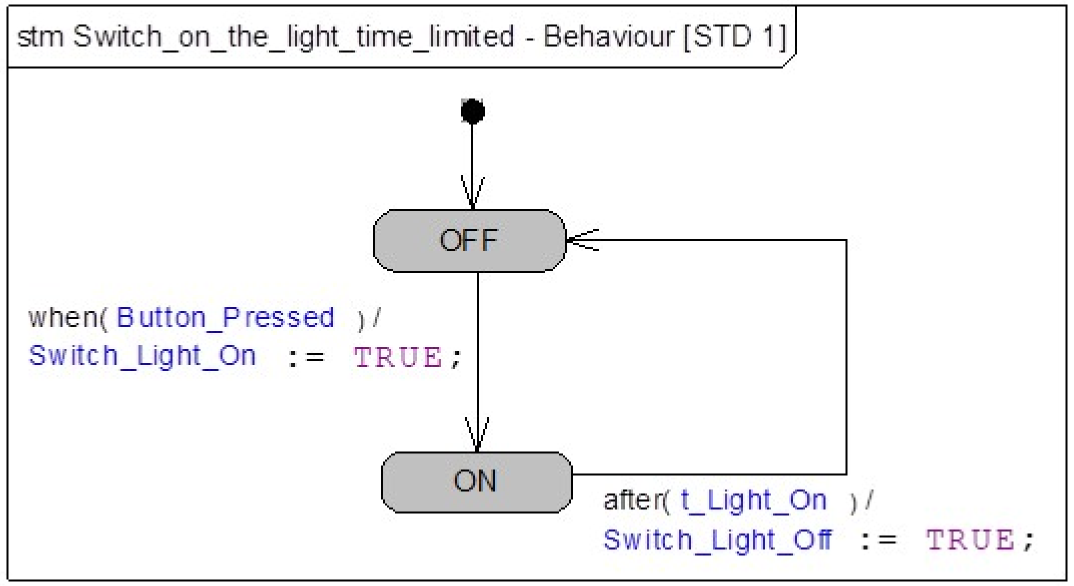
ID	Requirement
	Interaction: **I.** - The SUS or SIUS receives a stimulus. **II.** *The SUS or SIUS validates the stimulus considering the current internal state.* **III.** The SUS or SIUS changes its internal state (or not). **IV.** The SUS or SIUS responds with the result (Please note: a result may also be that the SUS or SIUS "keeps quiet"). **Postcondition:** Definition of the postconditions
Eu.ModSt.2060	In those cases, the check may fail in the second step. From this step on, a different internal condition might need to be entered and a different response might need to take place. **Variants of the interaction** would therefore have to be considered.
Eu.ModSt.2062	**Interactions** and **contracts**, as defined above, provide the basic schemata for the model-based description of functional system requirements in **stimulus-response form**. Depending on the abstraction level two model-based description methods are used: • **Use case scenarios** (interaction scenarios) are used at abstraction level AL1 Subsystem Definition defining the interaction of the subsystem with its environment. • **State machines** are used at abstraction level AL2 Subsystem Requirements completely refining the externally visible stimulus-response behaviour described by means of the use case scenarios at abstraction level AL1 Subsystem Definition.
Eu.ModSt.2063	These two model-based description methods will be demonstrated defining the functional system requirements of a simple system based on the **functional user requirements (FUR)** listed below: **FUR1:** The user wants to switch on the light by pressing a button if the light is off, **FUR2:** The user wants the light to be switched off automatically after a defined time.
Eu.ModSt.2064	As shown in *Figure 3* the **SUS** named "**System**" is connected to the two actors "**Light**" and "**Button**" in the environment.
Eu.ModSt.2065	Figure 3: Simple system 
Eu.ModSt.2066	According to the functional user requirements described above the SUS is required to fulfil the functional system requirements (FSR), described in classical textual form below: **FSR1:** The system shall switch on the light if the light is switched off and the button is pressed, **FSR2:** The system shall switch off the light automatically after the time t_Light_On has expired.
Eu.ModSt.2067	**8.1.2.1.3 Description method using use case scenarios**
Eu.ModSt.2068	The functional user requirements **FUR1** and **FUR2** defined above (see ID 215) require the SUS "System" to provide a service for the users. As shown in *Figure 2070*, this service is defined as system use case "SysUC1.1: Switch on the light time-limited".

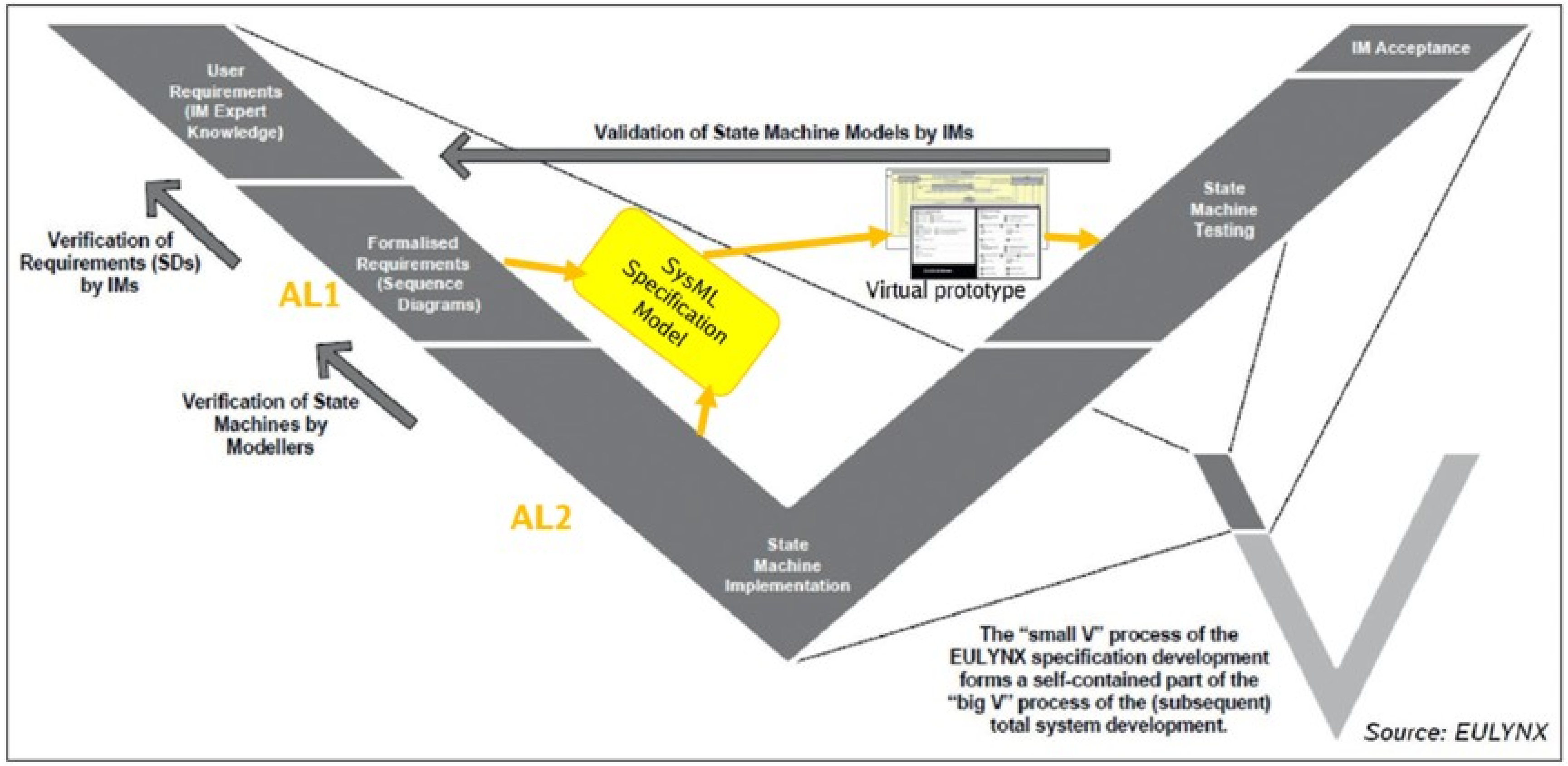
ID	Requirement
Eu.ModSt.2069	System use cases describe the functionality of a SUS or SIUS in terms of how it is used to achieve the goals of its various users. The users of a SUS or SIUS are described by actors (i.e. "Button" and "Light"), which may represent external systems or humans who interact with the system. A UseCase is denoted by an ellipse, and the actors participating in the UseCase are connected to the ellipse by solid lines.
Eu.ModSt.184	On the original work on UseCases by Ivar Jacobson, Jacobson defines a UseCase as follows [20]: *„A use case is a sequence of transactions performed by a system, which yields an observable result of value for a particular actor. A transaction consists of a set of actions performed by a system and is invoked by a stimulus from an actor to the system, or by a timed trigger within the system".*
Eu.ModSt.186	To understand transactions in the database sense is too narrow, because if a transaction succeeds then changes are made to the system (committed), otherwise the system is reverted to the original state (rollback).
Eu.ModSt.187	Cockburn interprets in his book [22] what Jacobson [20] means by a transaction in the four steps of an action block (see *Figure 173*) representing an interaction.
Eu.ModSt.189	The flow between the trigger and the result of a use case has a time coherence, i.e. no domain interruption is possible.
Eu.ModSt.2070	Figure 2070: UseCase shown in a UseCase diagram 
Eu.ModSt.2071	A complete use case, i.e. a primary UseCase consists of one or multiple interactions which can alternatively be formulated as contracts. A UseCase having only one interaction is an interaction written as a use case.
Eu.ModSt.2072	The interactions specifying a UseCase such as "SysUC1.1: Switch on the light time-limited" are described in a model-based way by use case scenarios. Use case scenarios are represented by SysML sequence diagrams.
Eu.ModSt.2073	The specification of the use case scenarios may cover a standard sequence and one or several alternative sequences, e.g. to represent a failed validation of the stimulus. Normally, the "good case" of an use case scenario is specified in the "standard sequence" and deviating sequences in "alternative sequences". If no unique standard sequence can be determined, it is also possible that only "alternative sequences" exist.
Eu.ModSt.2074	For this reason, a use case may be defined by use case scenarios in the following compositions: - **one** Main Success Scenario and any number of Alternative scenarios, - **only one** Main Success Scenario, - **any number** of Alternative Scenarios without a Main Success Scenario.
Eu.ModSt.2075	Several interactions may be combined directly after each other without explicitly depicting the pre- and postconditions between them in an interaction scenario if the postconditions of the previous interaction are identical to the preconditions of the subsequent interaction.
Eu.ModSt.2076	If it can be assumed that the current state of the SUS is visible in its environment, the textually formulated functional requirements **FSR1** and **FSR2** (see ID *Eu.ModSt.2066*) can be described as contracts: **FSR1:** **Precondition:** System is in state OFF **Interaction:** **I.** - System receives the request "Button_Pressed" from the actor "Button". **III.** System changes to state "ON". **IV.** System responds to the actor "Light" with the command "Switch_Light_On". **Postcondition:** System is in state ON **FSR2:**

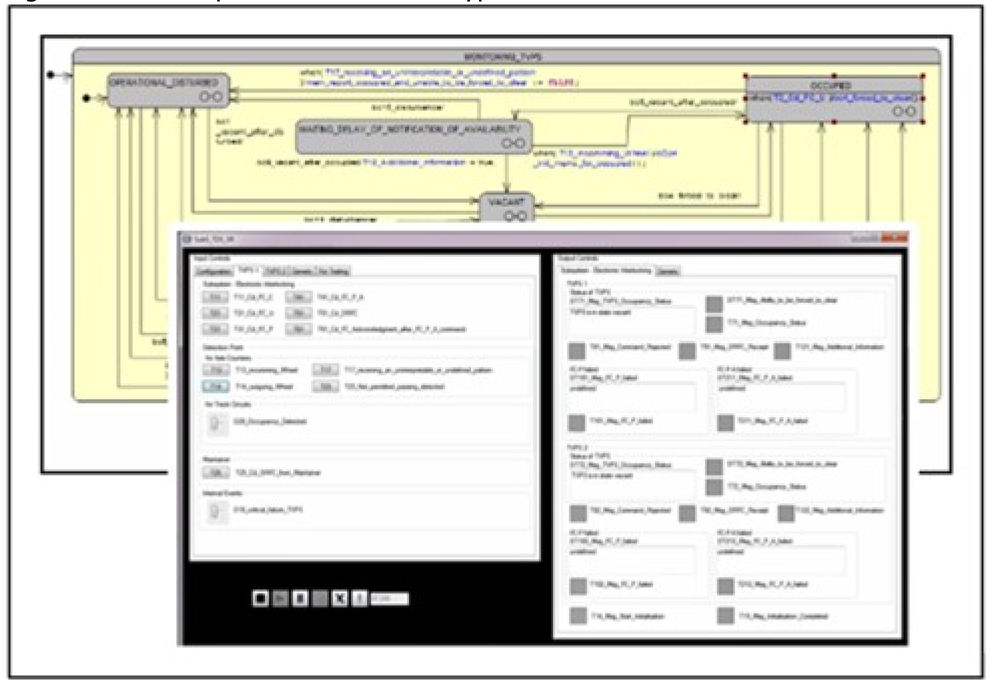
ID	Requirement
	Precondition: System is in state ON **Interaction:** **I.** - System detects that the time "t_Light_ON" has expired. **III.** System changes to state "OFF". **IV.** System responds to the actor "Light" with the command "Switch_Light_OFF". **Postcondition:** System is in state OFF
Eu.ModSt.2077	The corresponding use case scenario in the form of a Main Success Scenario is depicted in *Fgure 2078*. FSR1 and FSR2 are written as contracts and as a consequence no Alternative Scenarios are required. As the precondition of FSR2 is identical to the postcondition of FSR1 they are not explicitly depicted in the use case scenario.
Eu.ModSt.2078	Figure 2078 Main Success Scenario with FSR1 and FSR2 written as contracts 
Eu.ModSt.2079	If it can not be assumed that the current state of the SUS is visible in its environment, the textually formulated functional requirement **FSR1** is to be described as interaction without precondition. **FSR2** may be described as contract because the interaction is internally time-triggered and it is required that the current state may only be changed by this trigger: **FSR1:** **Precondition:** --- **Interaction:** **I.** - System receives the request "Button_Pressed" from the actor "Button". **II.** System evaluates that the request is valid because it is in state OFF. **III.** System changes to state "ON". **VI.** System responds to the actor "Light" with the command "Switch_Light_On". **Postcondition:**

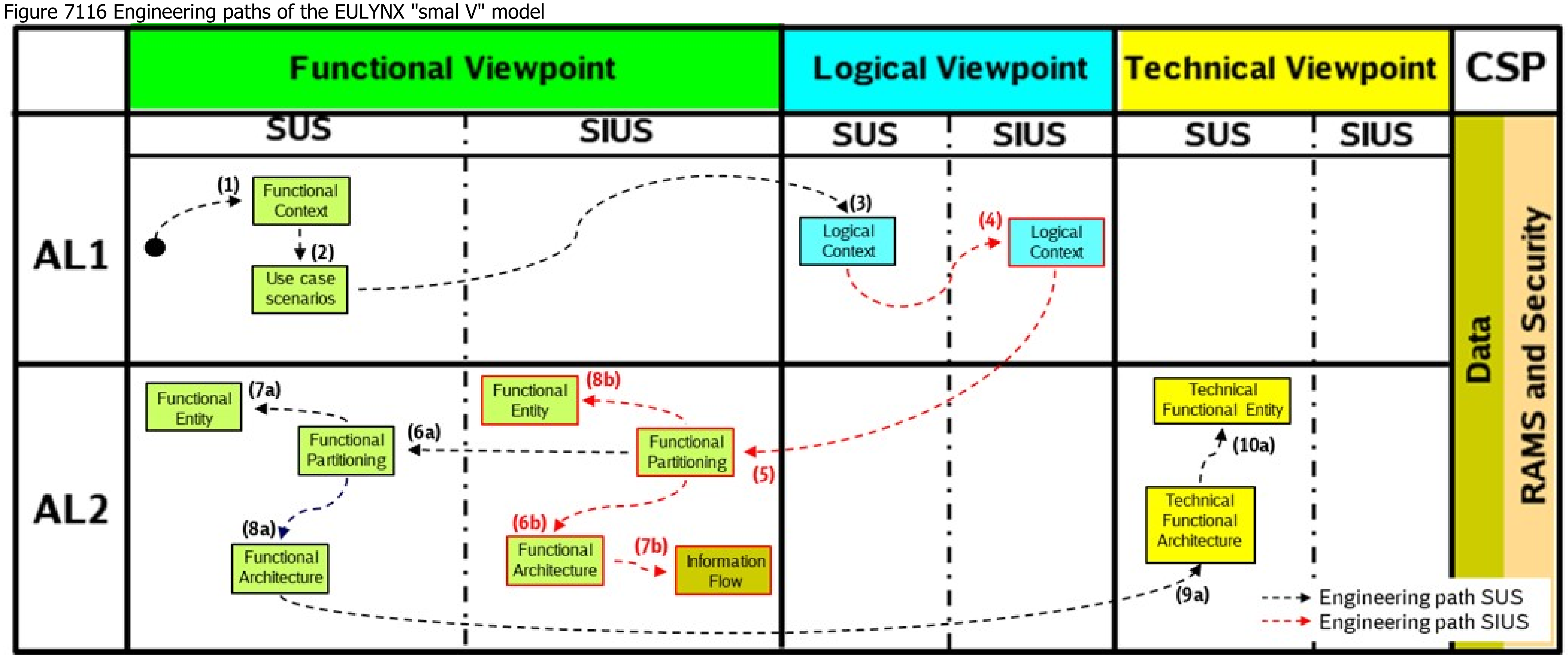
ID	Requirement
	System is in state ON **FSR2:** **Precondition:** System is in state ON **Interaction:** **I.** - System detects that the time "t_Light_ON" has expired. **III.** System changes to state "OFF". **IV.** System responds to the actor "Light" with the command "Switch_Light_OFF". **Postcondition:** System is in state OFF
Eu.ModSt.2080	The corresponding use case scenario in the form of a Main Success Scenario is depicted in *Figure 2081*.
Eu.ModSt.2081	Figure 2081 Main Success Scenario with FSR1 not written as contract 
Eu.ModSt.2082	As FSR1 is not written as a contract, action step 2 of the corresponding interaction may be evaluated as not valid. As a consequence, an alternative variant of the interaction has to be described: **FSR1:** **Precondition:** --- **Interaction:** **I.** - System receives the request "Button_Pressed" from the actor "Button". **III.** System evaluates that the request is not valid because it is in state ON.

ID	Requirement
	IV. System remains in state "ON". **Postcondition:** System is in state ON **FSR2:** **Precondition:** System is in state ON **Interaction:** **I.** - System detects that the time "t_Light_ON" has expired. **III.** System changes to state "OFF". **IV.** System responds to the actor "Light" with the command "Switch_Light_OFF". **Postcondition:** System is in state OFF
Eu.ModSt.2083	The corresponding use case scenario in the form of an Alternative Scenario is depicted in *Figure 2084*.
Eu.ModSt.2084	Figure 2084 Alternative Scenario 
Eu.ModSt.2085	**8.1.2.1.4 Description method using state machines**
Eu.ModSt.2086	**State machines** are used at abstraction level AL2 System Requirements to completely refine the stimulus-response behaviour which has been described by means of the use case scenarios at abstraction level AL1 System Definition.
Eu.ModSt.2087	*Figure 2088* shows a state machine specifying the stimulus-response behaviour of the UseCase "SysUC1.1: Switch on the light time-limited".

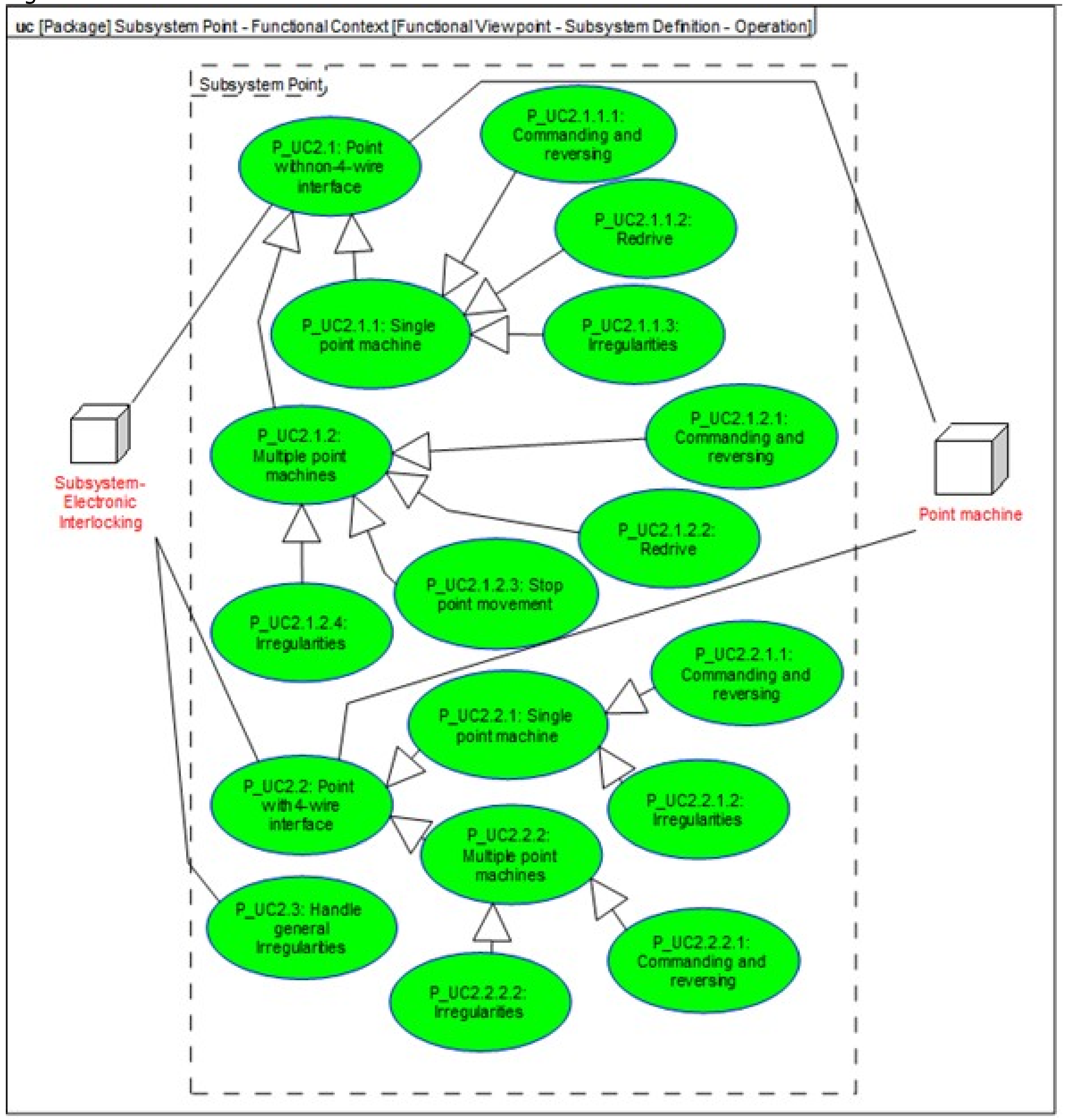
ID	Requirement
Eu.ModSt.2088	Figure 2088 FSR1 and FSR2 specified using a state machine 
Eu.ModSt.2089	The declaration of this state machine is identical to the original textual requirements (see ID 93) **FSR1** (Transition from state "**OFF**" to state "**ON**") and **FSR2** (Transition from state "**ON**" to state "**OFF**"): **FSR1:** The system shall switch on the light ("**Switch_Light_On := TRUE**") if the light is switched off (state "**OFF**") and the button is pressed ("**when(Button_Pressed)**"). The **Transition from state "OFF" to state "ON"** represents a **functional system requirement** and may be textually formulated in the requirements specification document as shown below: **Info | OFF** **Req | when(Button_Pressed)/Switch_Light_On := TRUE {OFF - ON}** **Info | ON** **FSR2:** The system shall switch off the light ("**Switch_Light_OFF := TRUE**") automatically after the time t_Light_On has expired ("**after(t_Light_On)**"). The **Transition from state "ON" to state "OFF"** represents a **functional system requirement** and may be textually formulated in the requirements specification document as shown below: **Info | ON** **Req | after(t_Light_On)/Switch_Light_Off := TRUE {ON - OFF}** **Info | OFF**
Eu.ModSt.7013	**8.1.3 Overview introduction to the EULYNX MBSE Process**
Eu.ModSt.1659	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 [31]. 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 and Phase 11: Operation and maintenance,
Eu.ModSt.1662	The CENELEC system life cycle follows the V-model, which highlights verification and validation, especially regarding the fulfilment of safety requirements, as important tasks.
Eu.ModSt.7101	Already during the specification phases of the V-model, verification and validation are important activities, applied to assure the quality of the specification itself.
Eu.ModSt.7102	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.
Eu.ModSt.7103	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.

ID	Requirement
Eu.ModSt.7104	In *Figure 1658*, the "small V" is highlighted in the "big V" and pictures the relationships of verification and validation as part of the virtual prototype development.
Eu.ModSt.1658	Figure 1658 EULYNX "smal V" model 
Eu.ModSt.1539	The AM MBSE essentially covers the "Formalised Requirements" and "State Machine Implementation" phases of the "small V" process. It defines the model views at abstraction levels AL1 and AL2 for the creation of: • **specification models of subsystems (SUS) and** • **specification models of interfaces (SIUS).**
Eu.ModSt.7469	The requirements at abstraction level AL3 of the AM MBSE are currently not defined in EULYNX in a model-based manner.
Eu.ModSt.1555	The behaviour of EULYNX SUS/SIUS is specified from the black box perspective. In a black box specification only the black box behaviour of the SUS/SIUS is considered, i.e. only the external properties of the SUS/SIUS are defined (externally visible input/output behaviour).
Eu.ModSt.7105	User Requirements derived from infrastructure manager (IM) expert knowledge are represented in both cases in the requirements management tool in the form of a "Function List". It lists the required functions used as input information for the creation of the model views at abstraction level "AL 1 Subsystem Definition" "or "Interface Definition" of the AM MBSE using the modelling tool.
Eu.ModSt.7931	If an architectural description of the overall system is available in the form of an analysis model, the model artefacts of the analysis model required for the creation of the respective EULYNX specification model are transferred to the EULYNX specification model by model-to-model transformation.
Eu.ModSt.7470	At this point, the SUS use cases (services) are defined with their stimulus-response behaviour selectively specified by means of use case scenarios using SysML sequence diagrams (Formalised Requirements).
Eu.ModSt.7471	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 modelling tool by a modeler in the role of a model verifier.
Eu.ModSt.7472	Additionally, the use case scenarios are validated by means of inspection by the corresponding IMs in the roles of model validators.
Eu.ModSt.7473	In the next step, the system views created at abstraction level "AL 1 Subsystem Definition/Interface Definition" are refined at abstraction level "AL 2 Subsystem Requirements/Interface Requirements" by means of executable SysML state machines (State Machine Implementation).
Eu.ModSt.7474	The conformity of the model to the SysML specification and the EULYNX Modelling Standard is verified tool-based and by means of inspection by the model verifier.
Eu.ModSt.7475	To implement the state machines as a virtual prototype, simulation code is generated. Subsequently, the GUI of the virtual prototype is designed, and an executable is created.
Eu.ModSt.7476	The executable representing the virtual prototype enables both the tool-independent standalone simulation of the specified behaviour and when connected to the simulation tool the simulation together with the animation of the corresponding state machines.

ID	Requirement
Eu.ModSt.7477	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 7481.
Eu.ModSt.7478	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 Subsystem Definition/Interface Definition" is dynamically verified by simulation-based testing of the virtual prototype carried out interactively by the model verifier.
Eu.ModSt.7479	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.
Eu.ModSt.7480	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 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.
Eu.ModSt.7481	Figure 7481 Principle of a virtual Prototype 
Eu.ModSt.7094	*Figure 7116* shows the commonly used engineering paths for generating the model views of the SUS or SIUS specification models in conformity with the "small V" shown in *Figure 1658*. Depending on the project-specific input conditions, the engineering paths can also be applied in a modified form.
Eu.ModSt.7118	In general, the engineering path for creating the SUS model views (black dashed arrows) includes the engineering path for creating the SIUS model views (red dashed arrows).
Eu.ModSt.7117	The model views used reflect the current state of the EULYNX MBSE methodology and may be complemented by further model views in the future (e.g. model views of the Technical Viewpoint or model views on AL3).

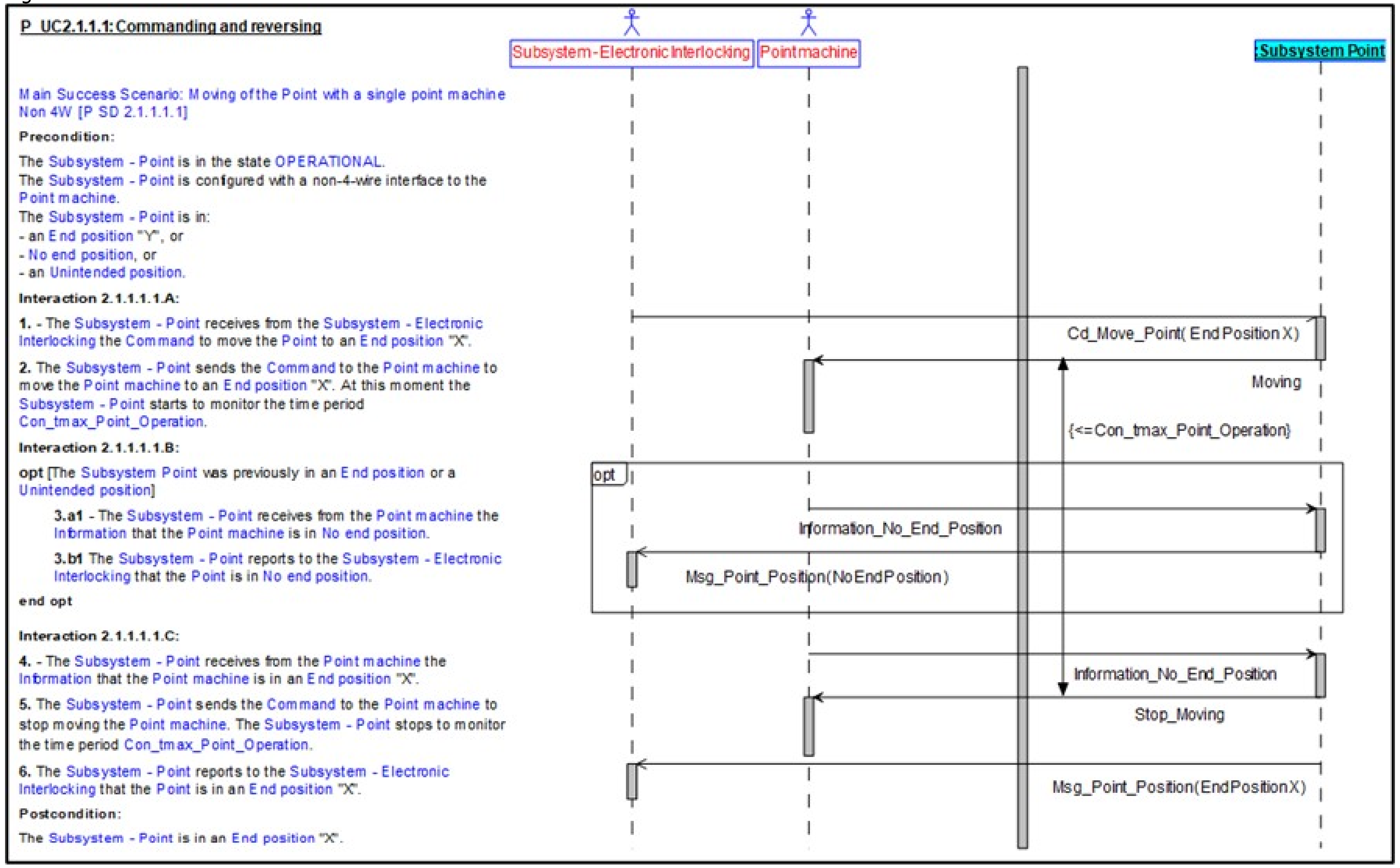
ID	Requirement
Eu.ModSt.7116	Figure 7116 Engineering paths of the EULYNX "smal V" model 
Eu.ModSt.1549	The engineering path for creating the SUS model views starts at the Functional Viewpoint on abstraction level AL1.
Eu.ModSt.1241	**Task (1): creation of model view "Functional Context"** Based on stakeholder requirements (for example IM requirements) which are defined in the area User Requirements of the MBSE SF, for example in the form of a function list, the model view "Functional Context" is created **(1)**.
Eu.ModSt.1630	As shown in *Figure 1633*, the model view Functional Context summarises the use case structure graphically and names all use cases the SUS is expected to perform. Furthermore, it allocates the use cases to the SUS and defines their interrelations as well as their relations to the actors in the SUS environment.
Eu.ModSt.1557	Use cases describe the functionality of a SUS such as "Subsystem Point" in terms of how it is used to achieve the goals of its various users. In model view "Functional Context" they are denoted by ellipses, and the actors participating in the use cases are connected to the ellipses by solid lines.
Eu.ModSt.1623	The users of a system are described by actors, which can represent external systems such as "Point machine" or people who interact with the system.
Eu.ModSt.1628	Consequently, a use case does not contain any information how it is implemented in the SUS.

<table>
<tr><th>ID</th><th>Requirement</th></tr>
<tr><td>Eu.ModSt.1633</td><td>
Figure 1633 Model view "Functional Context" of a SUS

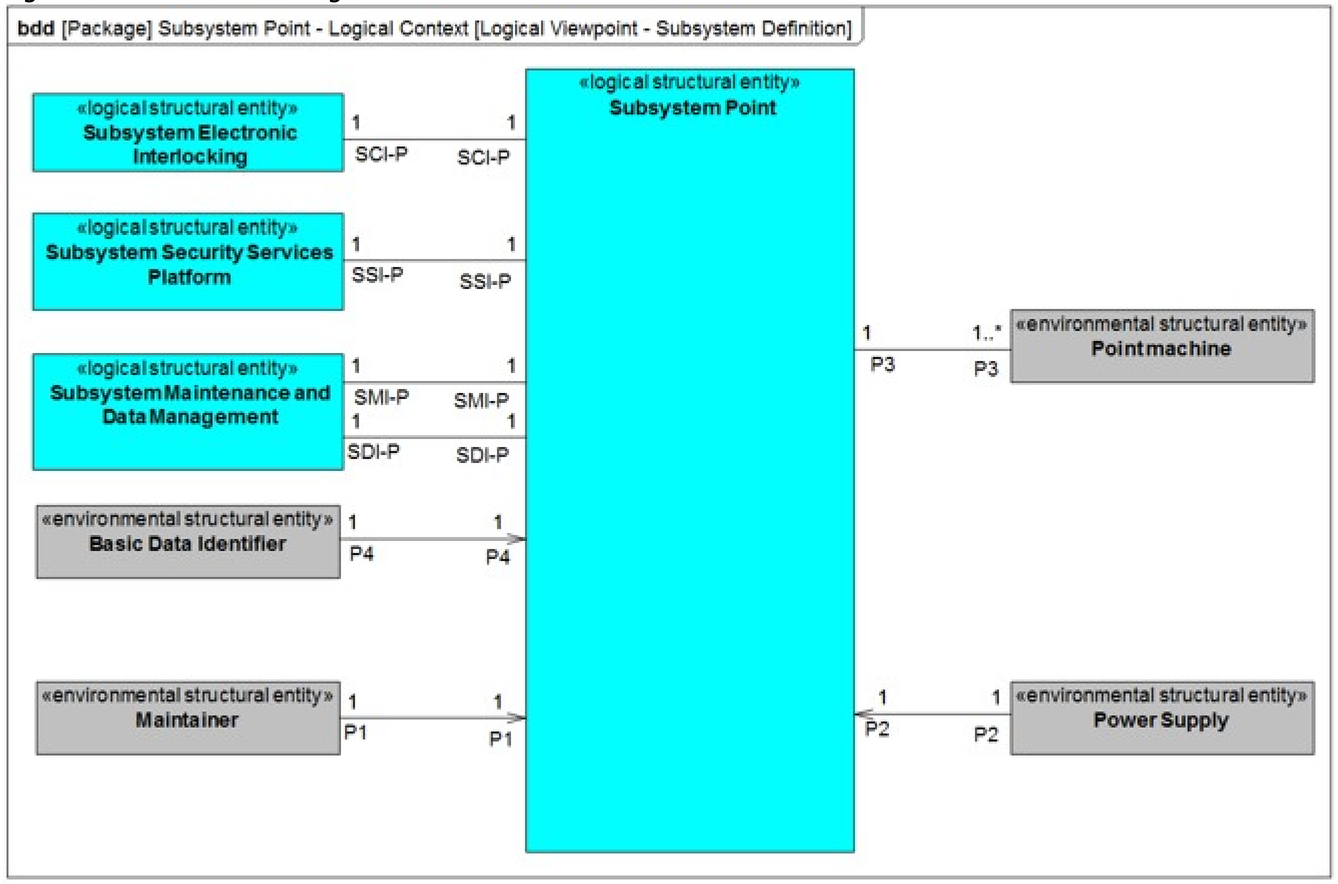


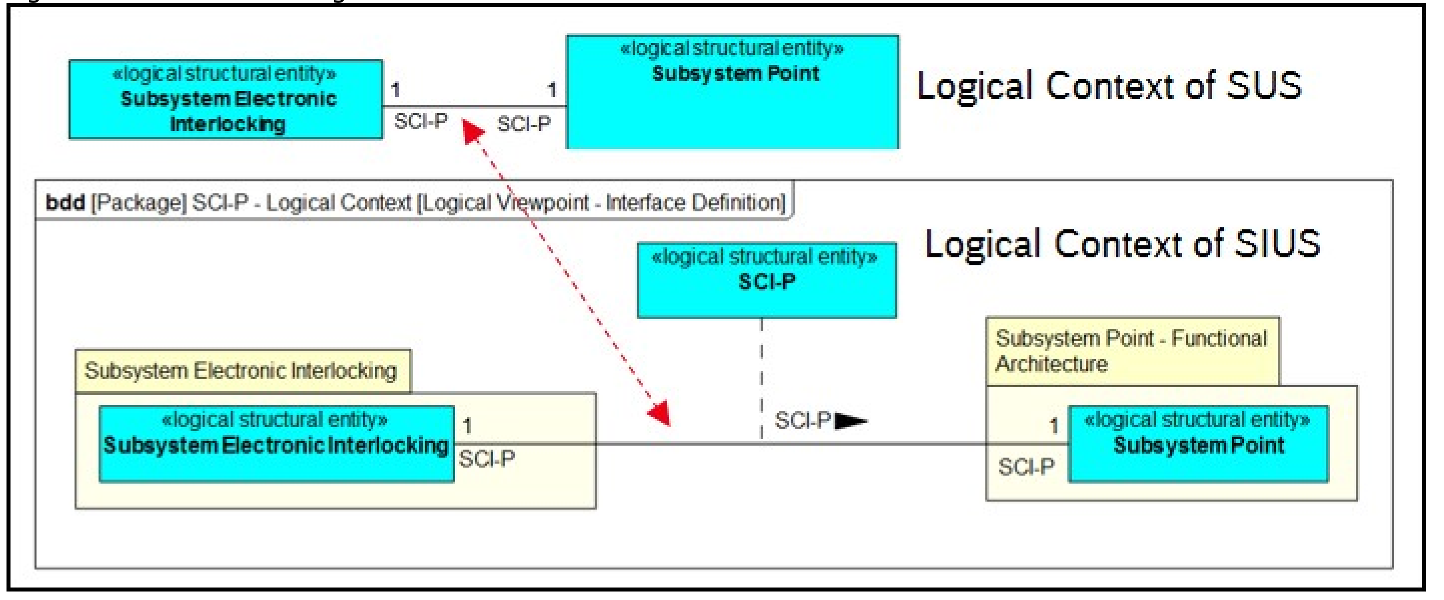
</td></tr>
<tr><td>Eu.ModSt.1622</td><td>Task (2): creation of model view "Use case scenarios"
Based on the definitions in the model view "Functional Context", the model view "Use case scenarios" is subsequently created.</td></tr>
<tr><td>Eu.ModSt.1653</td><td>A use case may be defined by one or more use case scenarios (SysML sequence diagrams) in order to describe the exchange of messages between the SUS and its environment. It is the central construct to define parts of behaviour of the SUS that can be observed at the system boundary.</td></tr>
<tr><td>Eu.ModSt.1634</td><td>An example use case scenario of the use case "P_UC2.1.1.1: Commanding and reversing" is depicted in Figure 1635.</td></tr>
</table>

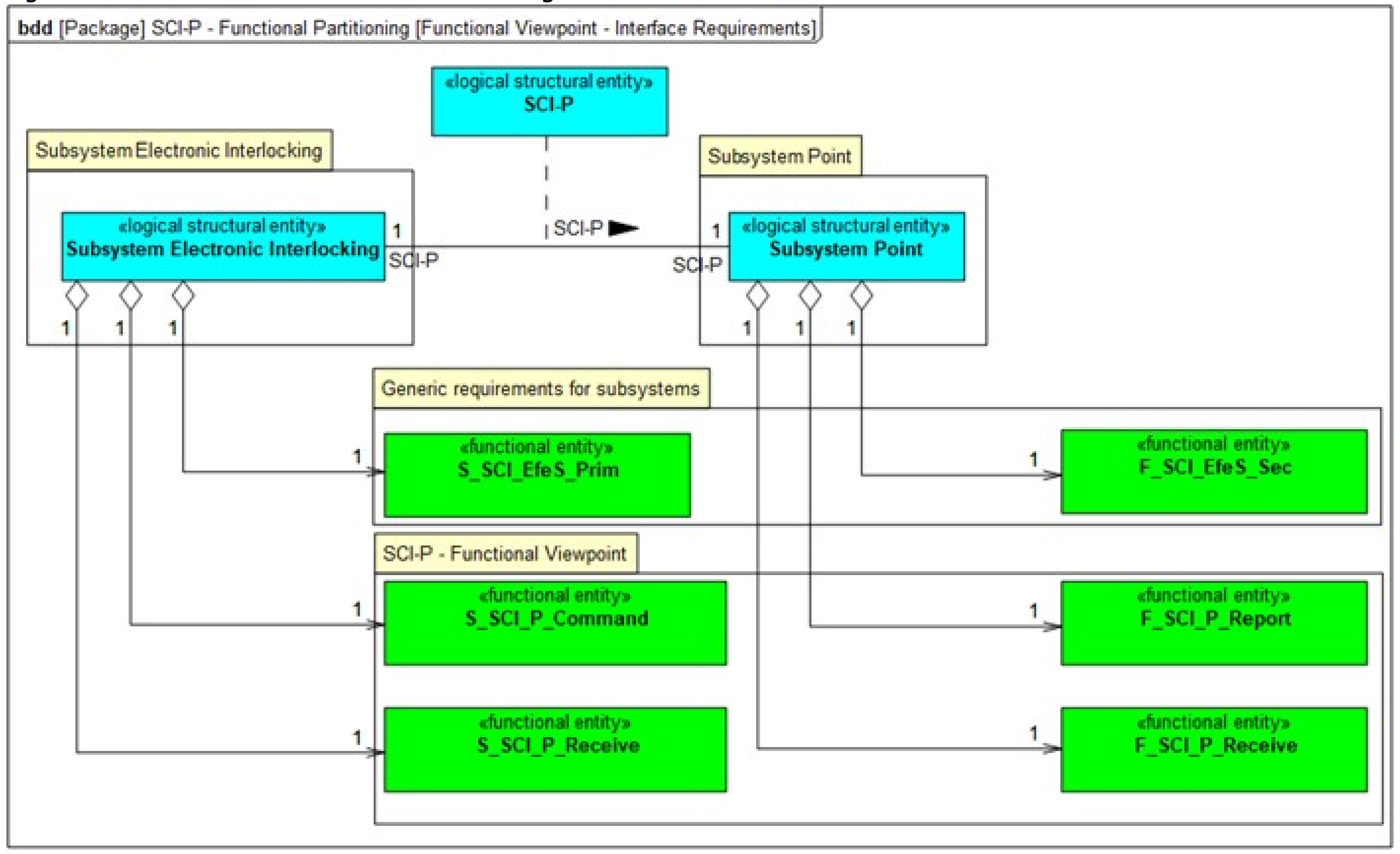
<table>
<tr><th>ID</th><th>Requirement</th></tr>
<tr><td>Eu.ModSt.1635</td><td>
Figure 1635 Model view "Use case scenario" of a SUS

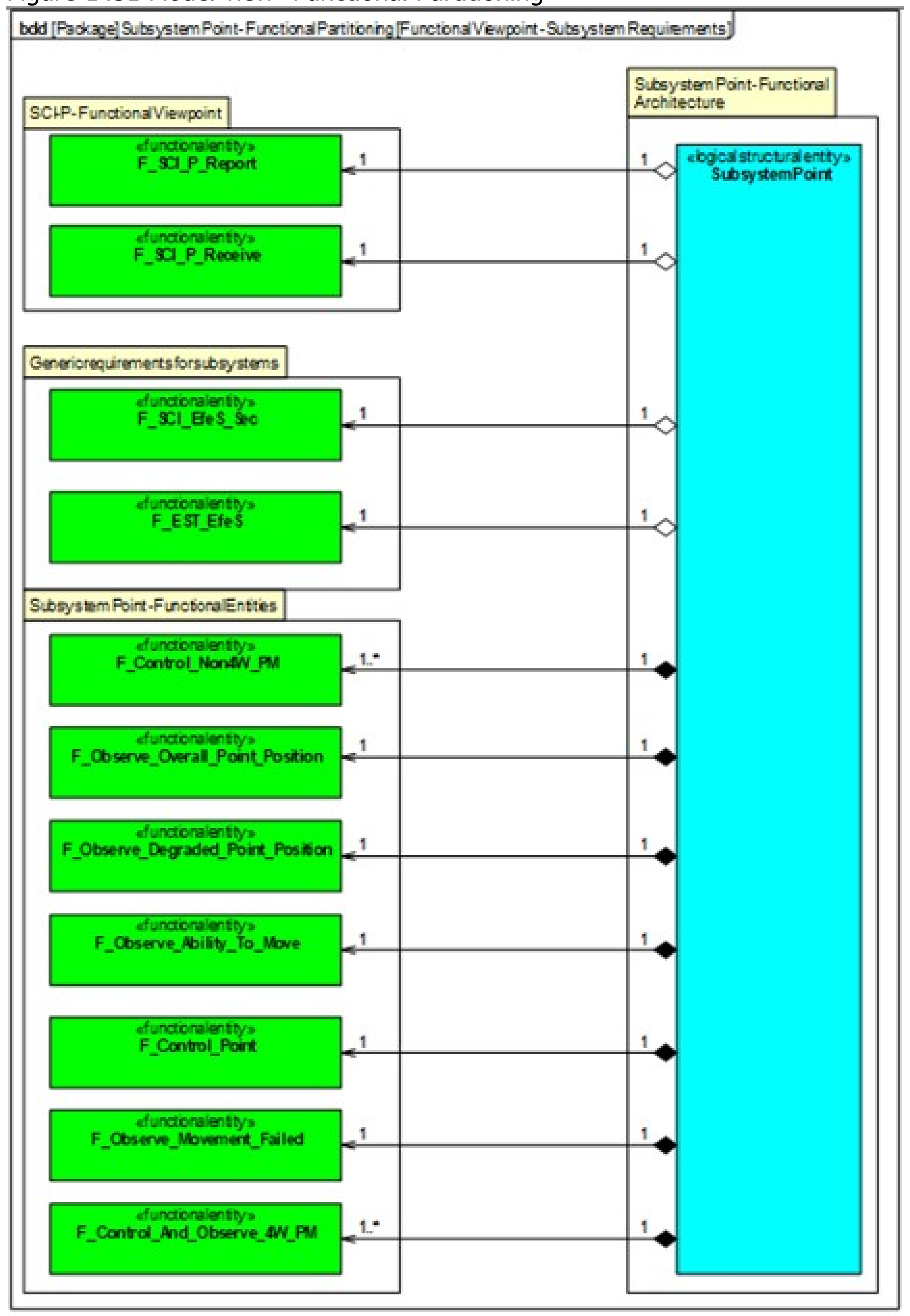


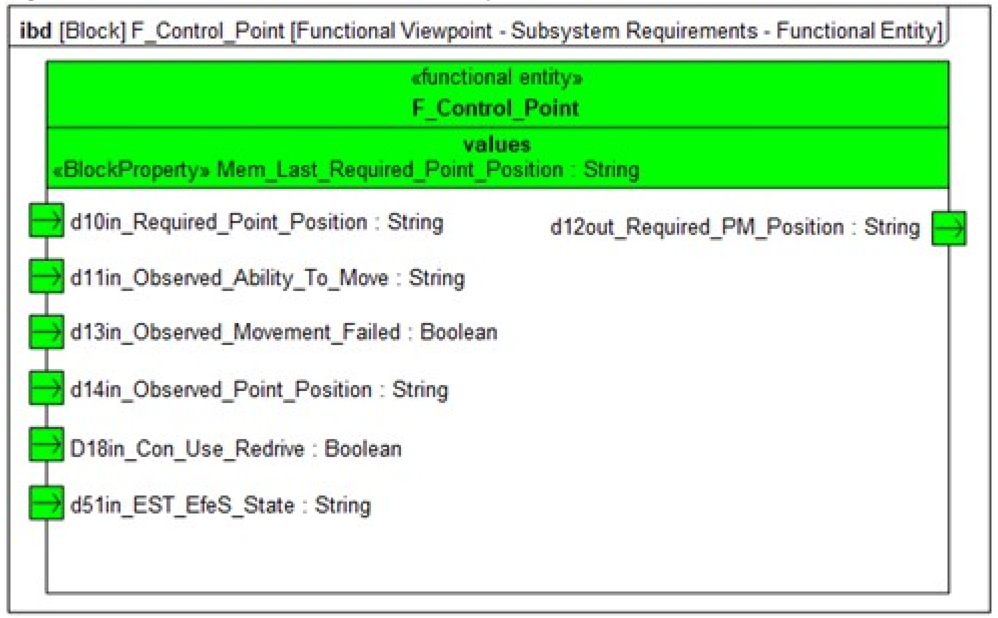
</td></tr>
<tr><td>Eu.ModSt.1629</td><td>Task (3): creation of model view "Logical Context" of a SUS
Based on the definitions in the model views "Functional Context" and "Use case scenarios" the model view "Logical Context" is subsequently created at the Logical Viewpoint on abstraction level AL1.</td></tr>
<tr><td>Eu.ModSt.1535</td><td>In the example shown in Figure 1540 the model view "<u>Logical Context</u>" is depicted. It describes the structure of the SUS at the top level and the actors in the environment interacting with it and their quantity structure (multiplicities). Furthermore, the logical interfaces such as SCI-P, SSI-P, P3 and so on between the SUS and the actors are defined.</td></tr>
</table>

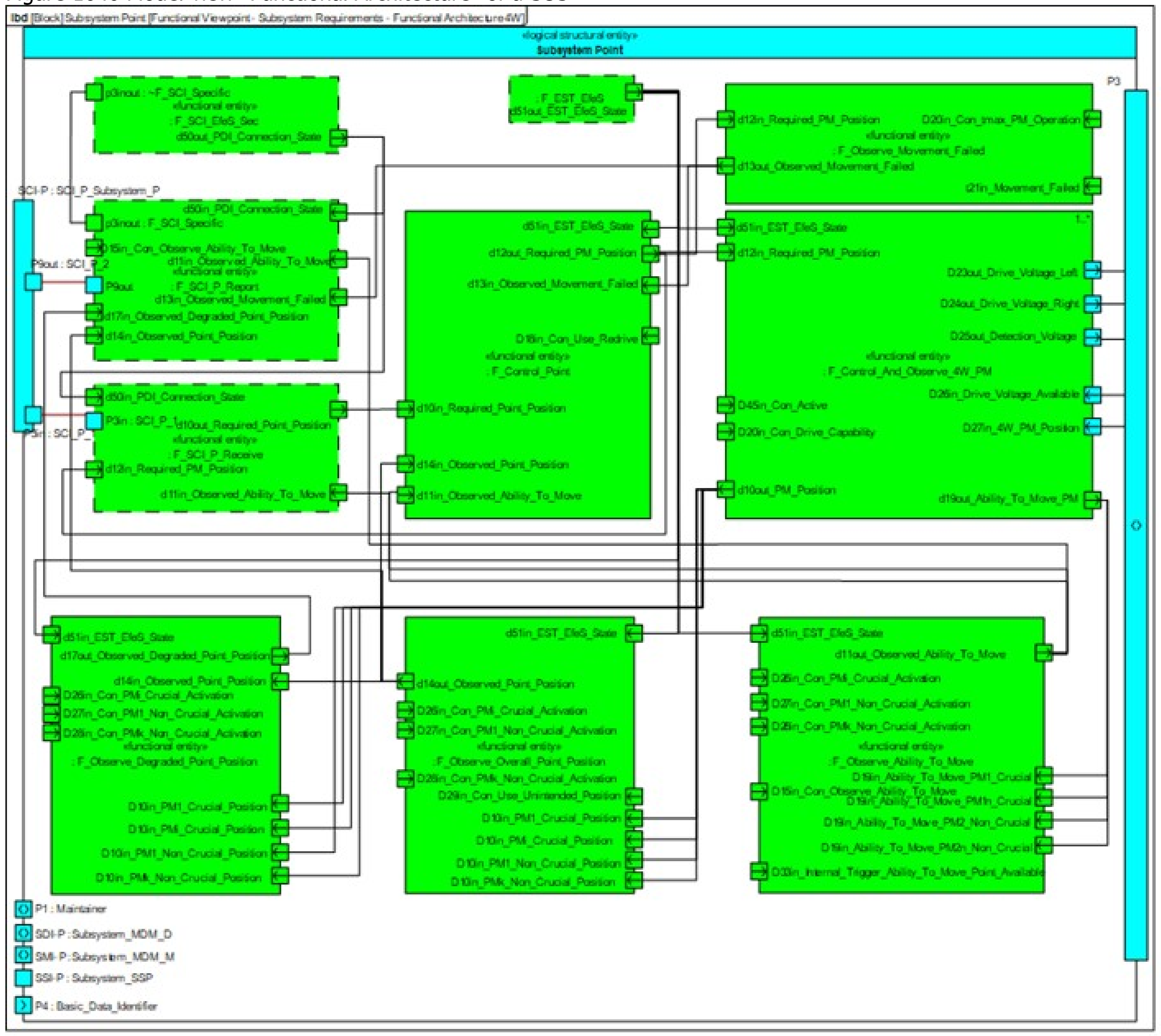
ID	Requirement
Eu.ModSt.1540	Figure 1540 Model view "Logical Context" of a SUS 
Eu.ModSt.1562	**Task (4): creation of model view "Logical Context" of the interfaces to be standardised** Based on the definitions of the logical interfaces defined in model view "Logical Context" of a SUS, the model view "Logical Context" of its standardised interfaces (SIUS) is subsequently created at the Logical Viewpoint on abstraction level AL1.
Eu.ModSt.7122	At the upper level of abstraction an interface is represented by a SysML association. An association is depicted as a continuous line between the communication participants. The association that represents a logical interface in the model view "Logical Context" of the SIUS corresponds to the respective association in the model view "Logical Context" of the SUS.
Eu.ModSt.1626	The model view "Logical Context" of a SIUS as shown in *Figure 1637* describes the logical view of an interface at the upper level of abstraction.
Eu.ModSt.7123	The SysML association is linked to a SysML association block, which serves to refine the relationship. The global behaviour of the application protocol (Railway Control Protocol: RCP) is then specified in this later.

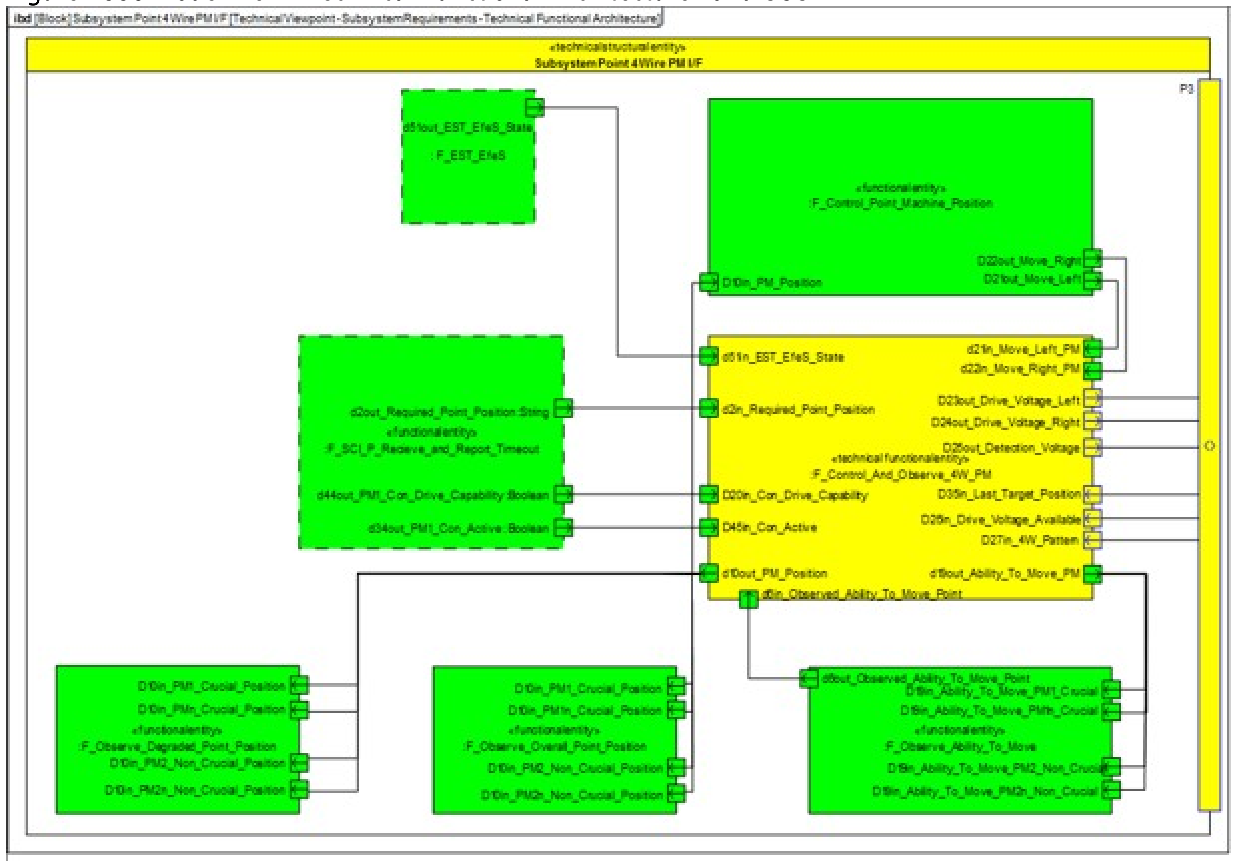
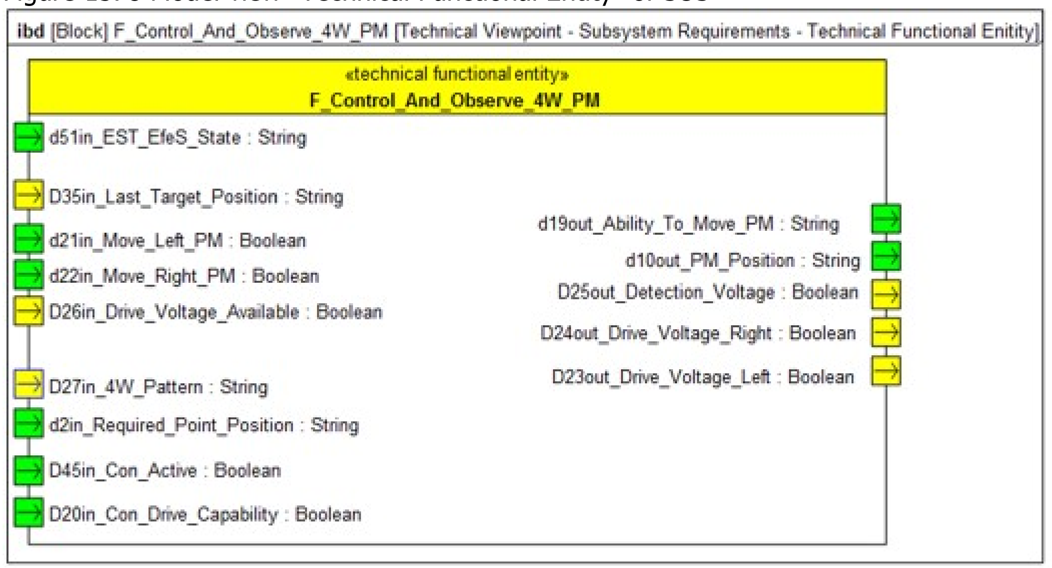
ID	Requirement
Eu.ModSt.1637	Figure 1637 Model view "Logical Context" of a SIUS 
Eu.ModSt.1627	**Task (5): creation of model view "Functional Partitioning" of the interfaces to be standardised** Based on the definition of the model view "Logical Context" of the relevant interfaces, the model view "Functional Partitioning" is subsequently created at the Functional Viewpoint on abstraction level AL1.
Eu.ModSt.1636	The model view "Functional Partitioning" as shown in *Figure 1643* describes the refinement of the interface defined in model view "Logical Context" using FEs. These FEs specify the local behaviours (see *chapter 8.2.4*) of the application layer (PDI: Process Data Interface Protocol) of the communication protocol stack on each side of the communication link.
Eu.ModSt.7901	The FEs are assigned to the involved subsystems via reference associations (marked with a white diamond). The reference associations express that the FEs are not part of the subsystems, but are only used there. They are part of the PDI.
Eu.ModSt.7319	In addition, the respective possible number of FEs is determined by multiplicities.
Eu.ModSt.7320	The model view "Functional Partitioning" of a SIUS is the basis for the model view "Functional Architecture" of a SIUS. While the former, however, defines the absolute behaviour (the maximum possible number of FEs is defined), the model view "Functional Architecture" also allows an excerpted description (Description of different configurations).

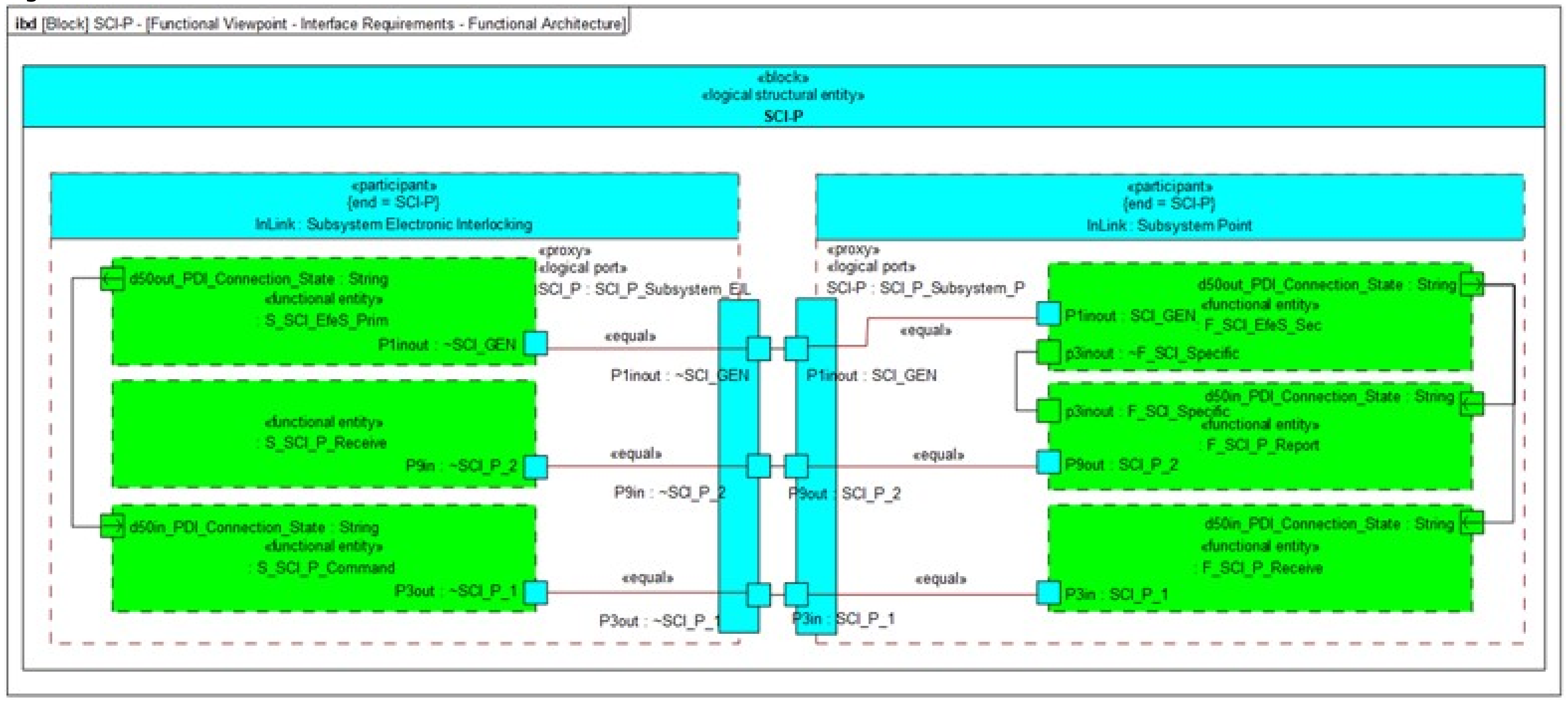
ID	Requirement
Eu.ModSt.1643	Figure 1643 Model view "Functional Partitioning" of a SIUS 
Eu.ModSt.1640	Since the FEs defined in the model view "Functional Partitioning" are used for the further specification of both the SUS and the SIUS, the engineering path splits at this point. The further creation of the model views takes place along two different engineering paths, which are described in the following two *subchapters 8.1.3.1 Engineering path SUS and 8.1.3.2 Engineering path SIUS*.
Eu.ModSt.1927	**8.1.3.1 Engineering path SUS**
Eu.ModSt.1537	**Task (6a): creation of model view "Functional Partitioning" of a SUS** Starting from the model view "Functional Partitioning" of the involved SIUS, the engineering path continues with the generation of the further model views of the SUS at the Functional Viewpoint at abstraction level AL2.
Eu.ModSt.208	First, the model view "Functional Partitioning" of the SUS as depicted in *Figure 1451* is created. It describes the refinement of the SUS by means of the FEs defined in the SIUS model view "Functional Partitioning", which represent the local behaviours of the PDI, as well as the FEs specific to the SUS (linking behaviour according to *chapter 8.2.4*).
Eu.ModSt.7902	FEs which are assigned to the subsystem via reference associations (marked with a white diamond) are not part of the subsystem, but are only used there. They represent the local behaviour of the PDI and are part of it.
Eu.ModSt.7903	FEs which are assigned to the subsystem via composite associations, i.e. so-called whole-part relationships (marked with a black diamond) are part of the subsystem. They represent the specific behaviour of the subsystem that influences more than one interface. This so-called "linking behaviour" is also used to link the behaviour assigned to the interfaces.
Eu.ModSt.7318	In addition, the respective possible number of FEs is determined by multiplicities.
Eu.ModSt.1930	The model view "Functional Partitioning" of a SUS is the basis for the model view "Functional Architecture" of a SUS. While the former, however, defines the absolute behaviour (the maximum possible number of FEs is defined), the model view "Functional Architecture" also allows excerpted descriptions (Description of different configurations).

ID	Requirement
Eu.ModSt.1451	Figure 1451 Model view "Functional Partitioning" 
Eu.ModSt.1647	**Task (7a): creation of model view "Functional Entity" of a SUS** Based on the SUS-specific FEs defined in the model view "Functional Partitioning" of a SUS, the model view "Functional Entity" as shown in *Figure 1644* is created for these FEs.
Eu.ModSt.7322	SUS-specific FEs represent control system functions such as "F_Control_Point" (see *Figure 1644*). They have executable SysML state machines and SysML block operations to describe behaviour. SysML state machines enable the specification of finite discrete event dynamic behaviour. SysML block operations are used to perform logical or algebraic transformations.
Eu.ModSt.7463	The model view "Functional Entity" describes the behaviour or part of the behaviour of a SUS independent of technology.

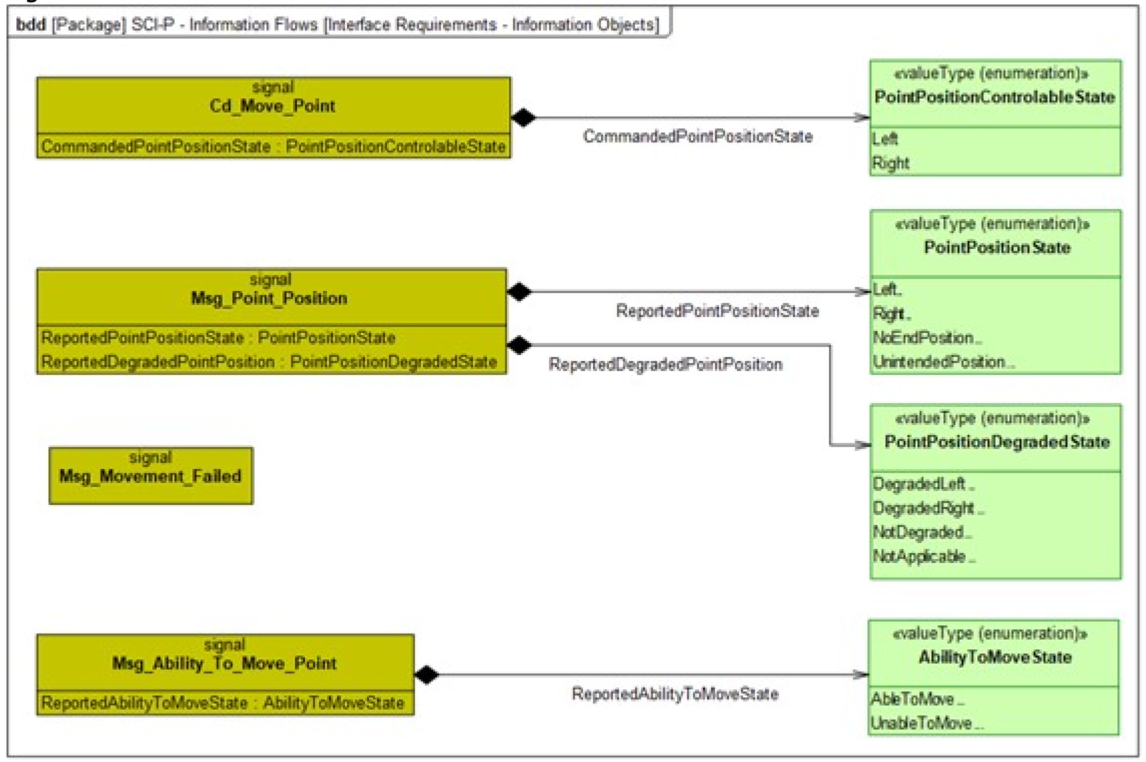
ID	Requirement
Eu.ModSt.1644	Figure 1644 Model view "Functional Entity" of a SUS 
Eu.ModSt.1645	**Task (8a): creation of model view "Functional Architecture" of a SUS** Based on the model view "Functional Partitioning" of the SUS, the model view "Functional Architecture" is created.
Eu.ModSt.7459	The model view "Functional Architecture" as shown exemplarily in *Figure 1646* describes the external visible stimulus-response behaviour of a SUS represented by a Logical Structural Entity (LSE) that is structured in a way that enables an interface centric specification approach as described in *chapter 8.2.4*. The behaviour of the SUS is divided into FEs, which communicate with each other via internal interfaces and with the environment via external interfaces.
Eu.ModSt.7460	The model view "Functional Architecture" describes the behaviour of a SUS independent of technology.

ID	Requirement
Eu.ModSt.1646	Figure 1646 Model view "Functional Architecture" of a SUS 
Eu.ModSt.1654	**Task (9a): creation of model view "Technical Functional Architecture**" **of a SUS** Based on the model view "Functional Architecture" of the SUS, the model view "Technical Functional Architecture" is created at the Technical Viewpoint on abstraction level AL2. This model view is only created if technical functional requirements are to be described in a model-based manner.
Eu.ModSt.7461	The model view "Technical-Functional Architecture" of the SUS, as exemplified in *Figure 1558*, describes the externally visible stimulus-response behaviour of a SUS represented by one or more TSEs based on technical requirements. The SUS is represented by a technical structural entity (TSE).
Eu.ModSt.7462	The technology-independent behaviour described in the Functional Viewpoint in the form of a Functional Architecture through FEs is complemented or substituted by technology-dependent behaviour in the form of TFEs. TFEs are coupled with each other, with already defined FEs or with the environment via external technical interfaces.

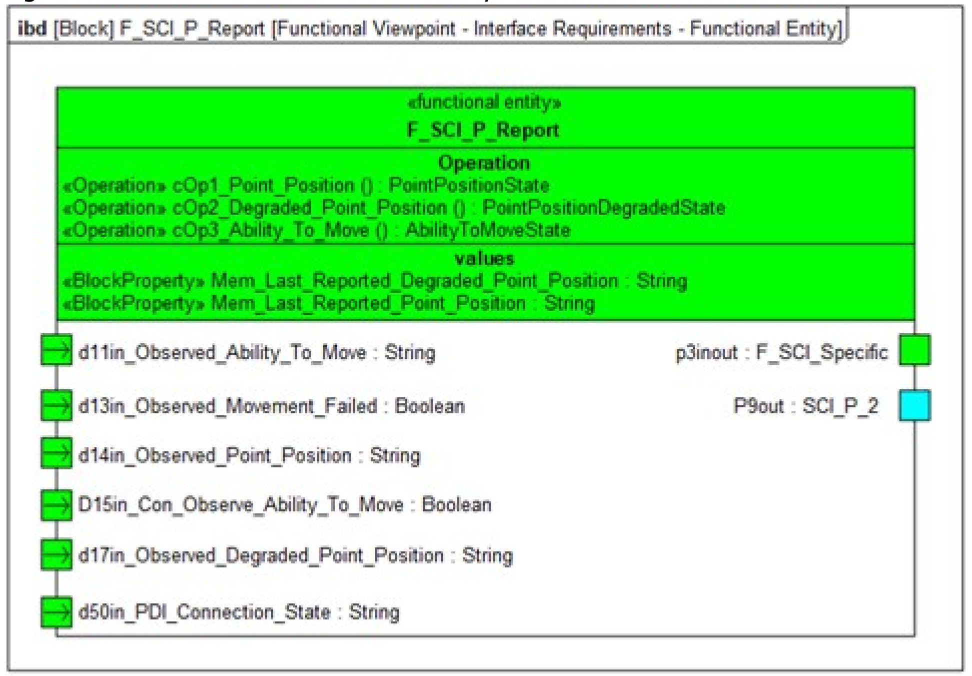
ID	Requirement
Eu.ModSt.1558	Figure 1558 Model view "Technical Functional Architecture" of a SUS 
Eu.ModSt.1652	**Task (10a): creation of model view "Technical Functional Entity" of a SUS** Based on technical requirements, the model view "Technical Functional Entity" as shown in *Figure 1578* is created at the Technical Viewpoint on abstraction level AL2.
Eu.ModSt.7464	TFEs represent technology-dependent control system functions such as "F_Control_And_Observe_4W_PM" (see *Figure 1578*). As well as FEs, TFEs also have executable SysML state machines and SysML block operations to describe behaviour. SysML state machines enable the specification of finite discrete event dynamic behaviour. SysML block operations are used to perform logical or algebraic transformations.
Eu.ModSt.1578	Figure 1578 Model view "Technical Functional Entity" of SUS 
Eu.ModSt.341	**8.1.3.2 Engineering path SIUS**
Eu.ModSt.1649	**Task (6b): creation of model view "Functional Architecture" of a SIUS** Starting from the model view "Functional Partitioning" of the involved SIUS, the engineering path continues with the generation of the further model views of the SIUS at the Functional Viewpoint at abstraction level AL2.

ID	Requirement
Eu.ModSt.7465	First, the model view "Functional Architecture" of the SIUS as depicted in *Figure 1648* is created. It defines the global behaviour of the application protocol. As described in *chapter 8.2.4* the global behaviour is described by connecting the local behavioural components referenced by a communication partner with the corresponding ones of the neighbour via communication channels.
Eu.ModSt.7466	The description of the global behaviour of the application protocol is done by the internal structuring of the association block defined in model view "Functional Partitioning" of the involved SIUS. In this process, the communication partners, which in turn reference the local behavioural parts of the protocol represented by FEs, are referenced in the form of SysML participant properties and connected via their interfaces with connectors.
Eu.ModSt.1648	Figure 1648 Model view "Functional Architecture" of a SIUS 
Eu.ModSt.1641	**Task (7b): creation of model view "Information Flow" of a SIUS** Based on the defined interfaces in model view "Functional Architecture" of a SIUS the model view "Information Flow" is created. The model view "Information Flow" as shown in *Figure 1567* describes the information objects to be exchanged via an interface.
Eu.ModSt.7467	The information objects are represented by SysML signals such as "Cd_Move_Point". These signals can in turn have typed attributes such as "CommandedPointPositionState" that represent parameters of the information objects. For example, the attribute "CommandedPointPositionState" is typed with the enumeration "PointPositionControlableState" with the available values "Left" and "Right".
Eu.ModSt.7468	The information objects are further refined into telegrams on AL3 of the AM MBSE. However, the telegrams are currently not yet implemented in a model-based way.

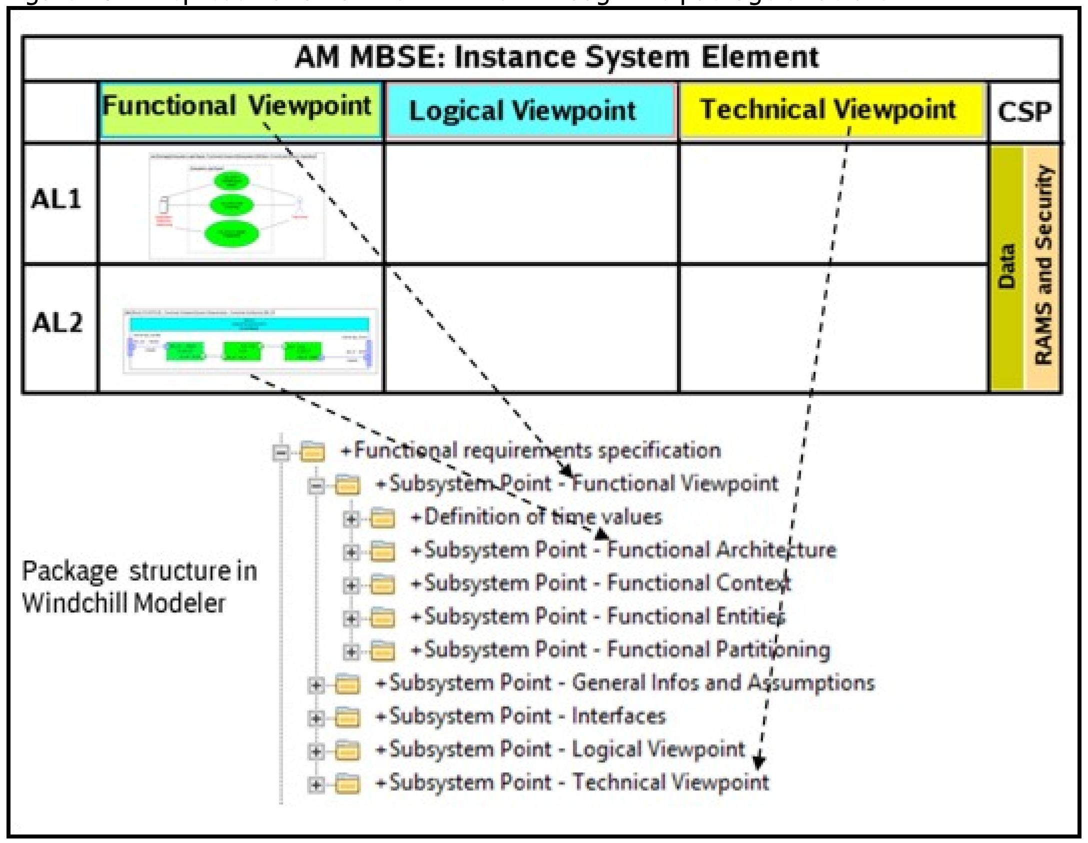
<table>
<tr><th>ID</th><th>Requirement</th></tr>
<tr><td>Eu.ModSt.1567</td><td>
Figure 1567 Model view "Information Flow" of a SIUS

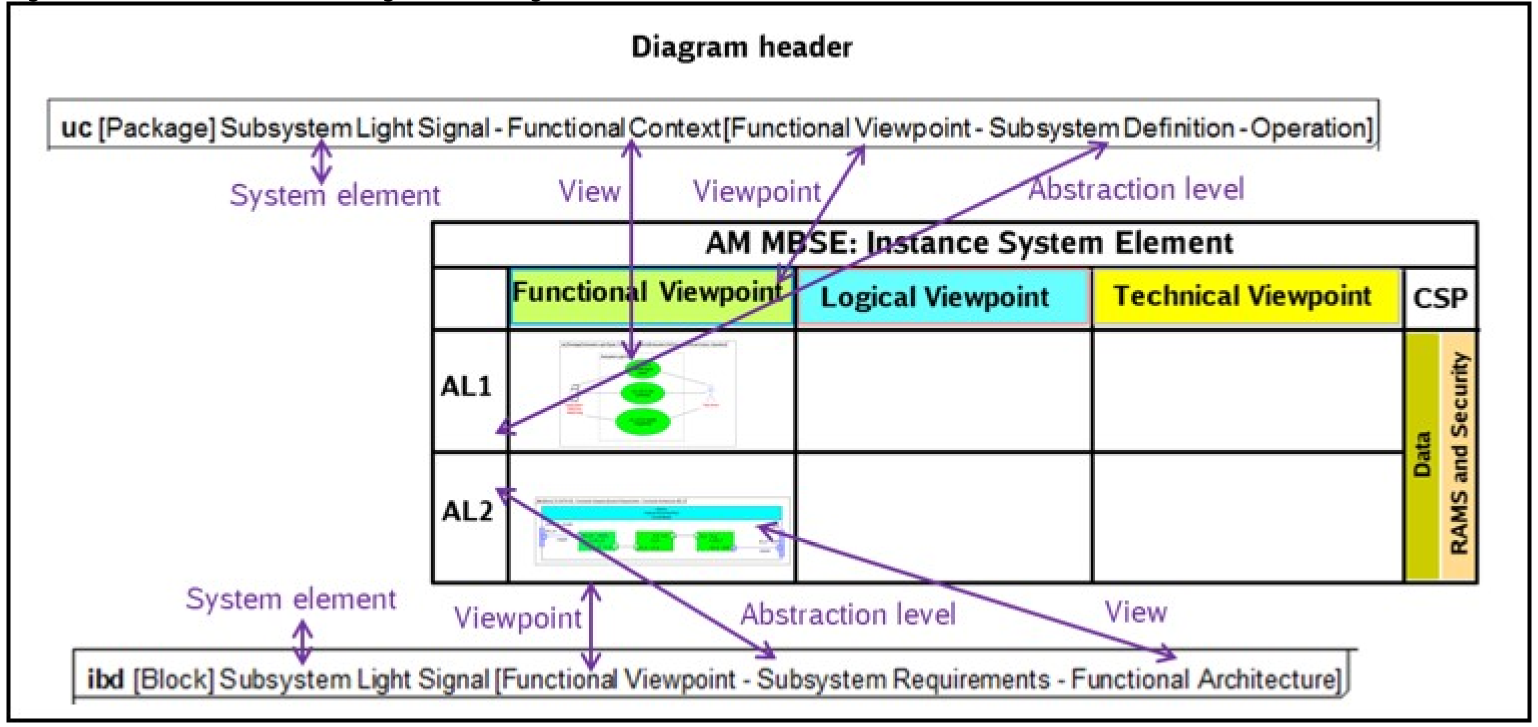


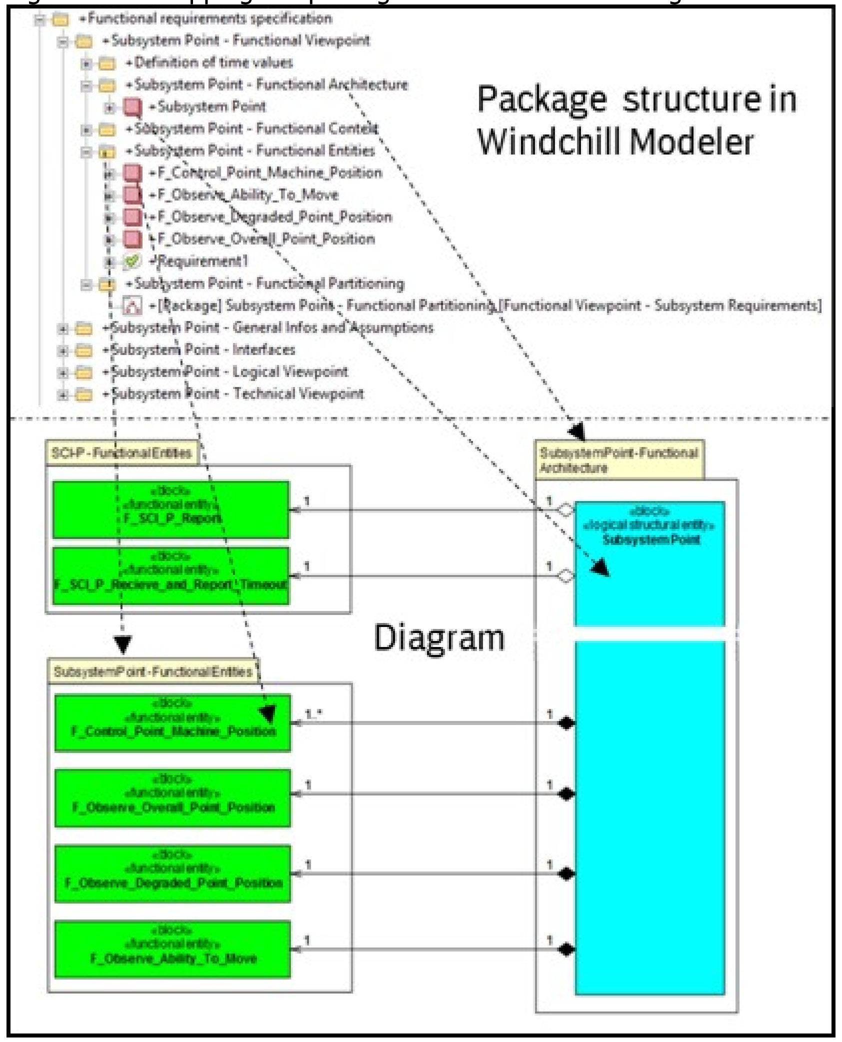
</td></tr>
<tr><td>Eu.ModSt.1642</td><td>
Task (8b): creation of model view "Functional Entity" of a SIUS
After the information objects are defined, the model views "Functional Entity" are created for the FEs defined in the model view "Functional Partitioning" of a SIUS. These FEs such as "F_SCI_P_Report" (see Figure 1579) represent the local behaviours of the RCP of the respective interface. They have executable SysML state machines and SysML block operations to describe behaviour. SysML state machines enable the specification of finite discrete event dynamic behaviour. SysML block operations are used to perform logical or algebraic transformations.
</td></tr>
<tr><td>Eu.ModSt.1579</td><td>
Figure 1579 Model view "Functional Entity" of a SIUS



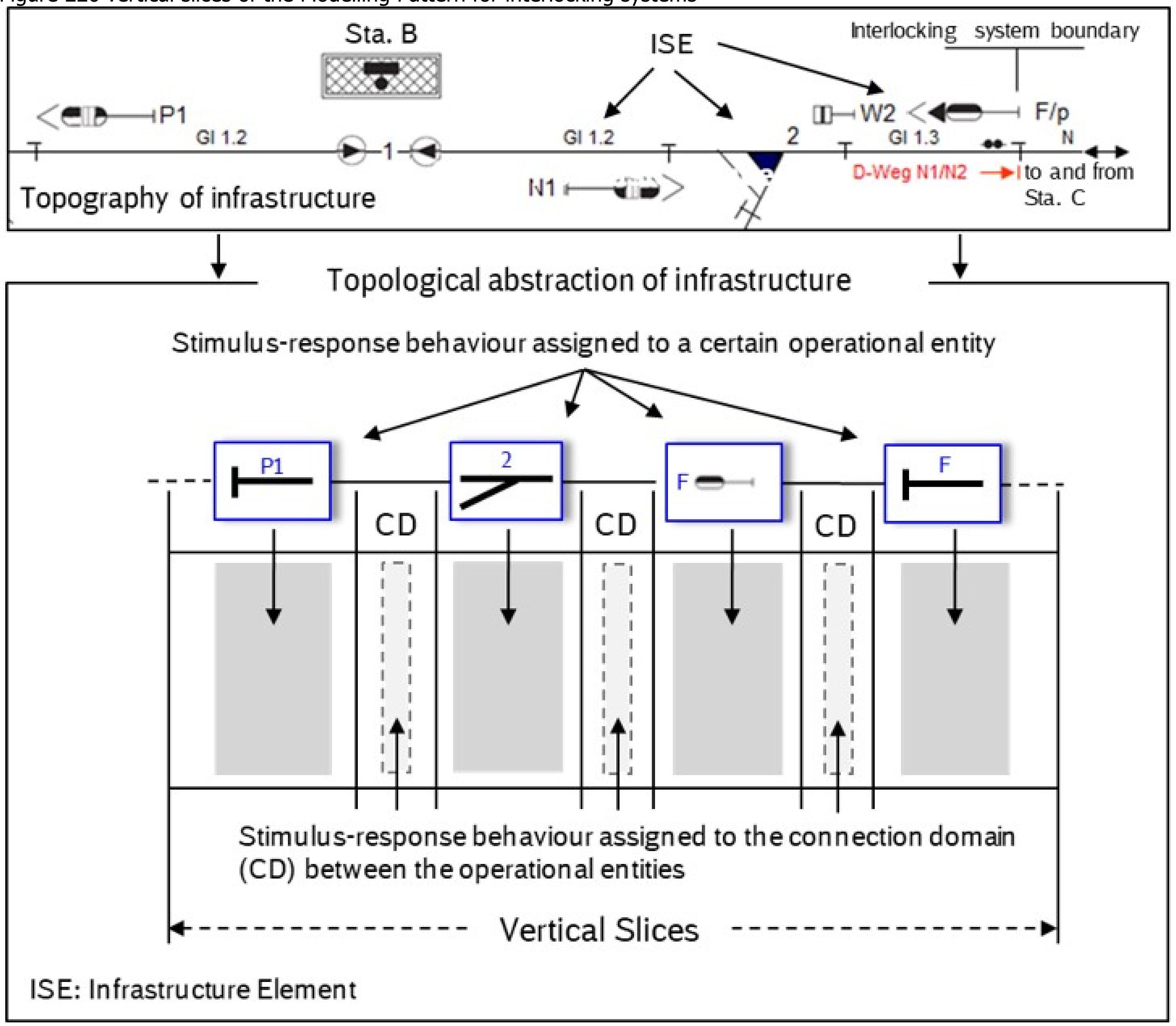
</td></tr>
<tr><td>Eu.ModSt.2121</td><td>
The following chapters describe general modelling rules (chapter 8.2) and the rules for creating the model views used to specify EULYNX SUS (chapter 8.3) and the ones used to define EULYNX SIUS (chapter 8.4). As the model views "Functional Entity" and "Technical Functional Entity" are used for the specification of EULYNX SUS as well as for the specification of EULYNX SIUS they are described in the separate chapters 8.5 and 8.6.
</td></tr>
<tr><td>Eu.ModSt.363</td><td>8.2 Model views - General modelling rules</td></tr>
</table>

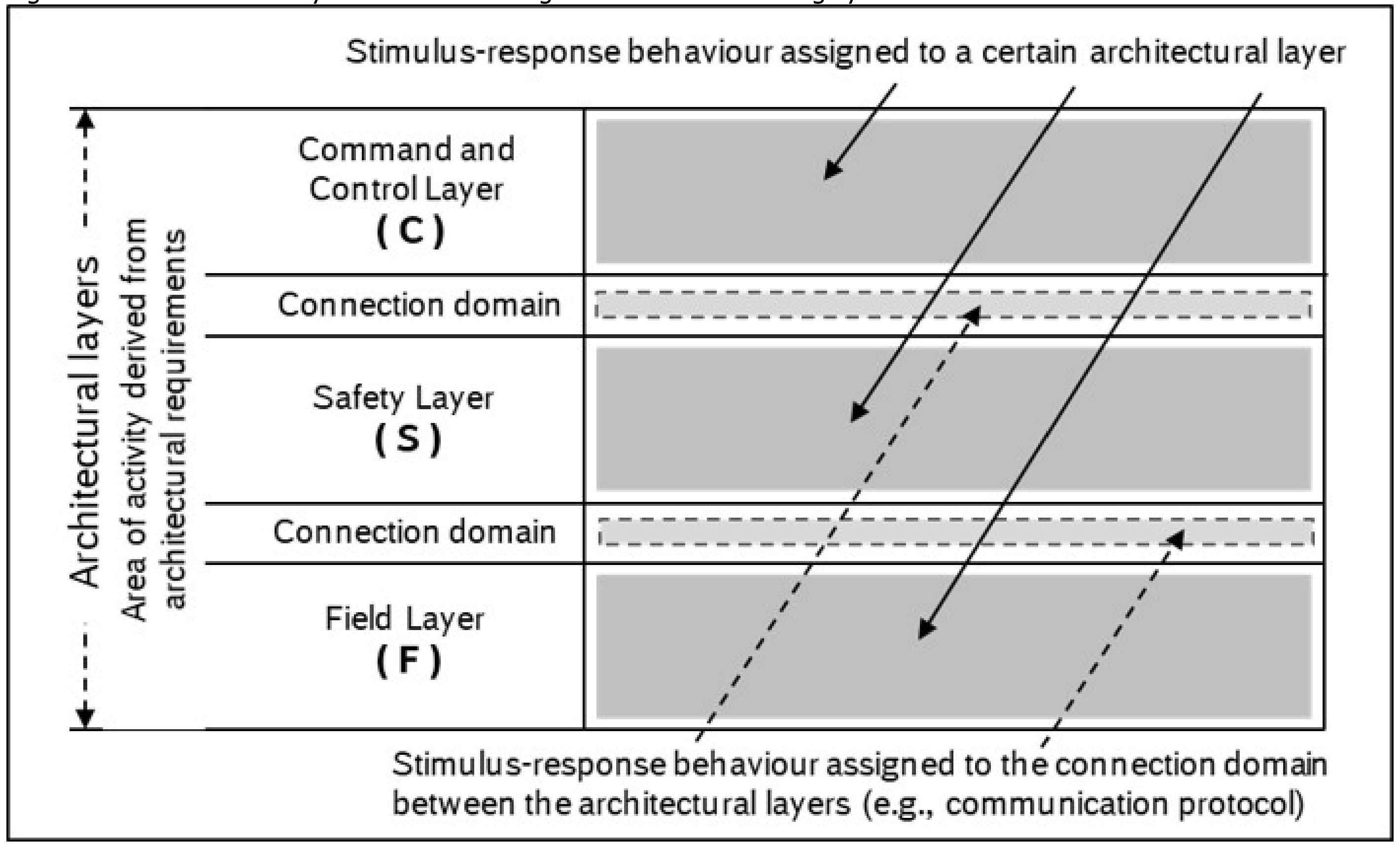
ID	Requirement
Eu.ModSt.58	The system requirements of a specification model (abstraction levels AL2 Subsystem Requirements and AL2 Interface Requirements) of the AM MBSE must be executable and provide a graphical user interface enabling model simulation.
Eu.ModSt.60	Before delivering derived specifications to the signalling system supplier, quality assurance must be completed by carrying out the verification and validation activities defined in the MBSE process.
Eu.ModSt.63	Links to model elements embedded blue-coloured in model descriptions formulated in prose must not be put in quotation marks.
Eu.ModSt.1160	The related information, which is required to convoy the complete meaning of a model element, must be documented for each used model element in the modelling tool (e.g. Properties ->Text->Description).
Eu.ModSt.1161	Unless there are project-specific commitments, stereotypes such as <<block>>, <<ProxiPort>> and so forth may be shown on the diagrams if the modeller regards it as beneficial.
Eu.ModSt.1162	Unless there are project-specific commitments, data types such as Boolean, Integer, PulsedIn, PulsedOut and so forth may be shown on the diagrams if the modeller regards it as beneficial.
Eu.ModSt.1239	Shapes and colours of model elements presented in this modelling standard can be adapted according to project-specific commitments, unless explicitly required. Example: An actor basically is depicted as a stickman. It might be project-specifically determined to use the image of a cube if the actor represents a system and a "stickman" if the actor represents a person.
Eu.ModSt.1456	Project-specific requirements transcending the requirements of Modelling Standard are to be documented separately.
Eu.ModSt.7847	As shown in principle in *Figure 7847*, the AM MBSE is to be represented by the package structure in the modelling tool.
Eu.ModSt.7844	Figure 7844 Representation of the AM MBSE through the package structure 
Eu.ModSt.2027	Viewpoint, abstraction level and model view of the AM MBSE name are made evident in the header of the diagram representing a certain model view.
Eu.ModSt.2028	Examples: • The view "Functional Context" depicted in *Figure 2029* describing a certain aspect of system element Subsystem Light Signal by a SysML use case diagram (uc) belongs to the "Functional Viewpoint" and has the granularity of abstraction level AL1 (Subsystem Definition). • The view "Functional Architecture" depicted in *Figure 2029* describing a certain aspect of system element Subsystem Light Signal by a SysML internal block diagram (ibd) belongs to the "Functional Viewpoint" and has the granularity of abstraction level AL2 (Subsystem Requirements).

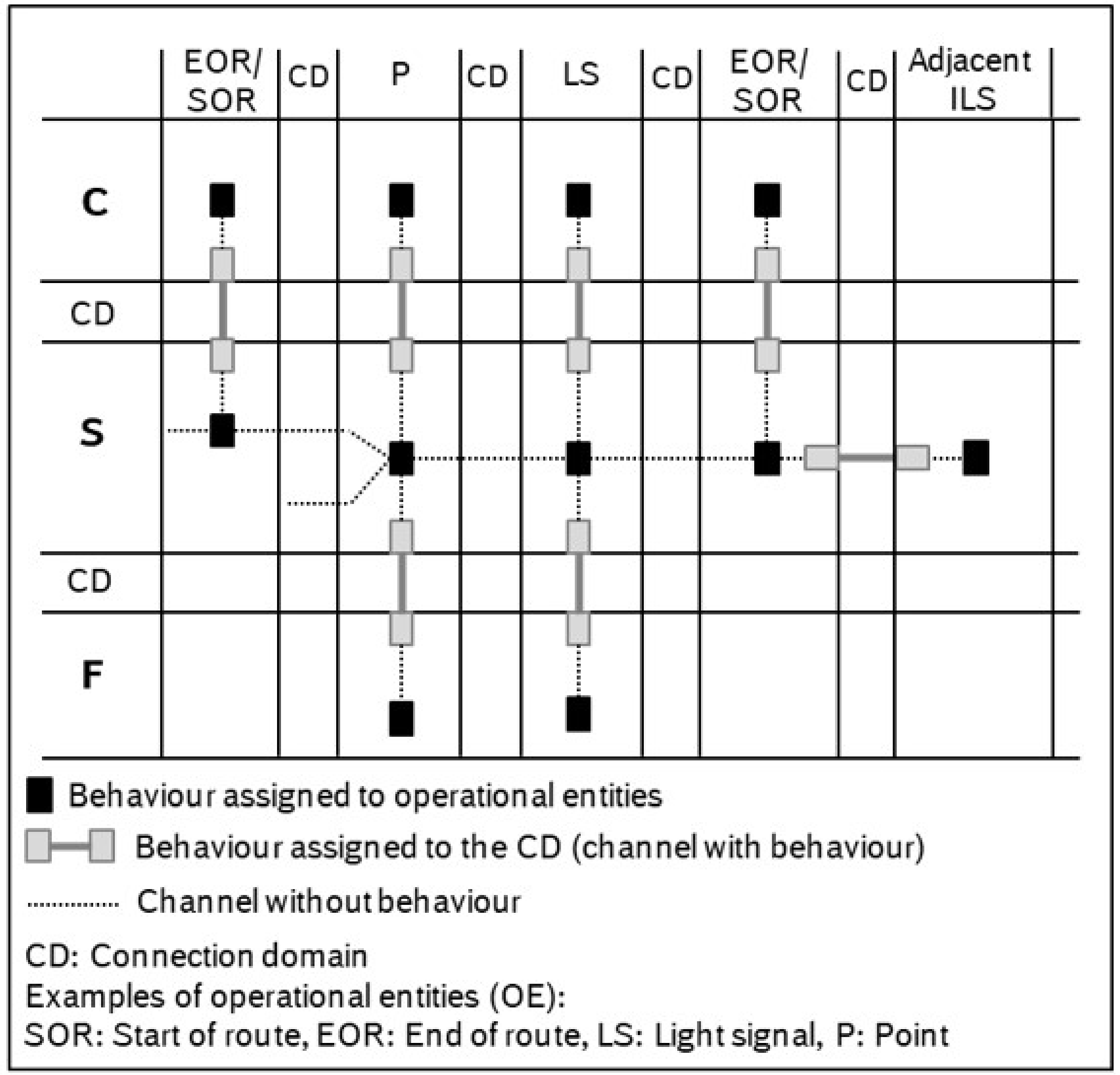
ID	Requirement
Eu.ModSt.2029	Figure 2029 Structure of the diagram headings 
Eu.ModSt.7845	As shown in Figure 7846 as an example, the packages in which the respective model elements are stored are to be displayed on the diagrams.

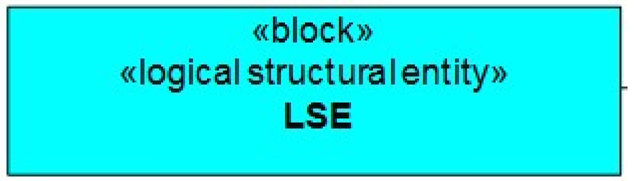
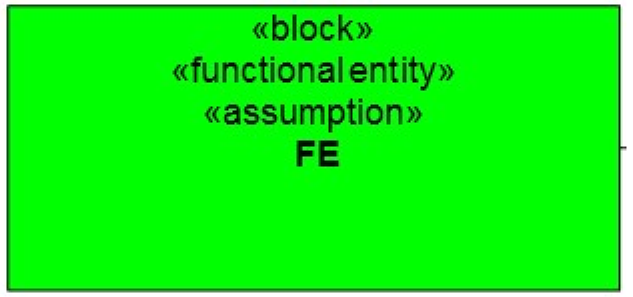
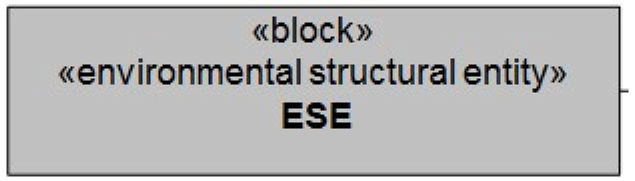
ID	Requirement
Eu.ModSt.7846	Figure 7846 Mapping the package structure onto the diagrams 
Eu.ModSt.7707	In the following *subsections 8.2.1, 8.2.2 and 8.2.3*, the binding of requirements, the modelling pattern for interlocking systems supporting the EULYNX methodology and the basic structural model elements used are introduced.
Eu.ModSt.7065	**8.2.1 Binding nature of the requirements and their structuring**
Eu.ModSt.2030	The SUS and SIUS SysML specification models are stored in the repository of the modelling tool. Relevant artefacts of them are depicted in a traceable manner as surrogates in the requirement specification documents in the form of atomic referenceable functional SUS or SUIS requirements.
Eu.ModSt.7060	Each of these atomised requirements is assigned a liability in the form of an object type. A distinction is made between the object types "Req", "Def", "Info" and "Head".
Eu.ModSt.7061	• "**Req**": This denotes a mandatory requirement.
Eu.ModSt.7062	• "**Def**": This denotes referenceable model elements that are used in the model-based creation of requirements
Eu.ModSt.7063	• "**Info**": This denotes additional information to help understand the specification. These objects do not specify any additional requirements.
Eu.ModSt.7064	• "**Head**": This denotes chapter headings.
Eu.ModSt.7937	**Please note:** State machines or several state machines linked together in a Functional Architecture define the totality of all functional requirements of an SUS or an SIUS in a coherent and consistent manner. State diagrams of a corresponding state machine are marked with the object type "**Req**". For the later design and implementation, it is not the description language SysML that is binding, but the domain-specific meaning expressed by it. The specified behaviour can be converted into a vendor specific language but must retain the domain specific meaning describing the functional requirements. The specific model elements are additionally specified and defined by object type "**Def**" to allow for traceability to supplier designs or test cases. The compliance of products to the specifications must be demonstrated by testing against EULYNX test cases, which are derived from the functionality specified by the models.
Eu.ModSt.7896	**Please note:** The bindings assigned to each model view in this document can be adjusted on a project-specific basis. Thus, the bindings assigned in the specifications always apply.
Eu.ModSt.2031	A functional requirement consists of the respective SysML model element, for instance a SysML diagram, and if necessary, an additional extension of it.

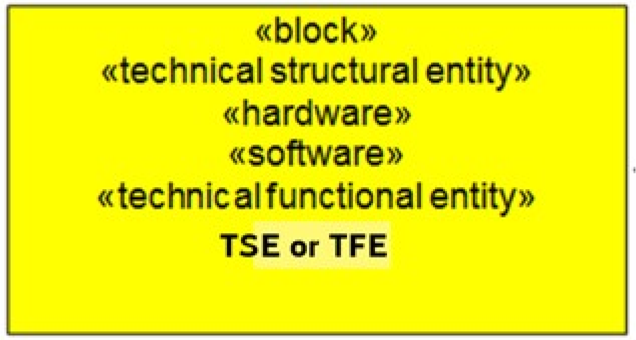
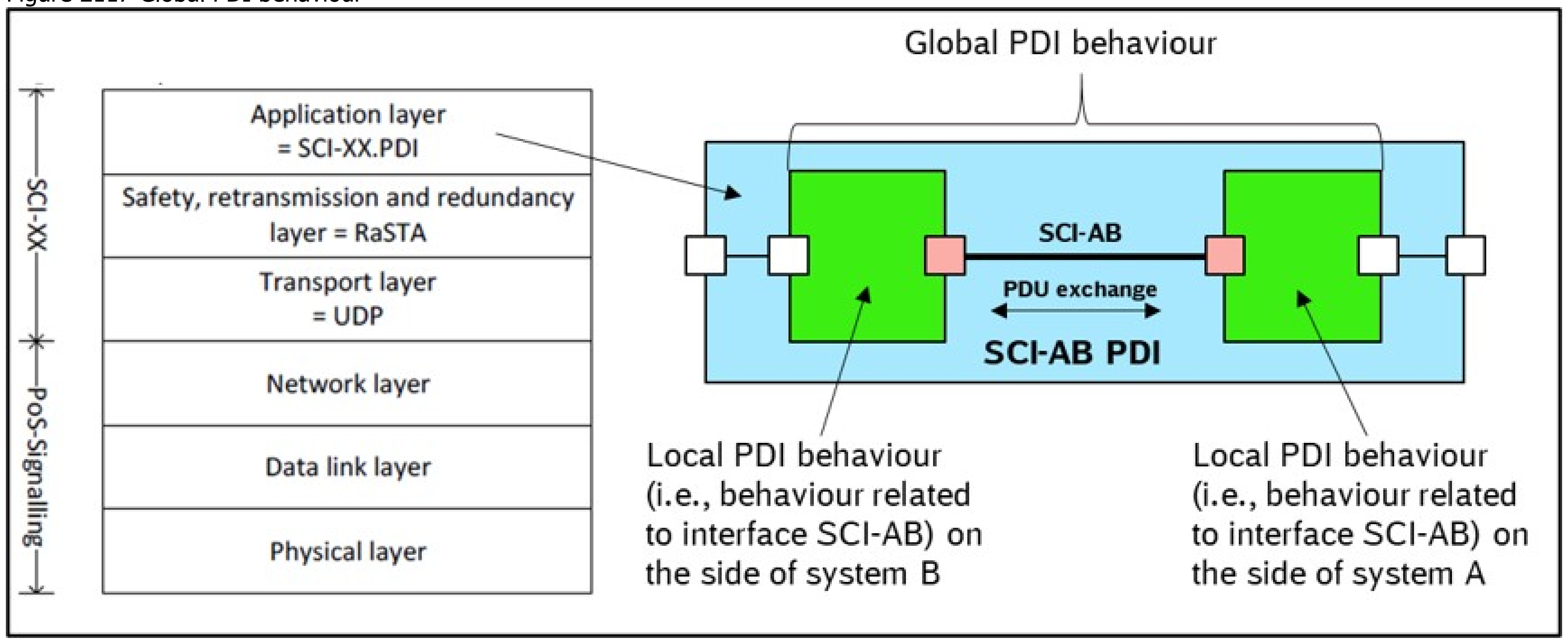
<table>
<tr><th>ID</th><th>Requirement</th></tr>
<tr><td>Eu.ModSt.2032</td><td>For this reason, functional requirements have two attributes "Requirement Part 1" and "Requirement Part 2", which are shown in adjacent columns (see Figure 2).</td></tr>
<tr><td>Eu.ModSt.2033</td><td>In "Requirement Part 1" the respective SysML model element is listed and in "Requirement Part 2" the corresponding extension is shown. Column 'Type' defines the bindingness of the requirement and applies normally both to "Requirement Part 1" and "Requirement Part 2".</td></tr>
<tr><td>Eu.ModSt.2034</td><td>In the case of requirements with a binding character "Req", in which the "Requirement Part 2" is provided with the heading "Information", the defined binding character "Req" only applies to "Requirement Part 1".</td></tr>
<tr><td>Eu.ModSt.2035</td><td>Figure 2: "Requirement Part 1" and "Requirement Part 2" as shown in the requirement specifications.
<table>
<tr><th>ID</th><th>Type</th><th>Requirement Part 1</th><th>Requirement Part 2</th></tr>
<tr><td>Eu.LS.4687</td><td>Req</td><td>Cd_Indicate_Signal_Aspect</td><td>Command (Cd) from the Subsystem - Electronic Interlocking to the Subsystem - Light Signal to indicate the transmitted Signal Aspect.</td></tr>
</table>
</td></tr>
<tr><td>Eu.ModSt.2036</td><td>Just this partition of requirements is applied throughout the entire requirement specification document regardless of whether a requirement has its origins in the SUS or SIUS model or it is for example a text-based nonfunctional requirement manually added.</td></tr>
<tr><td>Eu.ModSt.7704</td><td>8.2.2 Modelling Pattern for interlocking systems</td></tr>
<tr><td>Eu.ModSt.220</td><td>Assuming that the stimulus-response behaviour of an overall interlocking system is immanently allocated to the infrastructure elements and encapsulated in each, the vertical slices of a Modelling Pattern for an overall interlocking system as depicted in Figure 226, may be derived in form of a generic topological abstraction of the signalling infrastructure, i.e. following the geographical principle.</td></tr>
<tr><td>Eu.ModSt.221</td><td>This assumption has already been verified by the implementation of the all-relay interlocking in which the logic of routes is designed following the geographical principle (e.g. the Sp DRS 60 interlocking of Siemens AG as described in [18]).</td></tr>
<tr><td>Eu.ModSt.222</td><td>The geographical principle considers the interconnection of distinct pieces of functionality, immanently encapsulated in the infrastructure elements (ISE), in the form of modules according to the signal layout plan (topological abstraction of infrastructure).</td></tr>
<tr><td>Eu.ModSt.223</td><td>Hence, the functional structure within each vertical slice of the Modelling Pattern for an overall interlocking system may be derived from ISE specific behaviour and interconnected according to the signal layout plan (see Figure 226).</td></tr>
<tr><td>Eu.ModSt.224</td><td>Each of the vertical slices, i.e. each OE, represents the stimulus-response behaviour of a corresponding ISE.</td></tr>
<tr><td>Eu.ModSt.1237</td><td>The goal is to define the stimulus-response behaviour assigned to a vertical slice in a way that it fits into all valid variants of signal layout plans.</td></tr>
<tr><td>Eu.ModSt.1163</td><td>The OEs communicate as appropriate with one another, i.e. they exchange information.</td></tr>
<tr><td>Eu.ModSt.1164</td><td>Each information is sent out by a sender and received by one or multiple receivers. One of these is an OE; the other is an adjacent OE.</td></tr>
<tr><td>Eu.ModSt.1165</td><td>During its transmission, an information passes through a communication channel, which is the path through which the information travels from the sender to the receiver. This communication channel is assigned to the connection domain (CD).</td></tr>
<tr><td>Eu.ModSt.1166</td><td>If the information is given directly by the sender to the receiver a communication channel may be abstracted without specifying any behaviour.</td></tr>
<tr><td>Eu.ModSt.1167</td><td>In other cases, the communication channel is significant because in it information may be delayed, lost, transformed into a format more convenient for the receiver or ordered in time. In these cases, the behaviour of the communication channel is to be modelled explicitly.</td></tr>
</table>

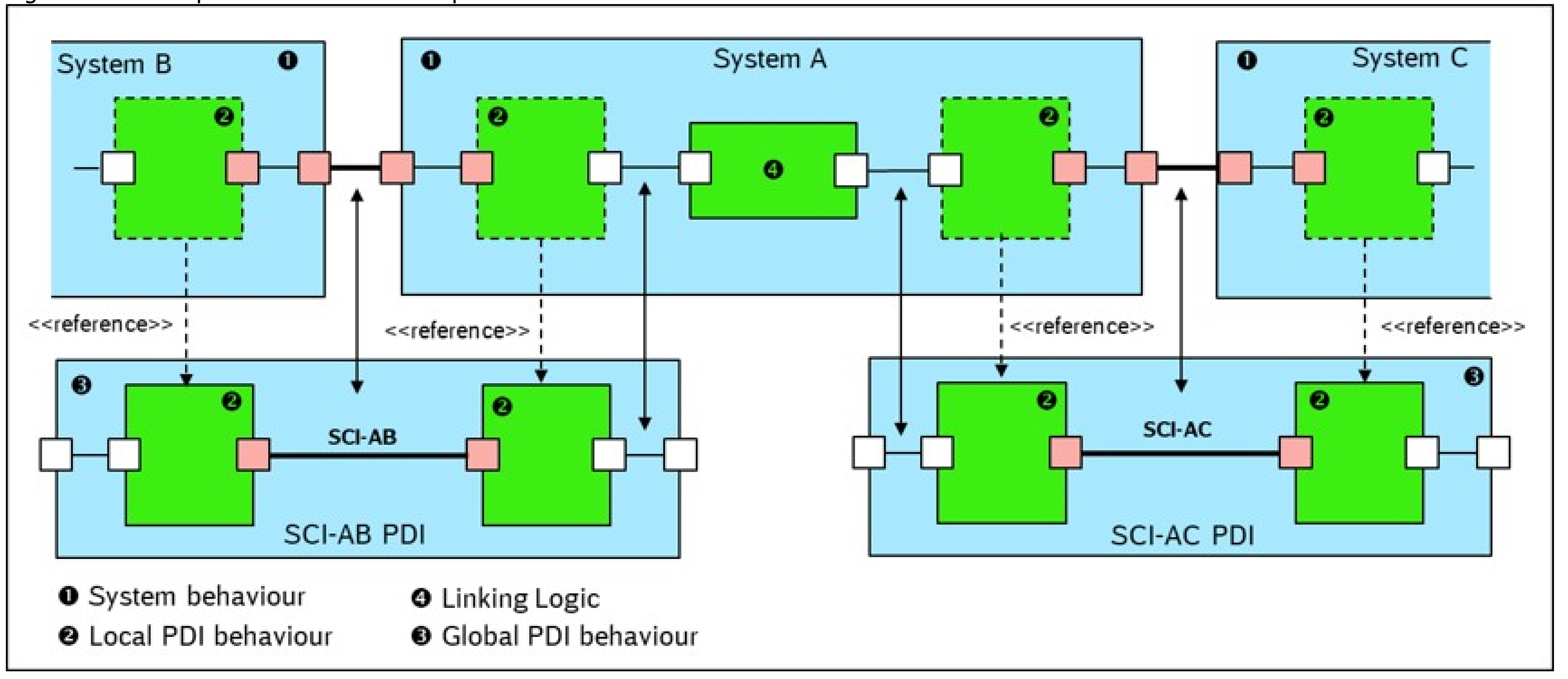
ID	Requirement
Eu.ModSt.226	Figure 226 vertical slices of the Modelling Pattern for interlocking systems 
Eu.ModSt.227	The layers of a Modelling Pattern for an overall interlocking system may be derived from architectural requirements based on the present architecture of an interlocking system [12] (see *Figure 228*): • **Command Control Layer** (acronym: **C**), • **Safety Layer** (acronym: **S**), • **Field Layer** (acronym: **F**).
Eu.ModSt.1168	The OEs exchange information between the different architectural layers as appropriate.
Eu.ModSt.1169	Each information has a sender and one or multiple receivers. One of these is a certain architectural layer of an OE; the other is the underlying or overlying architectural layer of this OE.
Eu.ModSt.1170	In the same way as between the vertical slices described above each information passes through a communication channel assigned to the CD. It connects sender and receiver and may have a behaviour or not.

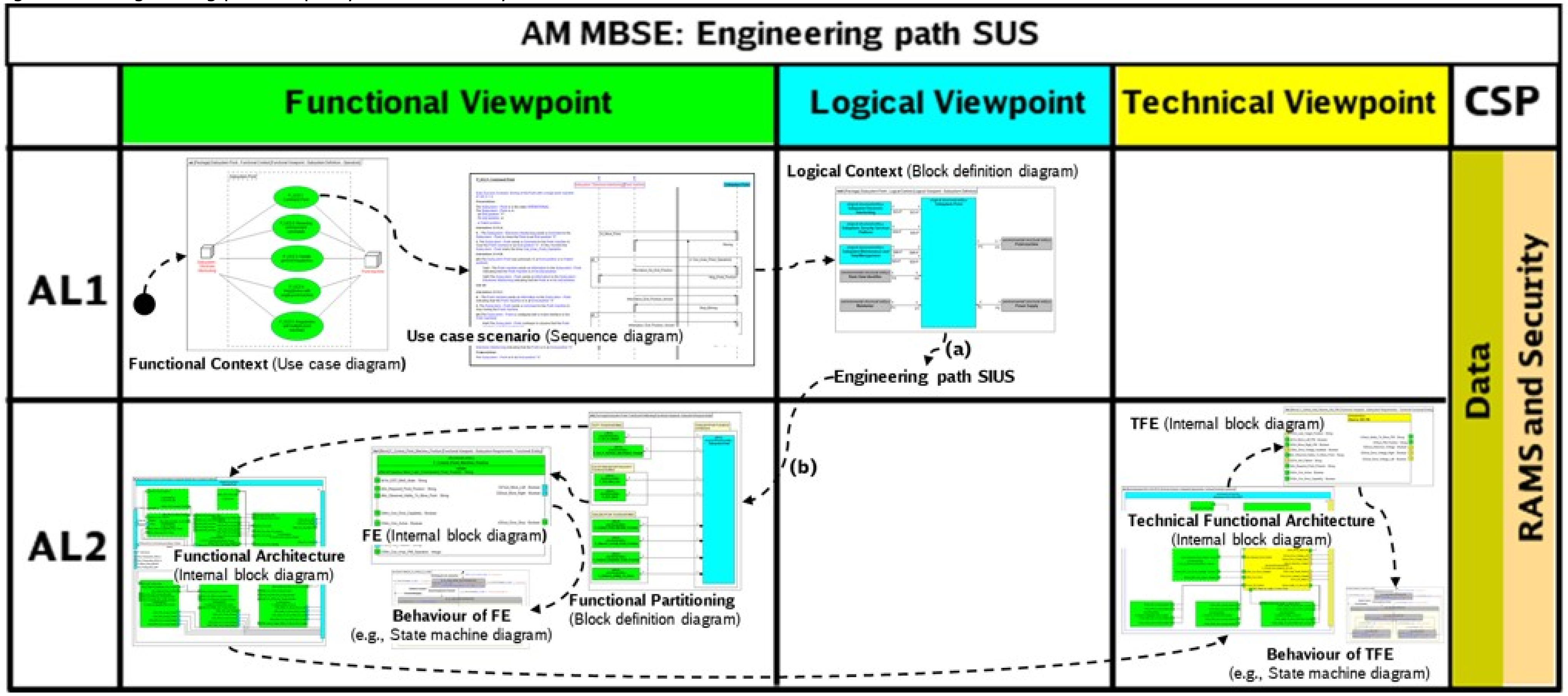
ID	Requirement
Eu.ModSt.228	Figure 228 Architectural layers of the Modelling Pattern for interlocking systems 
Eu.ModSt.231	The Modelling Pattern for interlocking systems, as depicted in principle in *Figure 230*, consists of vertical slices representing the required stimulus-response behaviour of corresponding OEs such as "Light Signal" or "Point" and adjacent vertical slices in which the behaviour of the CD is to be specified.
Eu.ModSt.1172	At the architectural layers **C**, **S** and **F**, the stimulus-response behaviour of the operational entities is put into the perspective of architectural requirements. The CD is to be specified at the underlying or overlying layer of the architectural layer **S**, respectively.
Eu.ModSt.232	Each cell of the so-defined matrix represents a piece of required stimulus-response behaviour of the corresponding OE, put into the perspective of architectural requirements inherent in the respective architectural layer.
Eu.ModSt.1292	This aforementioned behaviour is described in each cell by a FE or a number of FEs that are interconnected in a Functional Architecture.
Eu.ModSt.7705	A Functional Architecture divides the behaviour into Functional Entities, which communicate with each other via internal interfaces and with the environment via external interfaces.
Eu.ModSt.1294	A distinction is made between cells containing the behaviour assigned to OEs and those containing the behaviour of the CD.
Eu.ModSt.1293	The behaviour assigned to the CD specifies the **c**ommunication channel (i.e. the global behaviour of the application protocol RCP) between cells containing the behaviour of adjacent OEs (*see chapter 8.2.4 Interface centric specification*).
Eu.ModSt.7706	Channels without behaviour are represented by SysML connectors that connect the ports of the respective FEs.

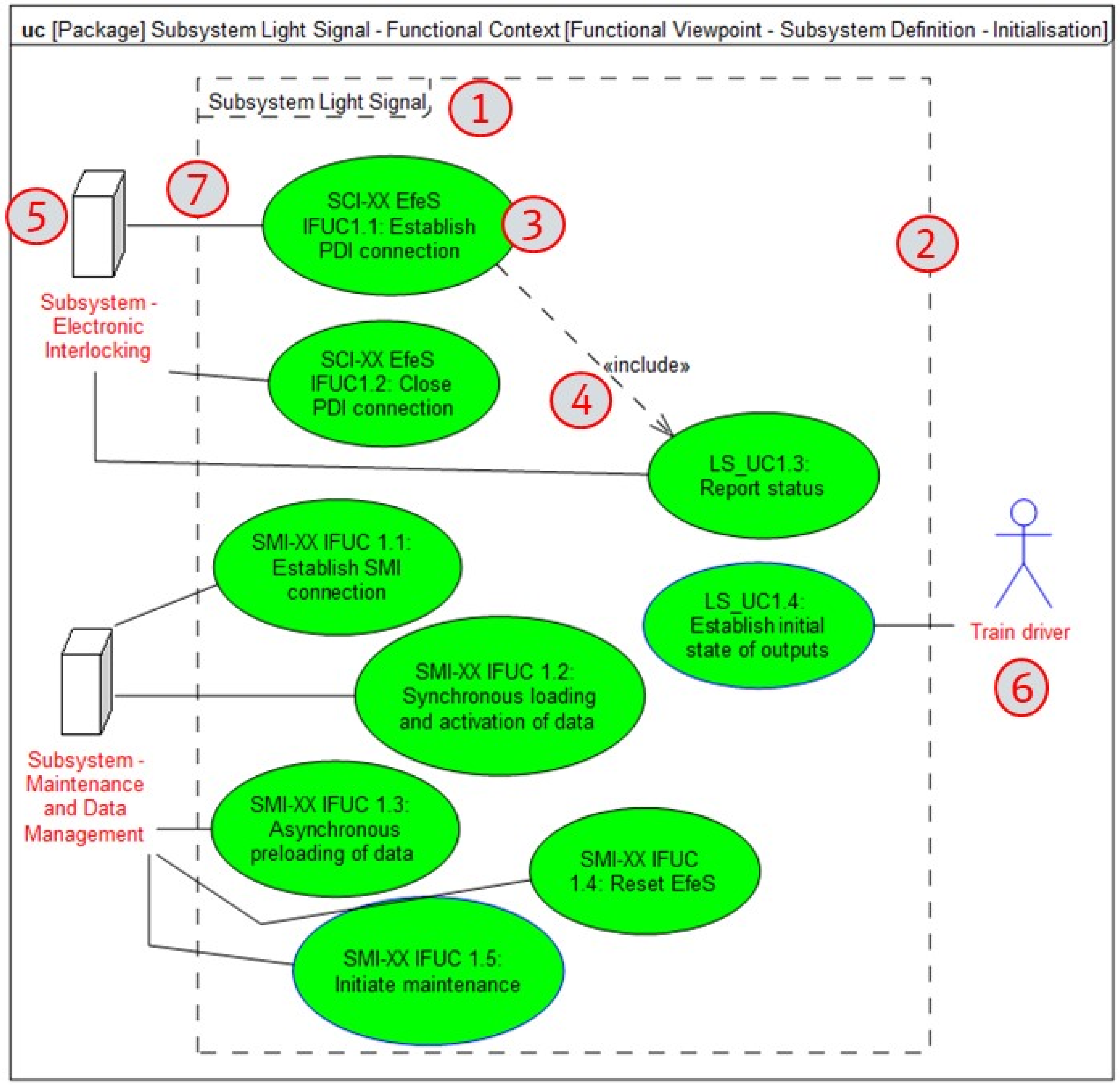
ID	Requirement
Eu.ModSt.230	Figure 230 Principle of a Modelling Pattern for interlocking systems (simplified) 
Eu.ModSt.2091	**8.2.3 Introduction of the basic structural model elements**
Eu.ModSt.2092	**8.2.3.1 Logical Structural Entity (LSE)**
Eu.ModSt.2093	A Logical Structural Entity (block in turquoise, stereotyped as <<logical structural entity>>) represents a system element from a logical point of view. It encapsulates either one or more LSEs interconnected in the form of a Logical Architecture or one or more FEs interconnected in the form of a Functional Architecture.
Eu.ModSt.1243	LSEs representing architectural entities are applied in order to structure a SUS according to architectural aspects aiming at a logical system architecture solution independent from any technological constraints. This kind of partitioning results in a glass box view of the SUS.
Eu.ModSt.355	In a glass box specification the SUS is described as a collection of subsystems.
Eu.ModSt.205	LSEs that are not required to be further decomposed by other LSEs are referred to as atomic LSEs.
Eu.ModSt.1101	The stimulus-response behaviour of a non-atomic LSE is represented by the interactions between its decomposed subcomponents and the interactions of those subcomponents with the interfaces of the SUS. These interactions are described by use case scenarios.
Eu.ModSt.203	Each atomic LSE encapsulates a piece of the "total" external visible stimulus-response behaviour of a SUS. This behaviour may be modularised by Functional Entities (black box view of a SUS).
Eu.ModSt.354	In a black box specification only the black box behaviour of the system to be specified is considered, i.e. only the external properties of the system are defined (externally visible input/output behaviour).

ID	Requirement
Eu.ModSt.2094	Figure 9 Logical Structural Entity 
Eu.ModSt.2095	**8.2.3.2 Functional Entity (FE)**
Eu.ModSt.2096	A functional entity (green block, stereotyped with <<functional entity>>) encapsulates a certain portion of technology-independent system behaviour of a system element.
Eu.ModSt.1247	FEs representing behavioural entities are applied to modularise the stimulus-response behaviour of an atomic LSE aiming at reusability and mastering the complexity. This kind of partitioning does not have any impact on system architectural aspects i.e. the atomic LSE remains a black box. A FE is not further decomposable.
Eu.ModSt.1102	The syntactic interface of a FE defines primarily the signatures of the in ports and the out ports and as appropriate the signatures of block properties and block operations**.** The semantic interface specifies the stimulus-response behaviour, i.e. the chronological order of stimuli and responses using a state machine. The syntactic interface as well as the semantic interface of a FE are explained in detail in the *chapters 8.5 and 8.6*.
Eu.ModSt.2097	A functional entity additionally stereotyped with <<assumption>>represents a set of assumptions which are not functional requirements. Assumptions are mainly used to restrict the environment of a FE.
Eu.ModSt.2098	Figure 10 Functional Entity 
Eu.ModSt.2099	**8.2.3.3 Environmental Structural Entity (ESE)**
Eu.ModSt.2100	In the environment of a SUS, there may be other system elements belonging to the same overall system (subsystems) with which the SUS in question has a communication relationship. These system elements are described by logical structural entities. However, the SUS can also have a relationship with system elements that are outside the associated overall system. These system elements are described by environmental structural entities (grey block, stereotyped with <<environmental structural entity>>).
Eu.ModSt.2101	Figure 11 Environmental Structural Entity 
Eu.ModSt.2102	**8.2.3.4 Technical Structural Entity (TSE) or Technical Functional Entity (TFE)**
Eu.ModSt.2103	**Technical Structural Entity:** A Technical Structural Entity (yellow-coloured SysML block stereotyped with <<technical structural entity>>) encapsulates one or more TSEs in the form of a Technical Architecture or one or more TFEs interconnected in the form of a Technical Functional Architecture based on technical requirements (<<hardware>>: TSE representing a hardware artefact, <<software>>**:** TSE representing a software artefact).
Eu.ModSt.2104	**Technical Functional Entity:** 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 in a Technical Functional Architecture supplementing or substituting the technology-independent behaviour defined by FEs.

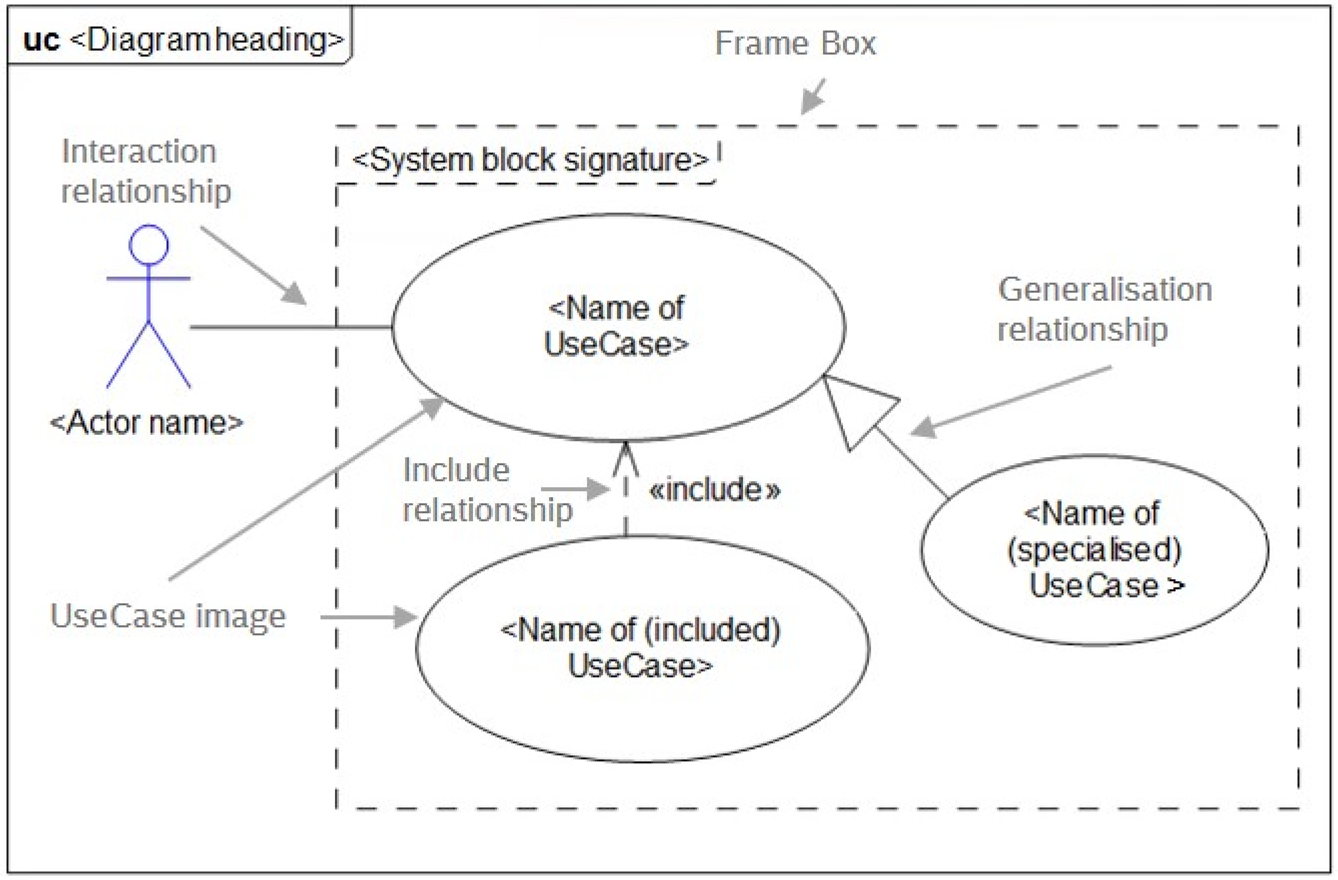
ID	Requirement
Eu.ModSt.2105	Figure 12 Technical Structural Entity or Technical Functional Entity 
Eu.ModSt.2106	**8.2.3.5 Information objects**
Eu.ModSt.2107	Information objects are the objects that are exchanged between the respective communication partners via a communication relationship. They are formed from signals and values of the signals, the so-called attributes and are made available or received at ports.
Eu.ModSt.2108	Ports are represented by small squares at the edge of a Functional Entity and represent the connections to the interfaces to other internal or external Functional Entities to which a communication relationship exists, or to external interfaces. The port also indicates the arbitrary port name and interface type in the format "port name:interface type". Communication relationships between functional entities are assigned a reading direction. In the case of ports, this is represented by the interface type being shown in conjugated form, i.e. by the symbol "~", on one side of the communication relationship.
Eu.ModSt.2109	**8.2.4 Interface centric specification**
Eu.ModSt.2112	By an interface centric approach**,** it is understood that the external visible stimulus-response behaviour (usage behaviour) of a SUS 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.
Eu.ModSt.2113	As depicted in *Figure 2117*, 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 (e.g., SCI-AB PDI).
Eu.ModSt.2114	**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 a chronological order.
Eu.ModSt.2115	The **local PDI behaviours** represent the behaviours of the communicating systems related to a certain interface.
Eu.ModSt.2116	The relation between local PDI behaviour and global PDI behaviour can be illustrated by a telephone call. The dialling is a local PDI behaviour at the initiator side, the ringing the associated local PDI behaviour at the partner side. Only the global PDI behaviour defines that the dialling must precede the ringing (i.e., the chronological order).
Eu.ModSt.2117	Figure 2117 Global PDI behaviour 

ID	Requirement
Eu.ModSt.2118	As the local PDI behaviours represent the interface behaviours of the communicating systems they may be specified in the model of the PDI.
Eu.ModSt.2119	As depicted in *Figure 2120*, in the model of a SUS such as System A, these local PDI behaviours are referenced and linked together (Linking Logic).
Eu.ModSt.2120	Figure 2120 Principle of interface centric specification 
Eu.ModSt.7952	**8.2.5 Functional packages**
Eu.ModSt.7953	The EULYNX specifications are to be divided into functional packages in the requirements management tool used. This is intended to enable Infrastructure Managers (IM) involved to select requirements in a targeted manner and thus apply the specifications to the desired capabilities of their products.
Eu.ModSt.7954	There are two types of packages that relate to product capabilities: • 'Basic packages', i.e. one or more packages, at least one of them must be implemented. It is allowed to combine and implement more than one 'basic package' in a product. • 'Optional package', i.e. one or more packages that can be optionally implemented in addition to one or more basic packages.
Eu.ModSt.7955	For the evaluation if a requirement is valid or not depending on the selected functional packages of an IM, the basic packages have an "or" relation and optional packages have an "and" relation to everything else. I.e. from mathematical point of view: ("Basic P1" or "Basic P2" or "Basic Pn") and "Option P1".
Eu.ModSt.7956	The functional packages are to be allocated to the requirements in the requirements management tool used. The practical implementation of the allocation depends on the capabilities of the tool.
Eu.ModSt.7957	The SysML specification model must be structured in such a way that the required functional packages can be separated from the overall functionality in order to enable clear allocation as described above.
Eu.ModSt.7958	For example, functional packages can be formed by encapsulating certain behaviours in functional entities, which are then used or not in the corresponding functional architecture as required.
Eu.ModSt.1509	**8.3 Model views used to specify EULYNX subsystems**
Eu.ModSt.2124	**Model view "Functional Context": Use case Diagram (uc)** 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.
Eu.ModSt.2125	**Model view "Use case scenario": Sequence Diagram (sd)** The model view "Use case scenario" describes the behaviour of the use cases defined in the model view "Functional Context" at the upper level of abstraction by means of one or more use case scenarios.
Eu.ModSt.2123	**Model view "Logical Context": Block Definition Diagram (bdd)** The model view "Logical Context" describes at the top level • the system/subsystem under specification (SUS), • the actors in the environment interacting with the SUS and their quantity structure (multiplicities) as well as the logical interfaces between the SUS and the actors.

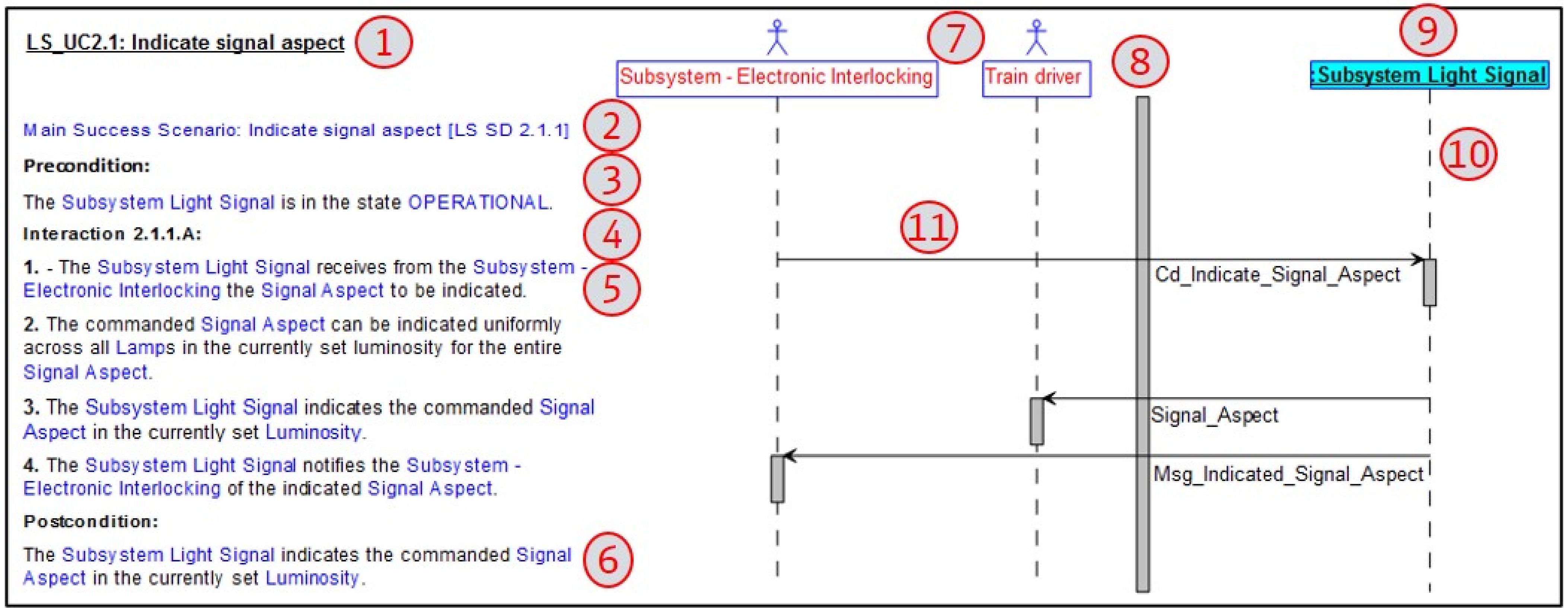
ID	Requirement
Eu.ModSt.7708	**Model view "Functional Partitioning": Block Definition Diagram (bdd)** The model view "Functional Partitioning" describes the refinement of the SUS by means of the FEs defined in the SIUS model view "Functional Partitioning", which represent the local behaviours of the PDI, as well as the FEs specific to the SUS (linking behaviour according to *chapter 8.2.4*).
Eu.ModSt.2126	**Model view "Functional Architecture": Internal Block Diagram (ibd)** The model view "Functional Architecture" refines or completes the behaviour of an 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. The FEs are defined in model view "Functional Partitioning".
Eu.ModSt.7720	**Model view "Technical Functional Architecture": Internal Block Diagram (ibd)** The model view "Technical Functional Architecture" supplements the behaviour described in the model view "Functional Architecture", which is independent of technology, with behavioural components derived from technical requirements. Either the entire behaviour can be described in a technical context or a mixture of functional and technical aspects.
Eu.ModSt.2127	**Model views "Functional Entity" and "Technical Functional Entity": Internal Block Diagram (ibd) and State Machine (stm)** The model view "Functional Entity" encapsulates a subset of technology-independent functional requirements and the model view "Technical Functional Entity" a subset of technology-dependent 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 FE is described by means of state machines. In this, the binding functional requirements are specified in the form of state transitions. Both model views are described in the separate *chapters 8.5 and 8.6*.
Eu.ModSt.2128	*Figure 2129* shows the engineering path of the model views used to specify a SUS considering the Functional Viewpoint, the Logical Viewpoint and the Technical Viewpoint. It describes the context of the model views, with the arrows indicating which model views are developed from which. During the development of the model, the model views "Functional Context" (the Use Cases), "Use case scenarios" and "Logical Context" are created. These model views form the basis for the description of the model views "Functional Partitioning", "Functional Architecture" and "Functional Entity". For the creation of the model view "Functional Partitioning", the FEs defined in the model view "Functional Partitioning" of the SIUS are required (b: see *Figure 2244* in *chapter 8.4*). In case technical requirements are to be considered, the model views "Technical Functional Architecture" and "Technical Functional Entity" are created based on the model view "Functional Architecture".
Eu.ModSt.2129	Figure 2129 Engineering path to specify a EULYNX subsystem 
Eu.ModSt.3550	**8.3.1 Model View "Functional Context" of a SUS (AL1) - Description**
Eu.ModSt.3495	The model view "Functional Context" as shown in *Figure 3496* defines the services to be provided by the SUS in the form of use cases. On one or more SysML use case diagrams all subsystem use cases and their relationships to the SUS environment and between the subsystem use cases themselves are depicted.

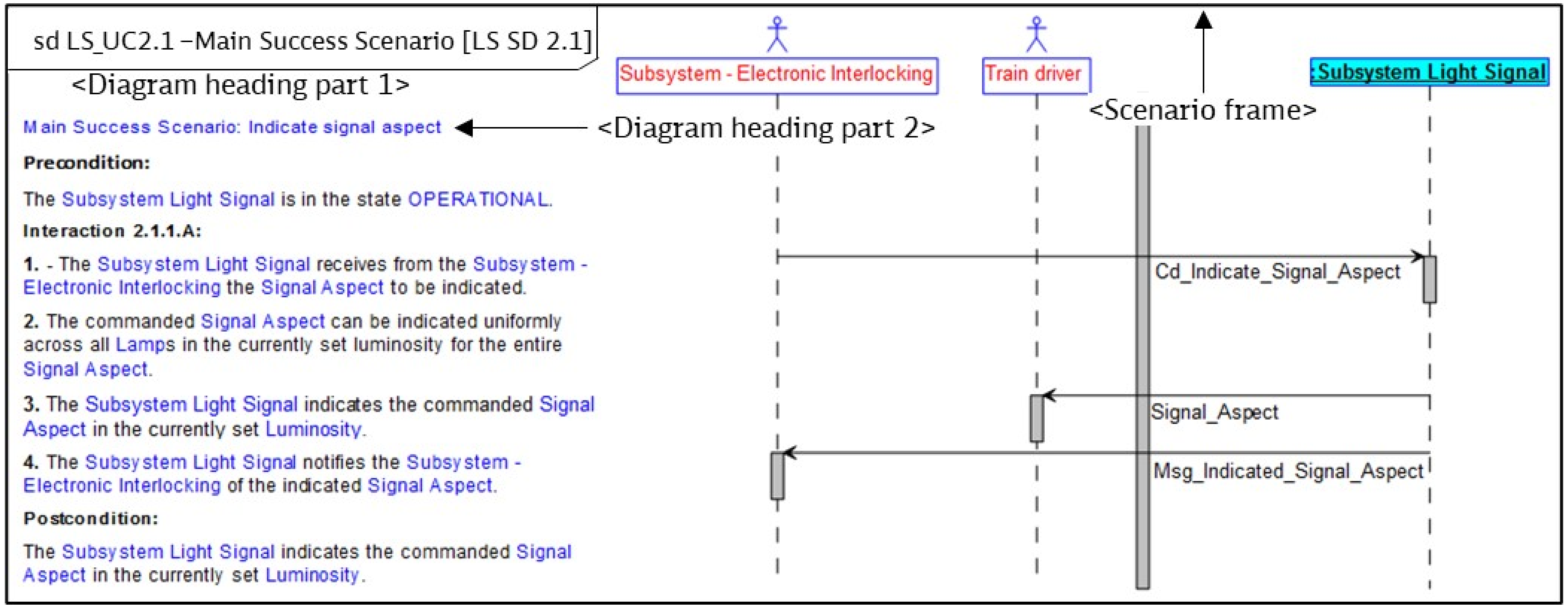
ID	Requirement
Eu.ModSt.3497	In the use case diagrams, the boundary **(2)** of the SUS **(1)** is shown as a frame with a dotted line.
Eu.ModSt.3498	The use cases of the SUS are shown as ellipses within the frame and have the name of the respective use case **(3)**.
Eu.ModSt.3499	A use case describes a service a SUS provides to its environment and is specified by one or more interaction scenarios (model view "Use case scenario").
Eu.ModSt.3500	Use cases are connected by interaction connectors **(7)** to those actors in the SUS environment with whom they interact. An actor may represent another system **(5)** or a person **(6)**.
Eu.ModSt.3501	Use cases may be connected to each other through include relationships **(4)**, which are represented by arrows with a dashed line stereotyped with <<include>>. Such a relationship indicates that the interaction scenarios of the use case at the arrowhead are included in the use case at the other end of the arrow. These included use cases encapsulate services that occur more than once, for example, and can also be included in other use cases.
Eu.ModSt.3496	Figure 3496 Example of SUS model view "Functional Context" 
Eu.ModSt.7711	**8.3.2 Model View "Functional Context" of a SUS (AL1) - Modelling rules**
Eu.ModSt.7713	**8.3.2.1 SysML Diagram**
Eu.ModSt.7715	**UseCase diagram (uc)**: depicts the model view "Functional Context" (one or more use case diagrams classified by domain motivated use case groups such as Start-up, Operation, Maintenance and so on).

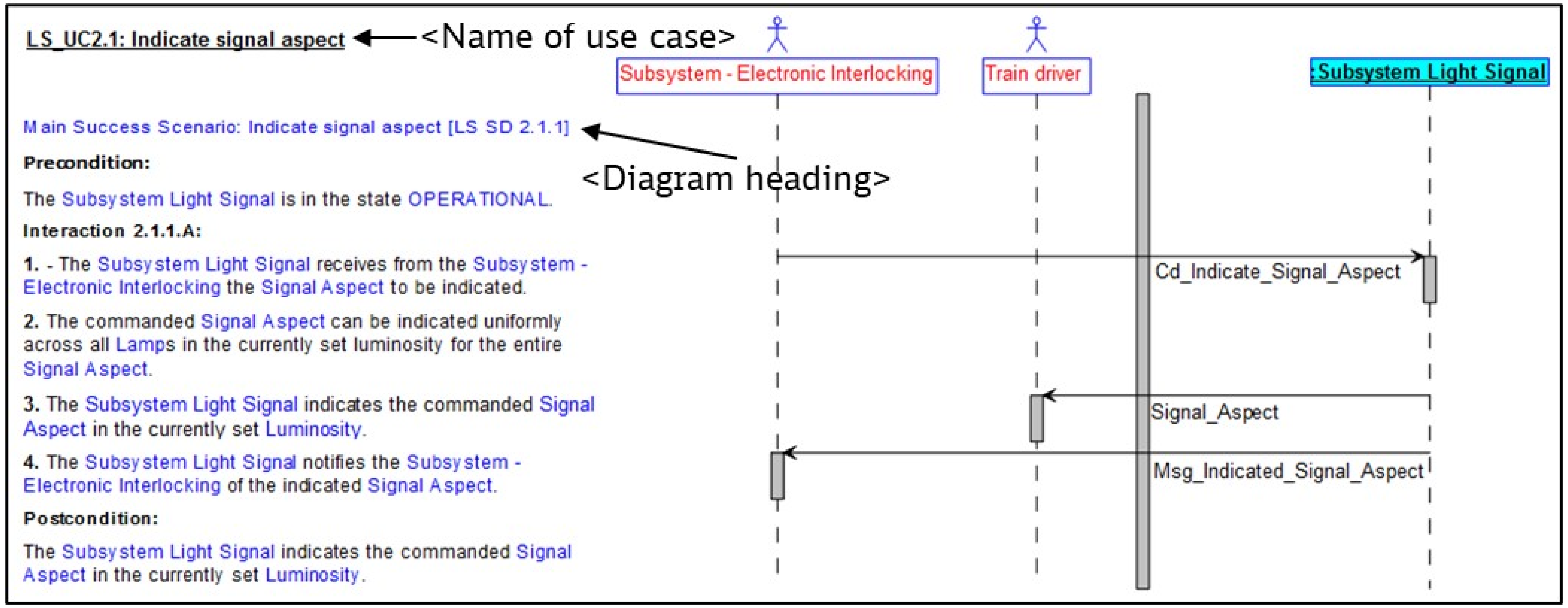
ID	Requirement
Eu.ModSt.7716	**Name of the Diagram:** *uc[Package]<><System Name><>-<>Functional Context<>[Functional Viewpoint<>-<>Subsystem Definition<>-<><Use case group><>DiaNo]*.
Eu.ModSt.7717	**Example:** uc[Package] Subsystem Light signal - Functional Context [Functional Viewpoint - Subsystem Definition - Initialization]
Eu.ModSt.1197	<Use case group> := <Main use case group><>-<><Sub use case group>
Eu.ModSt.1949	<Main use case group> := Broader term of the domain motivated group of services defined on the use case diagram
Eu.ModSt.1950	<Sub use case group> := Broader term of the subdomain motivated group of services defined on the use case diagram
Eu.ModSt.1199	**Examples:** Operation Operation - Direction
Eu.ModSt.1198	<DiaNo> := Number of use case diagram (Natural number starting with 1); optional to use
Eu.ModSt.1200	<Name of Frame Box> := <System block signature>
Eu.ModSt.1201	<Name of use case> := <UC designator>:<><Service to be described>
Eu.ModSt.1952	<UC designator> := <UC type>UC<DiaNo of uc>.<UCNo>
Eu.ModSt.1763	<UC type> := <Abbr. System type>
Eu.ModSt.1202	<UCNo> := Number of UseCase (Natural number).
Eu.ModSt.1203	<Service to be described> := The name of the service required by the system environment.
Eu.ModSt.1204	**Example:** LS_UC1.4: Establish initial state of outputs
Eu.ModSt.1205	<Name of UseCase> (generic UseCase) := <Gen UC designator>:<><Service to be described>
Eu.ModSt.1953	<Gen UC designator> := <Gen UC type>UC<DiaNo of uc>.<UCNo>
Eu.ModSt.1951	<Gen UC type> := Gen | <Abbr. System group>
Eu.ModSt.1955	<Abbr. System group> := Freely selectable designator such as EfeS (EULYNX field element system) or AdjS (adjacent system)
Eu.ModSt.1206	**Example:** EfeSUC1.2: Establish PDI connection GenUC1.4: Establish PDI connection
Eu.ModSt.728	**8.3.2.2 Model elements**
Eu.ModSt.926	The model elements basically used to describe the model view "Functional Context" are depicted in *Figure 746*.

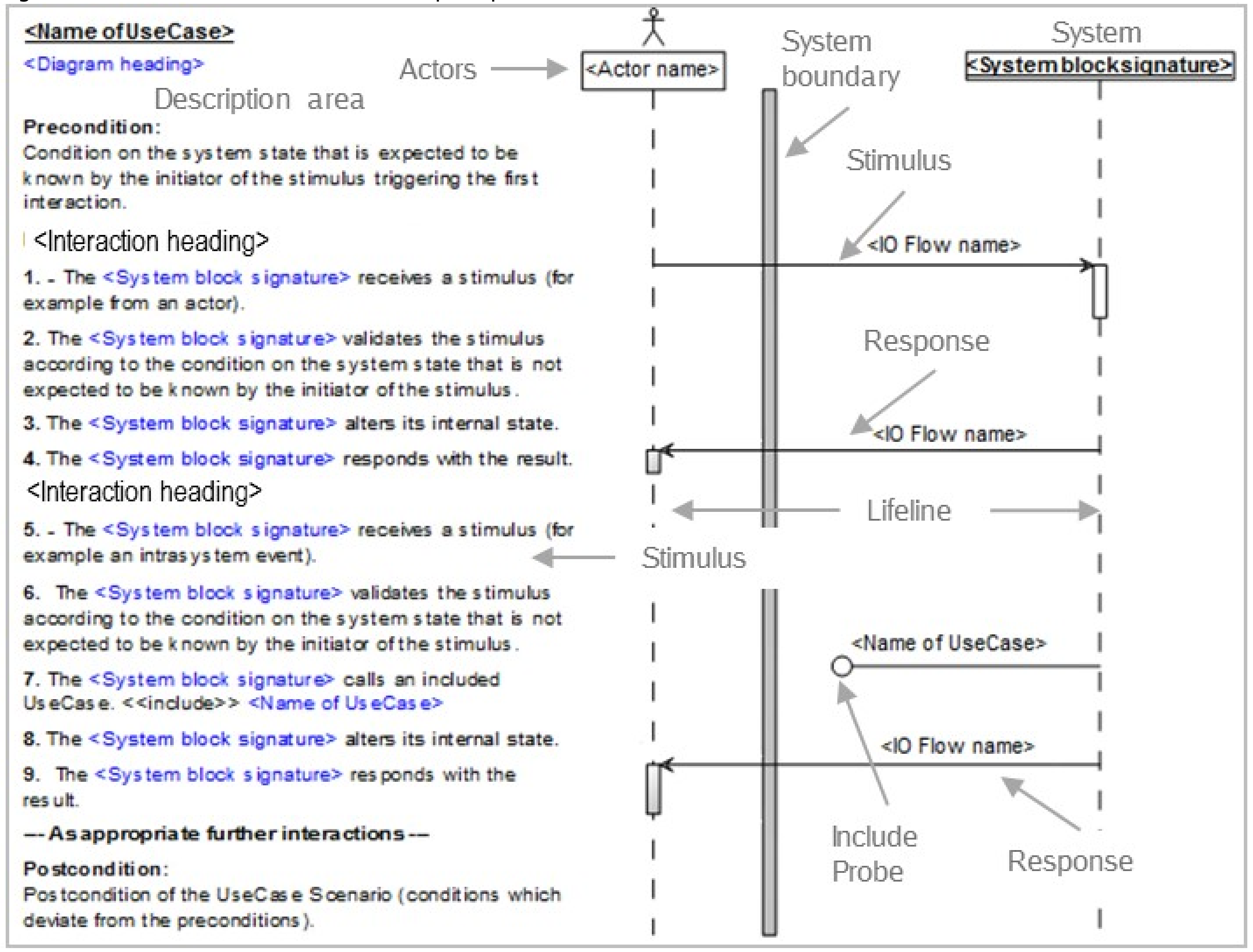
ID	Requirement
Eu.ModSt.746	Figure 746 Basically used model elements of model view "Functional Context" 
Eu.ModSt.729	**Frame Box:** Represents the boundary of the SUS the use cases are allocated to.
Eu.ModSt.731	**UseCase image:** Depicts a UseCase on the use case diagram.
Eu.ModSt.1714	It may be project-specifically determined that for each use case one constraint may be added for each of the following definitions: • the Purpose, • the Primary Actor and • the Secondary Actor.
Eu.ModSt.1715	It may be project-specifically determined that the purpose of the UseCase is to be written in accordance with the following pattern: This UseCase describes the <><UseCase Action><>of<><UseCase Object><>by<><UC Actor/s><><to do/ for doing><><summary of UseCase content>. <Optional free text description to add details about UseCase content>.
Eu.ModSt.1709	**Actor:** As stated earlier, an actor specifies a role played by user or any other system that interacts with the system. Cockburn [22] distinguishes between primary and secondary actors.
Eu.ModSt.1710	**Primary Actor:** The primary actor of a use case is the stakeholder that calls on the system to deliver one of its services. It has a goal with respect to the system – one that can be satisfied by its operation. The primary actor is often, but not always, the actor who triggers the use case.
Eu.ModSt.1711	**Secondary Actor:** The secondary actor of a use case is a stakeholder that the system needs assistance from to achieve the primary actor's goal.
Eu.ModSt.1712	In other words, secondary actors may or may not have goals that they expect to be satisfied by the use case, the primary actor always has a goal, and the use case exists to satisfy the primary actor.
Eu.ModSt.744	**Interaction relationship:** Connects the actors participating in the system use cases to the use case images (see *Figure 746*).
Eu.ModSt.745	The interaction relationship is an abstract representation of the exchange of messages temporally ordered (information flow from and to the system) within the scope of the corresponding SUS use case.
Eu.ModSt.1713	It may be project-specifically determined that only the primary actors participating in the SUS use cases are connected to the use case images. Secondary actors may not be connected for the benefit of the diagram´s readability.

ID	Requirement
Eu.ModSt.1207	**Generalisation relationship:** use cases can be classified using the standard SysML generalisation relationship. The meaning of classification is similar to that for other classifiable model elements. One implication, for example, is that the use case scenarios for the general use case are also use case scenarios of the specialised use case. It also means that the actors associated with a specialised use case can also participate in use case scenarios described by a general use case. Classification of use cases is shown using the standard SysML generalisation symbol (see Fig. 746).
Eu.ModSt.747	**Include relationship:** An include relationship between two UseCases means that the sequence of behaviour described in the included use case is included in the sequence of the base (including) use case.
Eu.ModSt.748	**Please note:** Include relationships are only to be used if absolutely necessary, whereas extends relationships are not to be used at all.
Eu.ModSt.749	The included use case may be a primary use case as well as a secondary use case.
Eu.ModSt.861	When including a use case, this use case shall be named in the description of the sequence.
Eu.ModSt.750	A primary use case is a complete UseCase having a domain trigger, a result, and a primary actor.
Eu.ModSt.751	A secondary use case is an incomplete use case fragment. This is a "piece" of use case that doesn't fulfil at least one of the criteria of a primary use case. It is modelled for example if its flow is part of several (primary) use cases. This allows to avoid redundant descriptions or enables the structured merge of specific behaviour and generic behaviour. "Include" creates a relationship between primary and secondary use cases.
Eu.ModSt.752	In the example depicted in *Figure 3496*, the system-specific use case "LS_UC1.3:Report status" is included in the generic UseCase " EfeSUC1.2: Establish PDI connection".
Eu.ModSt.7075	**8.3.2.3 Binding** (see *chapter 8.2.1*)
Eu.ModSt.7754	**Diagram of model view "Functional Context"** has an "**Info**" binding.
Eu.ModSt.7077	**Use Case** has an "**Info**' binding if it is further specified in a refined model view.
Eu.ModSt.7894	**Use Case** has a "**Req**" binding if it is not further specified in a refined model view.
Eu.ModSt.364	**8.3.3 Model View "Use case scenario" of a SUS (AL1) - Description**
Eu.ModSt.3503	The model view "Use case scenario" as shown in *Figure 3504* defines the behaviour of the use cases defined in the model view "Functional Context" by means of one or more use case scenarios at the upper level of abstraction. These use case scenarios describe the interaction between the SUS and the actors in the SUS environment using SysML sequence diagrams.
Eu.ModSt.3506	**Use case name (1)** Name of the use case to which the interaction scenario belongs (e.g., LS_UC2.1: Indicate signal aspect).
Eu.ModSt.3508	**Use case scenario name (2)** The use case scenario name is the name of a possible information flow (shown as a sequence diagram) within a use case (Main Success Scenario or Alternative Scenario).
Eu.ModSt.3510	**Preconditions (3)** Preconditions are conditions that must be met and known to the actor triggering the stimulus for the scenario to start (see *chapter 8.1.2.1.3*).
Eu.ModSt.3512	**Interaction (4)** An interaction consists of a sequence of steps, starting with a stimulus (prefixed by a dash "-"), a validation, possibly a state change and a reaction. In addition, combined fragments may be included. A use case scenario can consist of one or more interactions. The structure of an interaction follows the principle of the Action Block Scheme as described in *chapter 8.1.2.1.2*.
Eu.ModSt.3514	**Sequences and information flows (5)** Sequences consist of a text part describing the sequence and, in the case of an information flow, a graphical representation of the information flow in the form of arrows between the lifelines **(11)**. In the text part, elements of the model are shown in blue and explanatory text in black. In the graphical part, the corresponding exchange of information objects is shown accordingly. Here in the example (sequence 1), the information object "Cd_Indicate_Signal_Aspect" is sent from the "Subsystem Electronic Interlocking" to "Subsystem Light_Signal". As it is a stimulus it is prefixed by a dash "-" in the text part of the sequence. In sequence 2, the validation of the information object in the "Subsystem Light Signal" is described in the text part, without representation in the graphical part.
Eu.ModSt.3516	**Postconditions (6)** Postconditions are conditions for which changes have resulted from the sequence diagram. Conditions that have already been mentioned in the preconditions are not listed here.
Eu.ModSt.3518	**Actors (7)** Actors are systems (e.g., Subsystem Electronic Interlocking) or persons that interact with the SUS, i.e. trigger a stimulus and/or receive a response.
Eu.ModSt.3520	**System under specification and System boundary (8)** The boundary between the system under specification (SUS) and the actors is symbolised by a thick grey bar. The SUS **(9)** is located to the right of the grey bar and the actors to the left.
Eu.ModSt.3522	**Lifelines (10)** Lifelines represent the time axis of the SUS and the actors, with the time running from top to bottom.

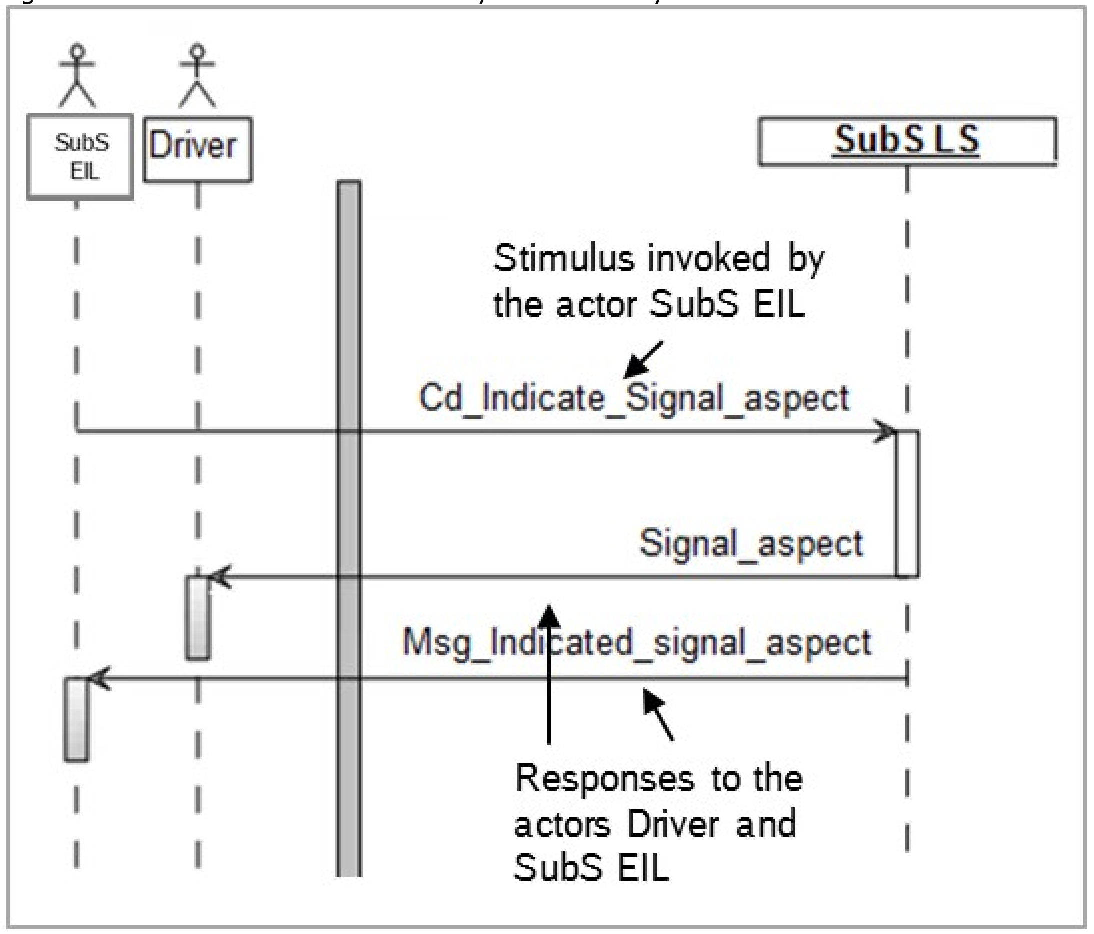
ID	Requirement
Eu.ModSt.3504	Figure 3504 Example of SUS model view "Use case scenario" 
Eu.ModSt.756	**8.3.4 Model View "Use case scenario" of a SUS (AL1) - Modelling rules**
Eu.ModSt.757	**8.3.4.1 SysML diagram**
Eu.ModSt.758	**Sequence Diagram:** A sequence diagram generally shows a stimulus-response behaviour, focusing on the temporal sequence of messages.
Eu.ModSt.759	A sequence diagram depicting a use case scenario shows a specific sequence of messages, i.e. it represents a possible variant of a SUS use case.
Eu.ModSt.760	**In contrast to the complete stimulus-response behaviour of a SUS use case, described using a state machine, a use case scenario only represents a "flash light" view of this behaviour.**
Eu.ModSt.761	There are two variants of use case scenario layouts: • Variant 1: Use case scenario with frame (*Figure 1690*) and • Variant 2: Use case scenario without frame (*Figure 6976*).
Eu.ModSt.1693	It has to be project-specifically determined which variant to apply. The example scenarios in this document are depicted according to variant 2.

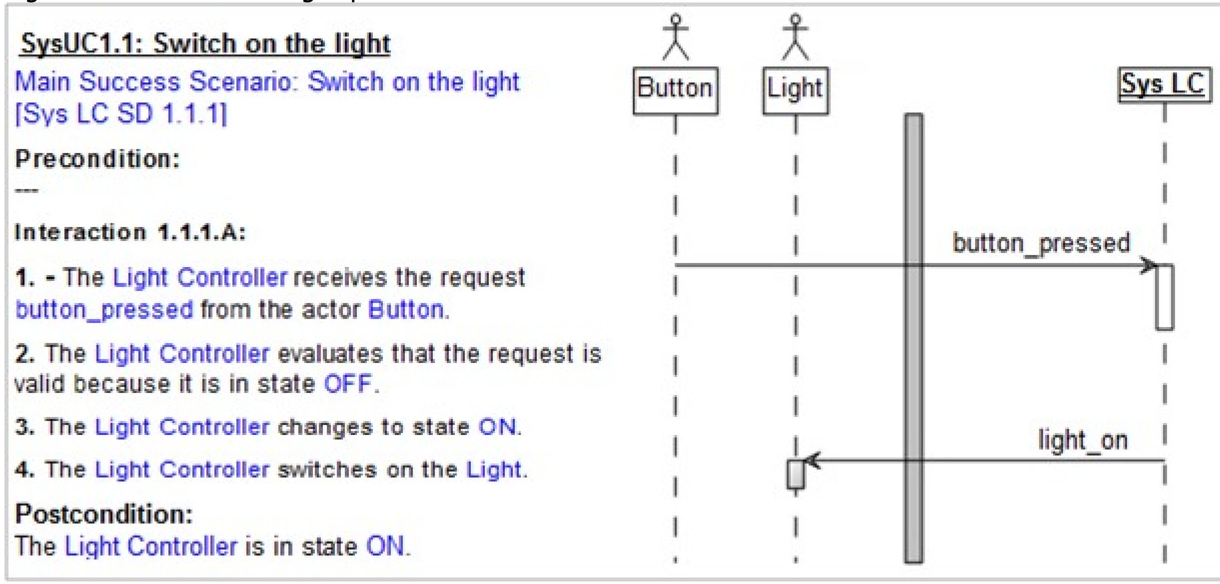
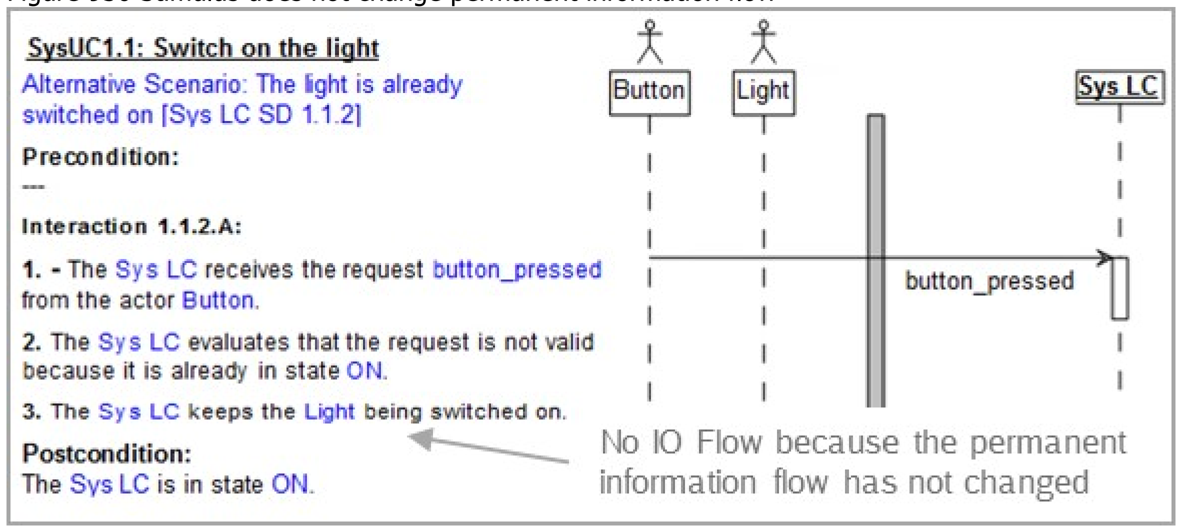
ID	Requirement
Eu.ModSt.1690	Figure 1690 Variant 1: Use case scenario with frame 
Eu.ModSt.1691	**Variant 1: Diagram heading part 1** *sd<><Abbr. System type>UC<DiaNo of UCD>.<UCNo>-<Scenario type><> [<Abbr. System ID><>SD<><DiaNo of UCD>.<UCNo>.<DiaNo of SD>]*
Eu.ModSt.1695	**Variant 1: Diagram heading part 2** ***<Scenario type>:<> <Scenario name>***
Eu.ModSt.766	A use case may be defined by one or more use case scenarios in the following compositions: - one Main Success Scenario and any number of Alternative Scenarios, - only one Main Success Scenario, - any number of Alternative Scenarios without a Main Success Scenario.
Eu.ModSt.1698	**Examples:** sd SubSUC2.1-Main Success Scenario [SubS LS SD 2.1.1] Main Success Scenario: Indicate signal aspect sd SubSUC2.2-Alternative Scenario [SubS LS SD 2.2.2] Alternative Scenario: Illuminant failure
Eu.ModSt.1696	**Variant 1: Diagram heading part 1 (generic UseCase Scenario)** *sd<><Gen UC type>UC<DiaNo of UCD>.<UCNo>-<Scenario type><>[<Gen UC type><>SD<> <DiaNo of UCD>.<UCNo>.<DiaNo of SD>]*
Eu.ModSt.1697	**Variant 1: Diagram heading part 2 (generic UseCase Scenario)** *<Scenario type>:<> <Scenario name>*
Eu.ModSt.1699	**Example:** sd GenUC1.2-Main Success Scenario [Gen SD 1.2.1] Main Success Scenario: Establish PDI connection sd EfeSUC1.2-Main Success Scenario [EfeS SD 1.2.1] Main Success Scenario: Establish PDI connection

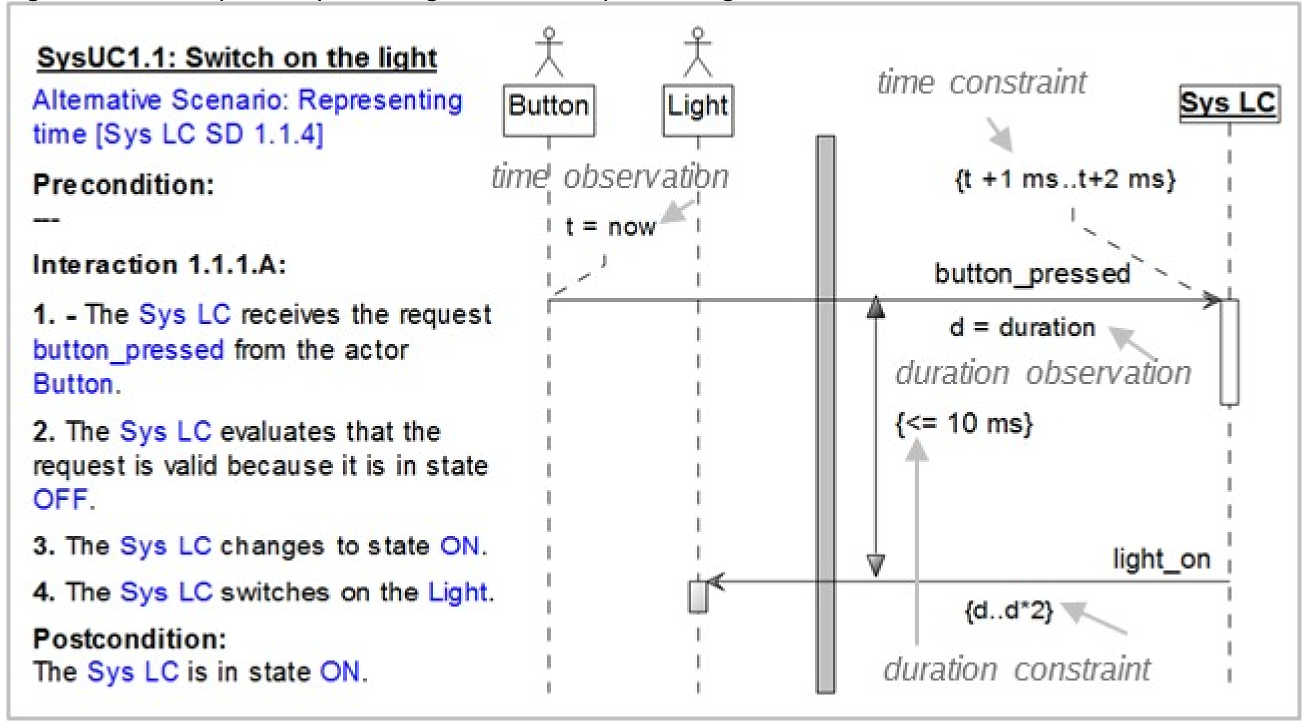
ID	Requirement
Eu.ModSt.6976	Figure 6976 Variant 2: Use case scenario without frame 
Eu.ModSt.6977	**Variant 2: Diagram heading** ***<Scenario type>:<><Scenario name><> [<Abbr. System ID><>SD<> <DiaNo of UCD>.<UCNo>.<DiaNo of SD>]***
Eu.ModSt.6978	**Examples:** Main Success Scenario: Indicate signal aspect [SubS LS SD 2.1.1] Alternative Scenario: Illuminant failure [SubS LS SD 2.1.2]
Eu.ModSt.5269	**Variant 2: Diagram heading (generic UseCase Scenario)** ***<Scenario type>:<><Scenario name><> [<Gen UC type><>SD<> <DiaNo of UCD>.<UCNo>.<DiaNo of SD>]***
Eu.ModSt.3562	**Example:** Main Success Scenario: Establish PDI connection [Gen SD 1.2.1] Main Success Scenario: Establish PDI connection [AdjS SD 1.2.1]
Eu.ModSt.765	**<Scenario type>** := "Main Success Scenario" | "Alternative Scenario" where the Main Success Scenario specifies the service to be provided when nothing goes wrong, and the Alternative Scenario describes deviations from the Main Success Scenario.
Eu.ModSt.1211	**<Scenario name>** := Unique designation of the scenario
Eu.ModSt.1210	**<DiaNo of SD> :=** Number of sequence diagram (Natural number starting with 1).
Eu.ModSt.1220	**<Interaction heading**> := Interaction <Name of interaction>:
Eu.ModSt.791	**<Name of Interaction> := <DiaNo of UCD>.<UCNo>.<DiaNo of SD>.<IId>**
Eu.ModSt.792	**<IId> :=** Id of an Interaction (Capital letters starting with "A"; if there are more than one Interactions on a scenario, the letter rises along the alphabet)

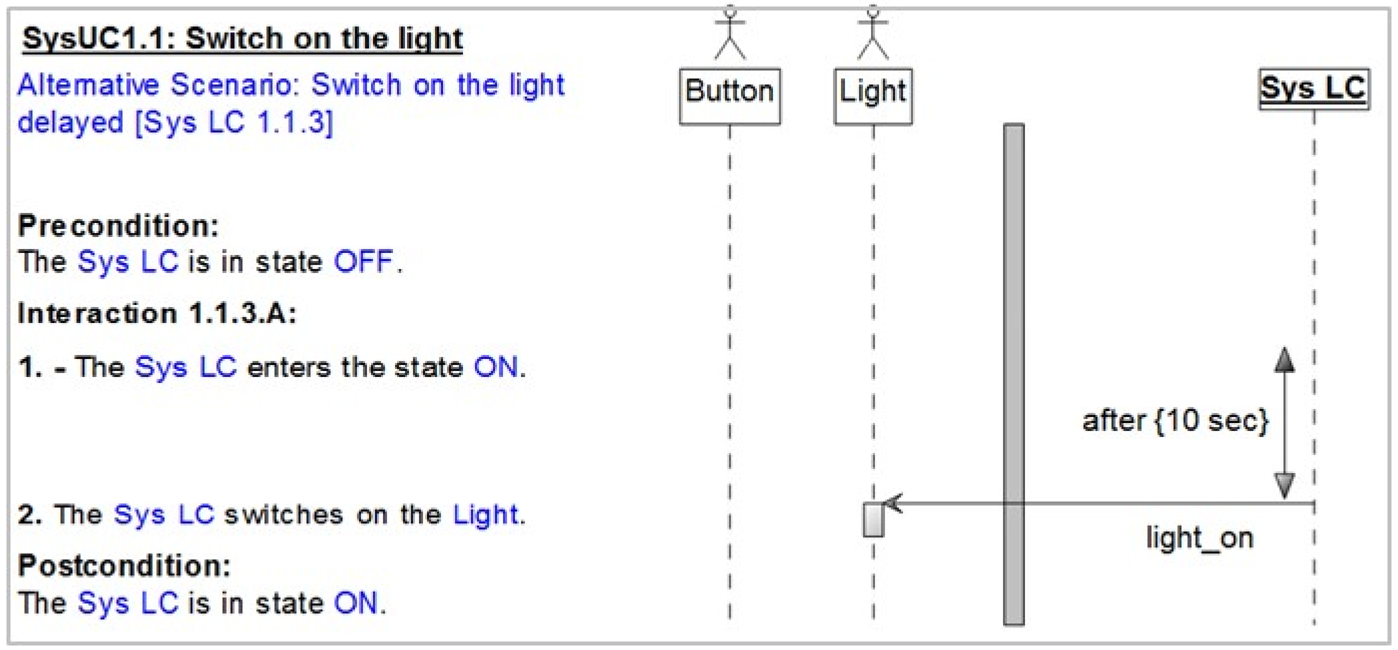
ID	Requirement
Eu.ModSt.793	**Example:** Interaction 2.1.1.A: 1. - ... 2. ... Interaction 2.1.1.B: 3. - ... 4. ...
Eu.ModSt.772	**8.3.4.2 SysML model elements**
Eu.ModSt.762	The model elements used to describe the model view "Use case scenario" and the structural principle are depicted in *Figure 763*.
Eu.ModSt.763	Figure 763 Model elements and structural principle of a use case scenario 
Eu.ModSt.773	As depicted in Fig. 763, a sequence diagram describing a UseCase scenario consists of the following vertical segments: **- Description area,** **- Lifelines of actors,** **- System boundary,** **- Lifeline of the system.**
Eu.ModSt.927	**Description area:** In the vertical segment "Description area" the action steps of the scenario are to be described.
Eu.ModSt.1278	**Lifelines:** The principal structural feature a of a scenario is the lifeline. A lifeline represents the relevant lifetime of a property of the scenario's owning block, which will be either a SysMl part or a SysML reference property. A part can be typed by an actor, which enables actors to participate in scenarios as well.

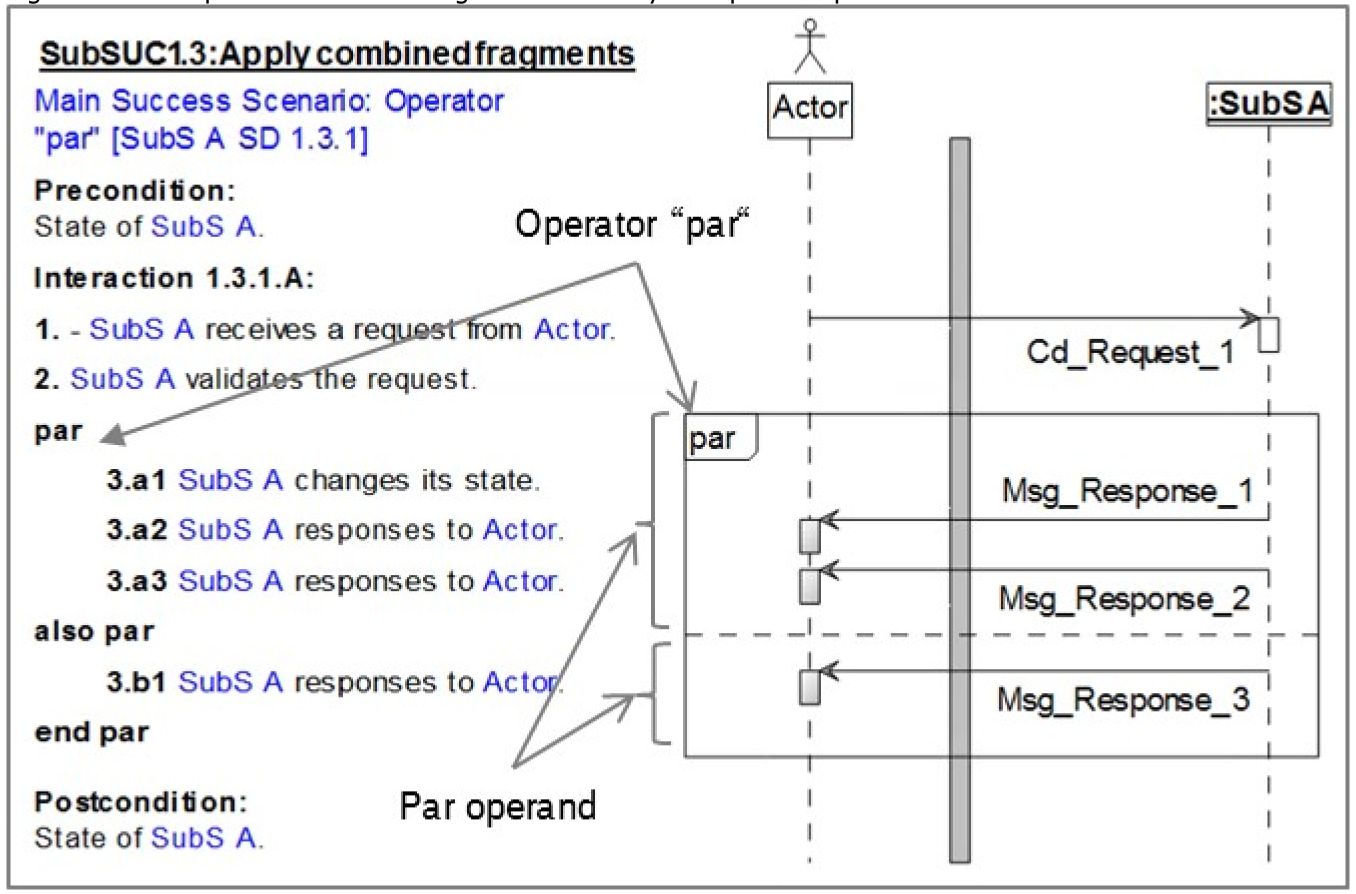
ID	Requirement
Eu.ModSt.928	**Lifelines of actors:** In the vertical segment Lifelines of actors, the actors of the system are to be arranged. This section may be empty.
Eu.ModSt.774	**Lifeline of the system:** The vertical segment Lifeline of the system is represented by an instance of the block describing the structure of the system such as "Subsystem Light Signal".
Eu.ModSt.775	**Please note:** The instance of the block has to be created once and used in all corresponding sequence diagrams.
Eu.ModSt.776	**Architectural boundary:** The architectural boundary (dashed vertical line depicted as default at any sequence diagram) is to be arranged to the right of the vertical segment "System" and overlaid by a white-coloured note.
Eu.ModSt.777	A Use case scenario of a primary Use Case is to be structured horizontally as depicted in Fig. 763.
Eu.ModSt.778	**Precondition:** After the declaration of the diagram heading, the preconditions are to be stated.
Eu.ModSt.1705	**General rules for pre- and postconditions:** Pre-and postconditions are to be defined in the following order: 1. States (if defined) of objects involved in the sequence, 2. States of timers (e.g. The Subsystem – Point monitors the Timevalue “Con_tmax_Point_Operation”) involved in the sequence, 3. All other conditions of objects, which are required before proceeding the sequence (in case of preconditions) or which are achieved after completing the sequence.
Eu.ModSt.1706	When objects are named in pre-or postconditions, the following order is to be followed: 1. Itinerary 2. Train Unit / Infrastructure Element 3. Vehicle
Eu.ModSt.1707	When nested states of objects (refer to ABB.4.250) are named in pre-or postconditions, all nested and parent states are to be named.
Eu.ModSt.1708	With the aforementioned rules, the pre-and postconditions are to be structured as follows: **<Pre/Post>conditions** <Object 1 is in state 1>. ... <Object 1 is in state n>. ... <Object 2 is in state 1>. ... <Object 2 is in state n>. ... <Object m is in state n>. ... <Conditions 1>. ... <Conditions n>.
Eu.ModSt.779	Preconditions denote what must be true before the UseCase runs. The preconditions are stated at this place if they are expected to be known by the initiator of the stimulus of the first interaction of the UseCase.
Eu.ModSt.780	The preconditions are to be structured as follows: **Precondition:** ***<Precondition 1>.*** ***...*** ***<Precondition n>.***
Eu.ModSt.782	If there are no preconditions to be stated, three hyphens are to be depicted instead of them: **Precondition:** **---**
Eu.ModSt.786	There may be cases when a precondition is not expected to be known by the initiator of the stimulus. In those cases, the precondition is to be described as validation condition at action step 2 within the first interaction according to the action block schema (see *chapter 8.1.2.1.2*).
Eu.ModSt.787	If stated at this place, alternative scenarios may be derived from that precondition.

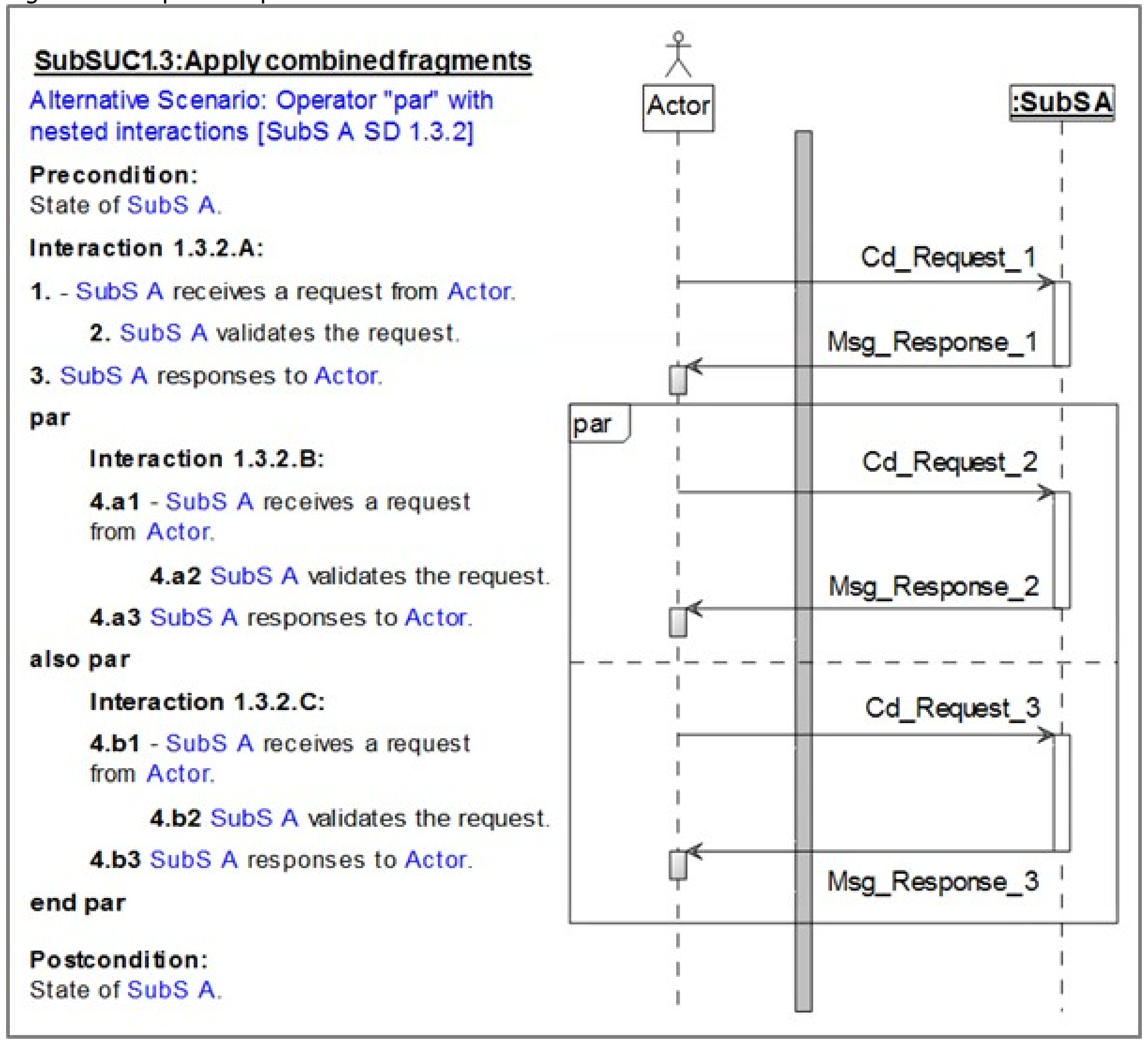
ID	Requirement
Eu.ModSt.789	The preconditions are followed by the occurrence specifications. A lifeline is related to an ordered list of occurrence specifications that describe what can happen to the instance (e.g. Subsystem Light Signal) represented by the lifeline during the execution of the scenario.
Eu.ModSt.1279	Those occurrences are specified by action steps structured by one or more interactions according to the structure depicted in *Figure 763*.
Eu.ModSt.790	**Interaction:** An interaction represents a functional system requirement structured according to the action block schema as described in *chapter 8.1.2.1.2*. It is understood as an interaction contract as introduced in *chapter 8.1.2.1.3*.
Eu.ModSt.794	An interaction is to be invoked at its first action step - **by a stimulus from an actor of the system**, - **by a timed trigger**, - **by an internal trigger** (that is, an event that occurs in the system) or - **when entering or leaving a system state**.
Eu.ModSt.795	The invoking of an interaction by a stimulus from an actor of the system is to be described as an information flow from the actor in the system environment to the system as depicted in *Figure 796*.
Eu.ModSt.797	The response of the system to an actor (primary actor or secondary actor) is to be described as an information flow from the system to the actor in the system environment as depicted in *Figure 796*.
Eu.ModSt.796	Figure 796 Information flow across the system boundary 
Eu.ModSt.799	The information flows are to be defined using SysML Item Flows or SysML signal events (in the following referred to as IO Flows) .
Eu.ModSt.800	The data types of the SysML Item Flows are to be hidden on the sequence diagram unless there is a project-specific commitment.
Eu.ModSt.7941	When using SysML signal events as IO Flows, the parameter values can also be displayed. **Example:** Msg_TVPS_Occupancy_Status(Vacant, Unable to be forced to clear, Command from EIL).
Eu.ModSt.888	An IO Flow which represents a permanent information flow is only to be depicted on the diagram as demonstrated in *Figure 932* if this information flow has changed.

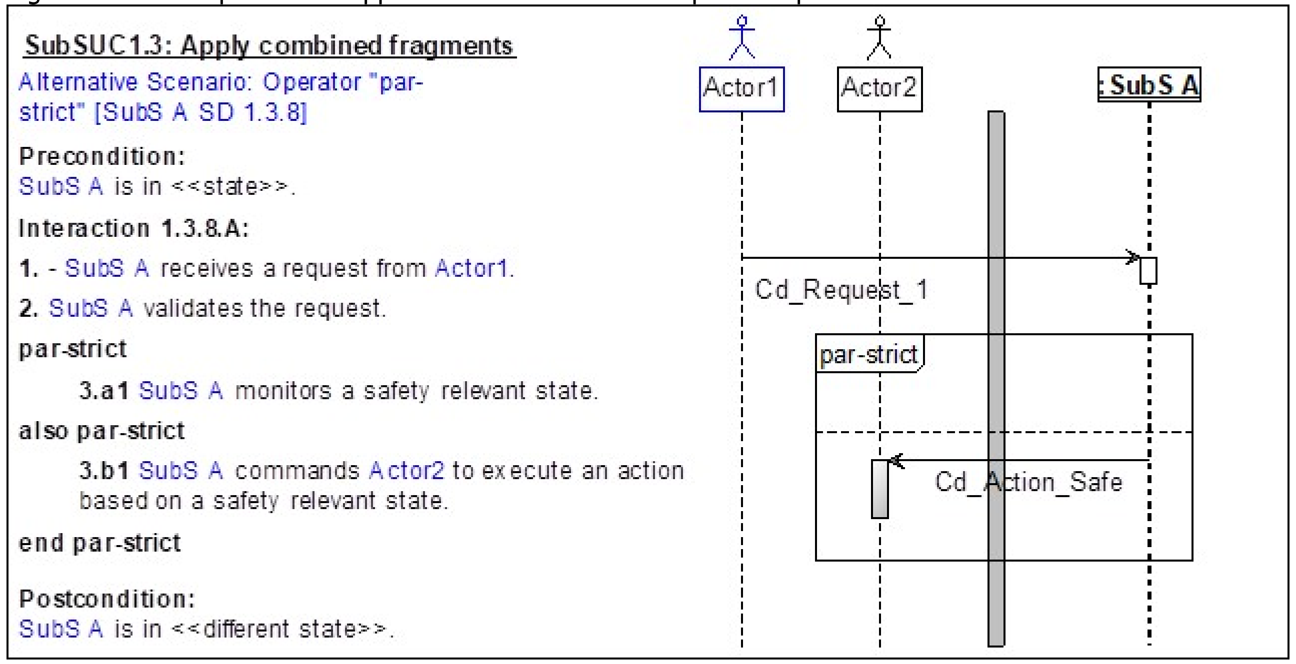
ID	Requirement
Eu.ModSt.932	Figure 932 Stimulus changes permanent information flow 
Eu.ModSt.931	In the example depicted in *Figure 930*, the stimulus "button_pressed" does not change the permanent information flow "light_on". Thus, the IO Flow "light_on" is not depicted on the diagram.
Eu.ModSt.930	Figure 930 Stimulus does not change permanent information flow 
Eu.ModSt.1267	**Representing time on a sequence diagram:** In a sequence diagram, time progresses vertically down the diagram and occurrences on a lifeline are correspondingly ordered in time. In addition, the send occurrence and receive occurrence for a single message are also ordered in time.
Eu.ModSt.1274	**Time observation and duration observation:** In addition to relative ordering in time, time can be represented explicitly on sequence diagrams. A time observation refers to an instant in time corresponding to the occurrence of some event during the execution of the scenario, and a duration observation refers to the time taken between two instants during the execution of the scenario.
Eu.ModSt.1268	**Time constraint and duration constraint:** A time constraint and a duration constraint can use observations to express constraints involving the values of those observations. A time constraint identifies a constraint that applies to a single occurrence on the sequence diagram. A duration constraint identifies two occurrences, called start and end occurrences, and expresses a constraint on the duration between them. A duration constraint can apply to any element deemed to have duration, such as a message or an execution, in which case the constraint applies between the occurrences that bracket the element's duration.
Eu.ModSt.1269	A time constraint is shown using a standard constraint expression in braces attached by a dashed line to the constrained occurrence.
Eu.ModSt.1270	A duration constraint is shown by a double-headed arrow between the two constrained occurrences with the constraint floating near it, also expressed in standard constraint notation (i.e. in braces). A duration constraint may also be shown as a standard constraint floating close to an element such as a message.

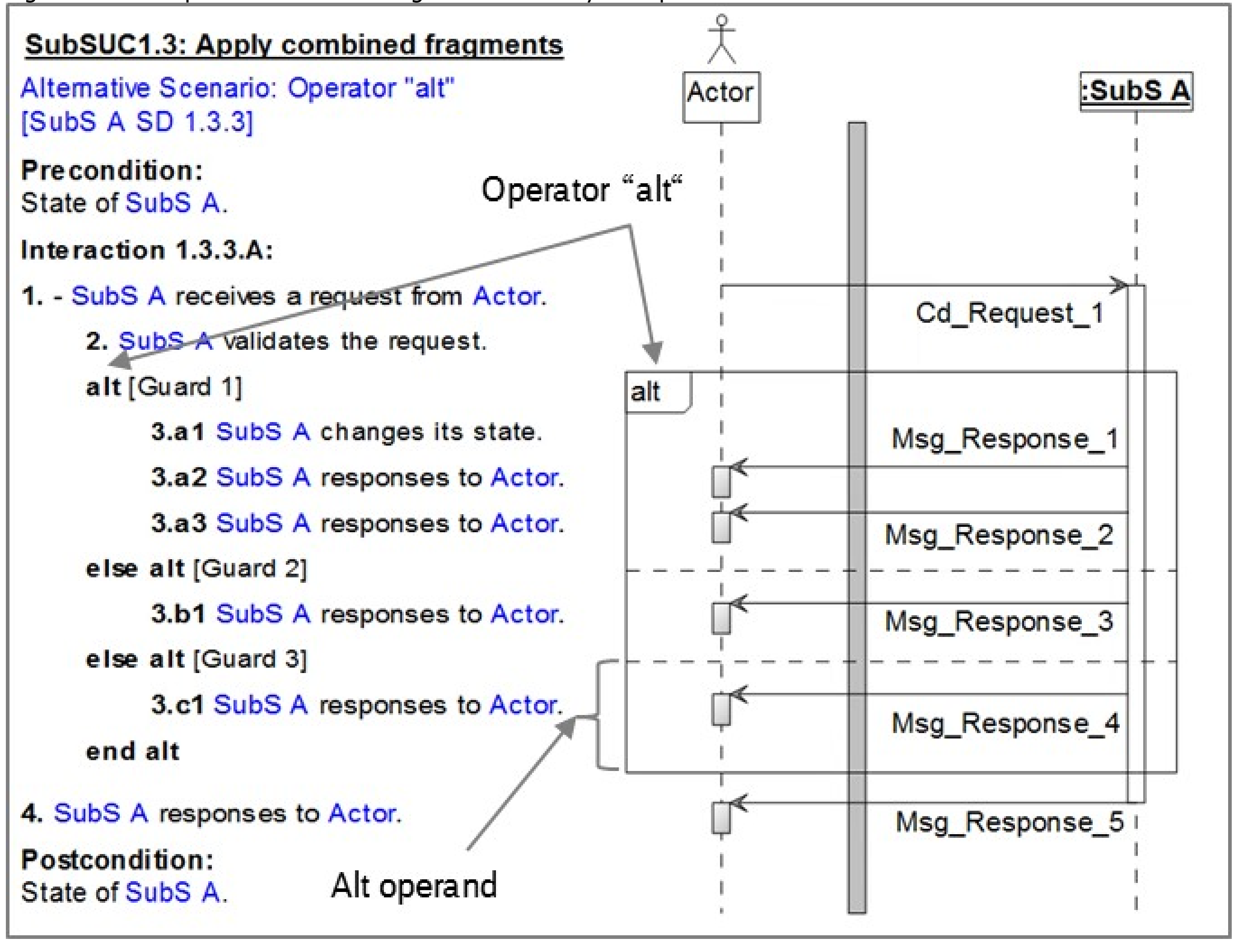
ID	Requirement
Eu.ModSt.1277	Observations are shown in a way similar to constraints, but instead of an expression in braces, an observation has the name of the observation followed by an equal sign and then an expression indicating how the value for the observation is obtained.
Eu.ModSt.1275	An example of representing time on a sequence diagram is shown in the scenario depicted in *Figure 1272*. A time observation, t, is taken at the point when the button is pressed using the expression "t = now". The time constraint {t + 1 ms..t + 2 ms} indicates that the message receipt must occur between 1 ms and 2 ms after t. The total time taken between pressing the button and switching on the light should be not more than 10 ms, as indicated by the duration constraint between action step 1 and action step 4. The duration between pressing the button and receiving the corresponding message is observed via a duration observation d, and there is a constraint ({d..d*2}) on the response "light_on" to not exceed 2 times the duration d.
Eu.ModSt.7940	**Please note:** always use "<=" instead of "<".
Eu.ModSt.1272	Figure 1272 Example of representing time on a sequence diagram 
Eu.ModSt.804	**Timed trigger (timer):** A timed trigger indicates that a given time interval has passed since the occurrence of some event, such as entering a state (internal trigger) or receiving a request during the execution of the scenario.
Eu.ModSt.1221	The term "after**"** followed by the time such as "after {10 sec}", or "after{t_con_t_max}" indicates that the time is relative to the moment of an occurrence.
Eu.ModSt.1276	An example of a timed trigger is shown in the scenario depicted in *Figure 805*. The system responses with "light_on" 10 sec after the state ON has been entered.

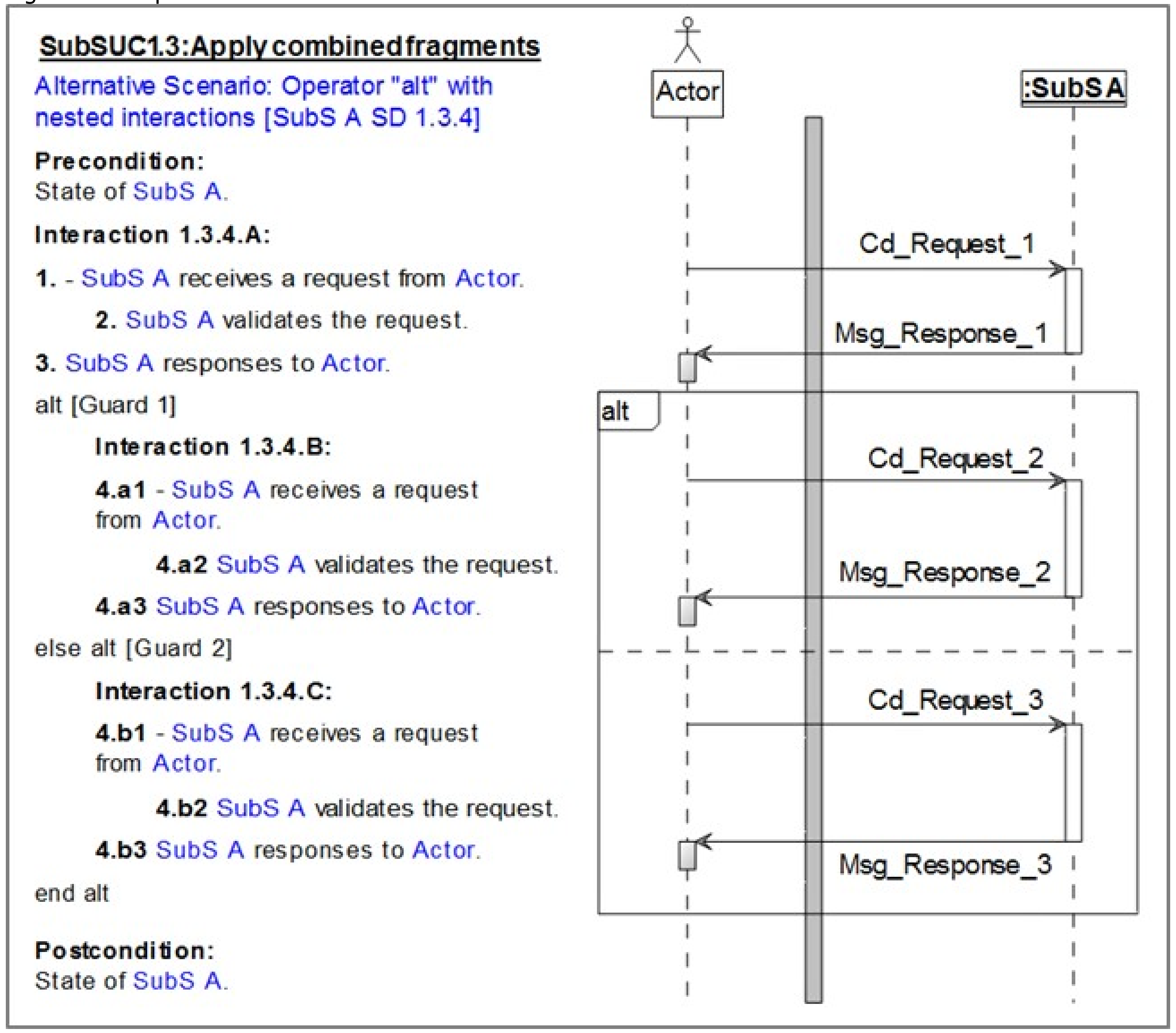
ID	Requirement
Eu.ModSt.805	Figure 805 Example of a timed trigger 
Eu.ModSt.806	**Internal trigger:** An internal trigger is described as demonstrated in the following example: 1. -The SubS LS detects a change of the indicated signal aspect.
Eu.ModSt.807	A stimulus created by entering or leaving a system state is to be described as demonstrated in the following examples. 1. - SubS LS enters the state OPERATING. 1. - SubS LS exits the state OPERATING.
Eu.ModSt.7939	The graphical representation of the time behaviour as shown in figure 1272 and figure 805 can be supplemented by a description in the description area of the sequences. "t_con_t_max" represents the defined time period (duration): • Start of timer should be mentioned within the corresponding step (trigger). • "Subsystem X starts to monitor the time period "t_con_t_max"." • Reaction for timer that shall be waited for --> where possible combine within corresponding step otherwise keep it separate. • "Subsystem X detects that time period "t_con_t_max" has expired." • Reaction for timer that has been exceeded (unintended case) --> where possible combine within corresponding step otherwise keep it separate. • "Subsystem X detects that time period "t_con_t_max" has exceeded." • Restart of a timer within the corresponding step (trigger). • "Subsystem X stops to monitor time period "t_con_t_max" caused by first command and starts to monitor the time period ""t_con_t_max" caused by second command." • Reset of a timer within the corresponding step (trigger). • "Subsystem X stops to monitor time period "t_con_t_max"."
Eu.ModSt.7943	**Time periods** shall be defined using block properties without further specification of the values. The values to be used shall be specified separately in the requirements management tool (chapter 5.3 Configuration and engineering data) as binding requirements and linked to the corresponding definitions.
Eu.ModSt.808	**Combined fragments:** In order to parallelize interactions as well as action steps of an interaction or define alternatives or loops, combined fragments defined by the Operators **"par"**, **"alt"** or **"loop"** may be used.
Eu.ModSt.809	In sequence diagrams, combined fragments are logical groupings, represented by a rectangle, which contain the conditional structures that affect the flow of messages. A combined fragment contains operands and is defined by operators (see *Figure 812* and *Figure 935*).
Eu.ModSt.855	Operands are separated by dashed lines.
Eu.ModSt.856	Depending on the operator, there is a guard containing a constraint expression that indicates the conditions under which it is valid for the operand to begin execution. Guards appear at the beginning of the combined fragment following the corresponding operator (example: alt [Guard]).
Eu.ModSt.810	The operator identifies the type of logic or conditional statement that defines the behaviour of the combined fragment.

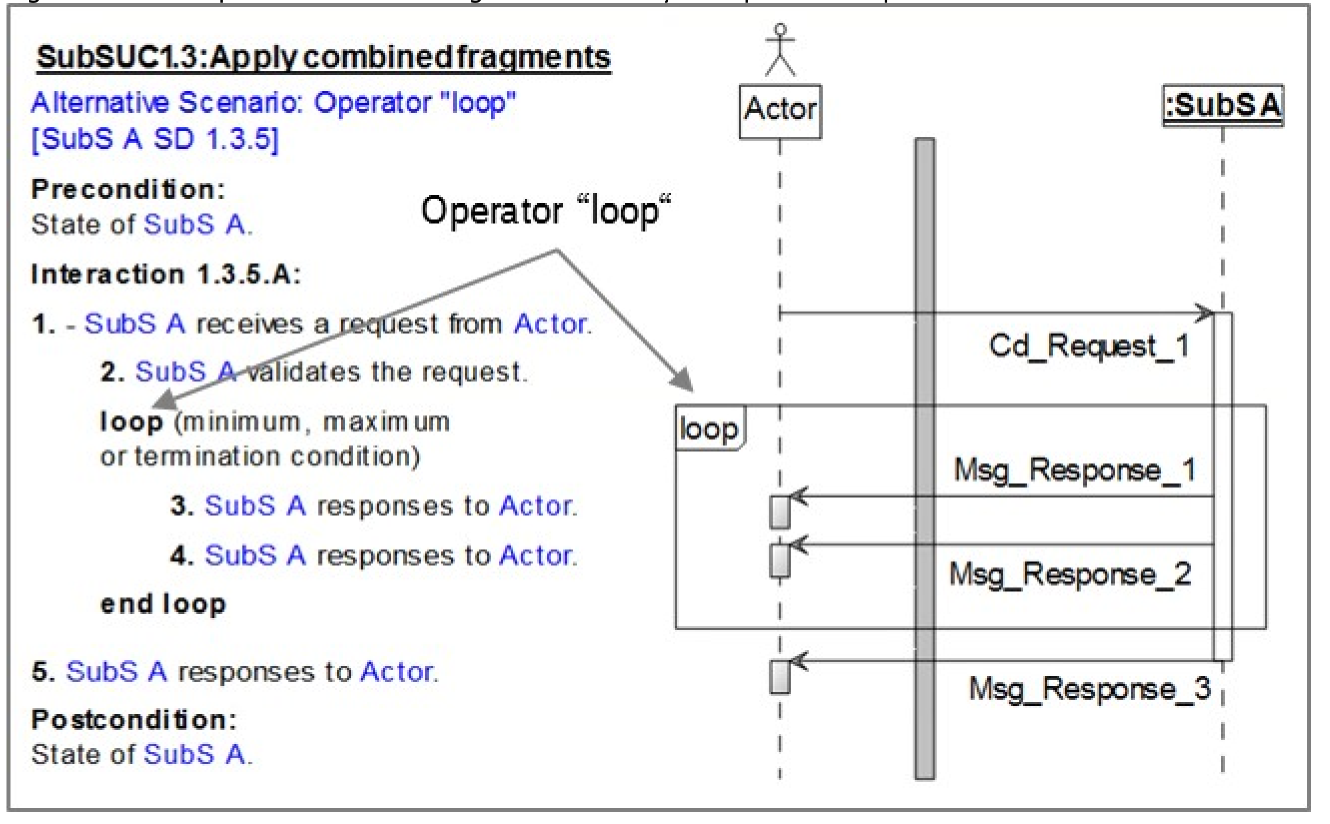
ID	Requirement
Eu.ModSt.811	**Operator "par"** In the example depicted in *Figure 812*, the usage of the operator "par" is demonstrated. The message *Msg_Response_3* is parallelized to *Msg_Response_1* followed by *Msg_Response_2* using two par operands.
Eu.ModSt.857	If a par operand consists of more than one action step, the action steps are structured according the following schema (see also *Figure 812*)**:** **par** 3.a1 action step. 3.a2 action step. 3.ax ... **also par** 3.b1 action step. 3.b2 action step. 3.bx ... **also par** 3.c1 action step. 3.cx... **end par**
Eu.ModSt.812	Figure 812 Example of a combined fragment defined by the operator "par" 

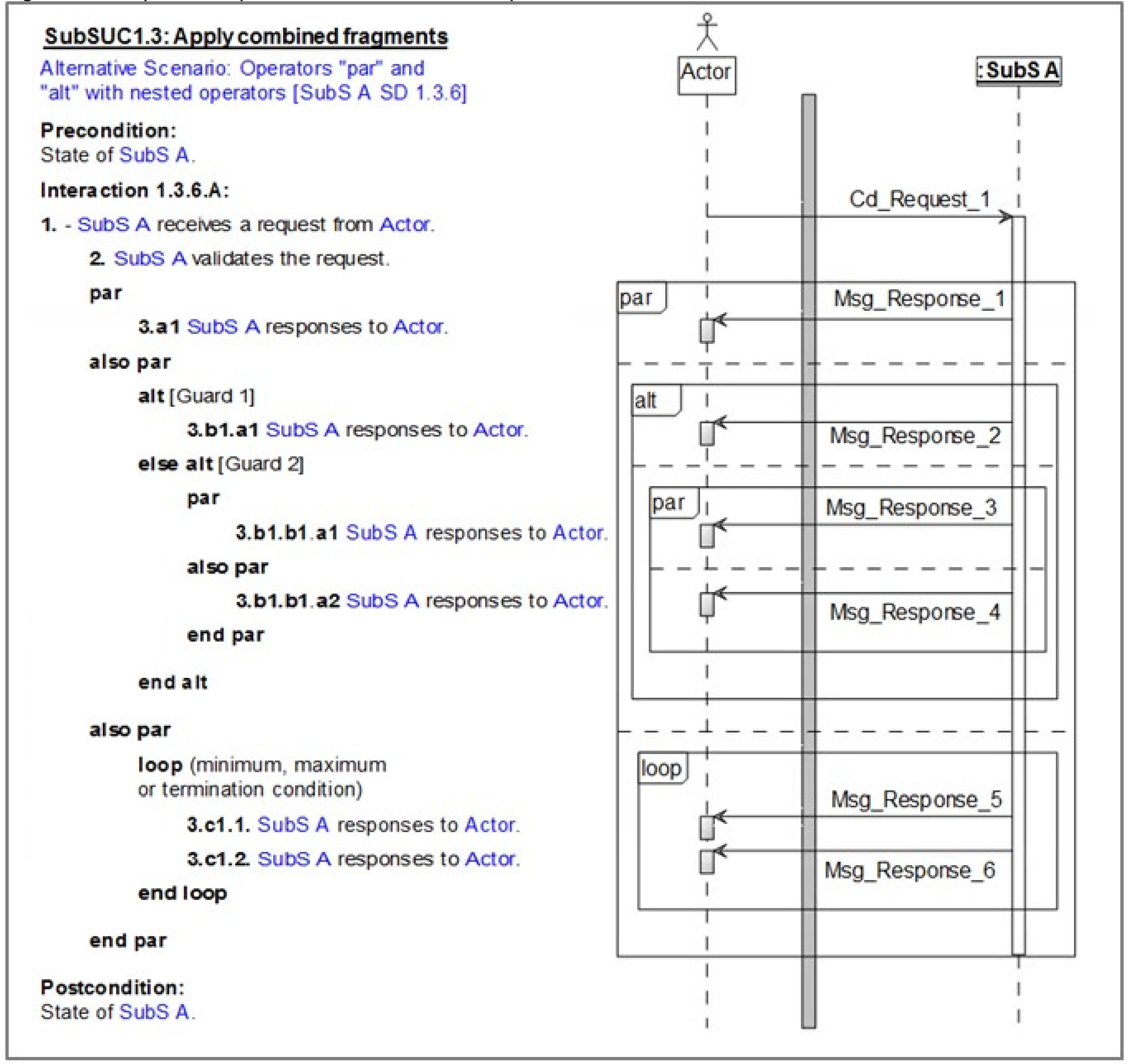
ID	Requirement
Eu.ModSt.813	Interactions are to be parallelized according to the following schema **(**see also Fig. 1255**):** **par** Interaction <Name of the interaction> 4.a1 - action step. 4.a2 action step. Interaction <Name of the interaction> 4.a3 - action step. 4.a4 action step. 4.ax ... **also par** Interaction <Name of the interaction> 4.b1 - action step. 4.b2 action step. 4.bx ... **end par**
Eu.ModSt.1255	Figure 1255 Operator "par" with nested interactions 
Eu.ModSt.1700	**Operator "par-strict"** The keyword "strict" is defined as extension to the operator "par": • Semantics: If the "par" operator of a combined fragment is extended by the keyword "strict", all operands must be executed strictly parallel. This means that IOFlows are sent at the exact same time and included or extended UseCases are invoked at the same time and terminated at the same time. • Syntax: Extend keyword "par" in sequence text as well as in graphical frame box by "-strict"
Eu.ModSt.1701	In the example in Fig.1702, the usage of the extension "strict" of the operator "par" is shown.

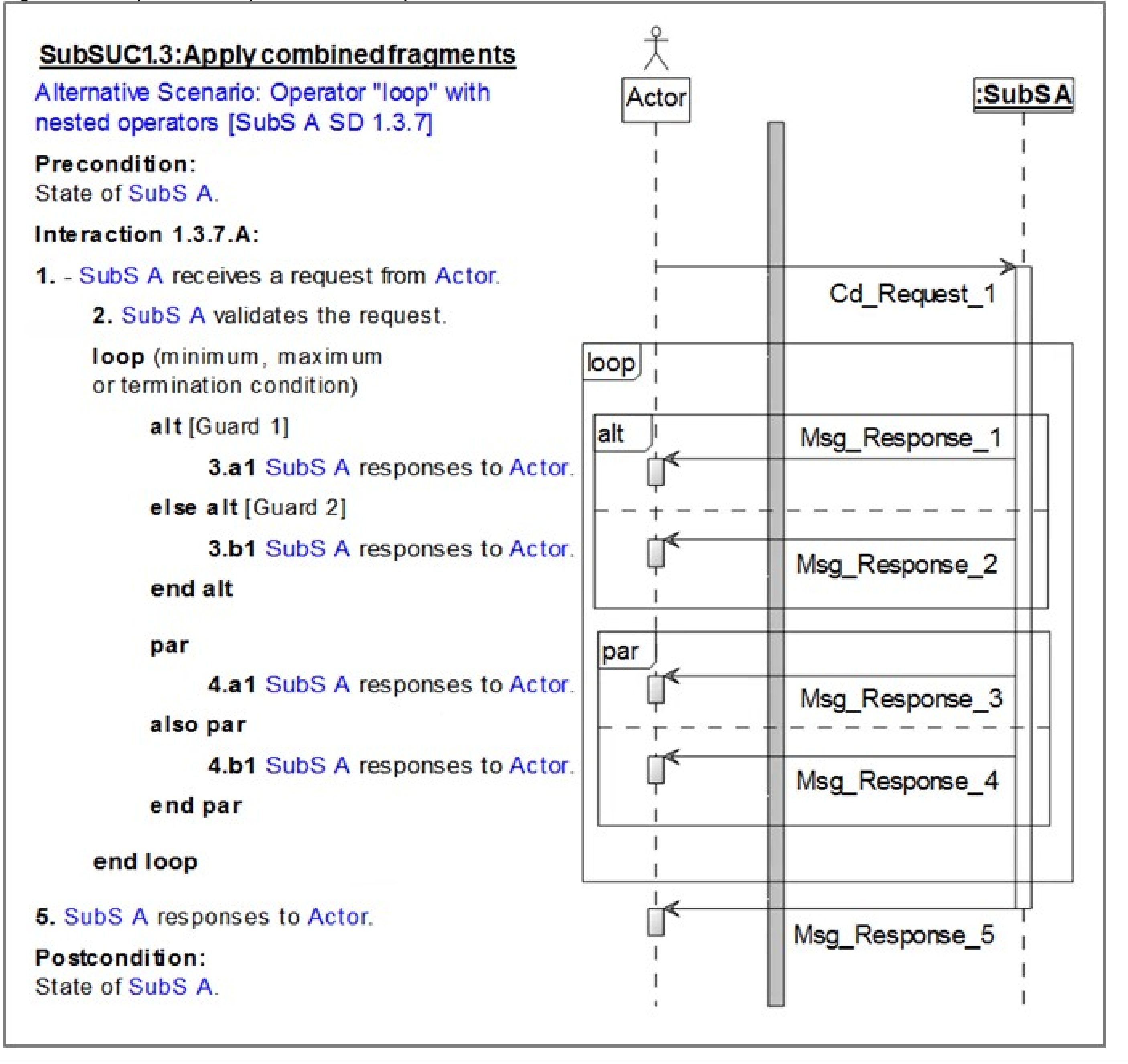
ID	Requirement
Eu.ModSt.1702	Figure 1702 Example for the application of the extended operator "par-strict" 
Eu.ModSt.1703	If a "par-strict" operand consists of more then one action step, the action steps are structured according the following schema: **par-strict** 3.a1 action step. 3.a2 action step. 3.ax ... **also par-strict** 3.b1 action step. 3.b2 action step. 3.bx ... **also par-strict** 3.c1 action step. 3.cx... **end par-strict**
Eu.ModSt.936	**Operator "alt"** In the example depicted in *Figure 935*, the utilisation of the operator "alt" is demonstrated in the way that exactly one of its operands is selected based on the value of its guard. The guard on each operand is evaluated before selection, and if the guard on one of the operands is valid, that one is selected. If more than one operand has a valid guard, the selection is nondeterministic. An optional else fragment (else fragment without guard) is valid only if none of the guards on the other operands are valid.
Eu.ModSt.1704	In case no guard of an alt operand is valid then no operand is executed, unless an optional else fragment without a guard is defined, in which case that operand is selected.
Eu.ModSt.814	If an alt operand consists of more then one action step, the action steps are structured according the following schema (see also *Figure 935*)**:** **alt** [Guard 1] 3.a1 action step. 3.a2 action step. 3.ax... **else alt** [Guard 2] 3.b1 action step. 3.b2 action step. 3.bx... **end alt**

ID	Requirement
Eu.ModSt.935	Figure 935 Example of a combined fragment defined by the operator "alt" 
Eu.ModSt.937	Interactions are to be used in alt operands according to the following schema (see also Figure 1256)**:** **alt** [Guard 1] Interaction <Name of the interaction> 4.a1 - action step. 4.a2 action step. Interaction <Name of the interaction> 4.a3 - action step. 4.a4 action step. 4.ax ... **else alt** [Guard 2] Interaction <Name of the interaction> 4.b1 - action step. 4.b2 action step. 4.bx ... **end alt**

ID	Requirement
Eu.ModSt.1256	Figure 1256 Operator "alt" with nested interactions 
Eu.ModSt.854	Please note: the guards of the alt operands are not to be depicted inside the combined fragment but only in the textual description of it.
Eu.ModSt.1983	**Operator "opt":** The operator **"opt"** (optional sequence) is equivalent to the operator **"alt"** with only one **operand**. This implies that the operand is either executed or skipped depending on the validity of the **guard** (condition).
Eu.ModSt.858	**Operator "loop"**: A loop is specified by the interaction operator "loop" in which the trace represented by its operand repeats until its termination constraint is met. It may define lower and upper bounds on the number of iterations as well as the guard expression. As shown in *Figure 1257*, these bounds are documented in brackets after the loop keyword as (minimum, maximum or termination condition), where the maximum (upper bound) may have the value * indicating an unlimited upper bound.
Eu.ModSt.859	A combined fragment describing a loop is to be structured according to the following schema (see also *Figure 1257*)**:** **loop** (minimum, maximum or termination condition) 1. action step. 2. action step. 3. action step. 4. ...

ID	Requirement
Eu.ModSt.1257	Figure 1257 Example of a combined fragment defined by the operator "loop" 
Eu.ModSt.860	Note: the (minimum, maximum or termination condition) of the loop operand is not to be depicted inside the combined fragment but only in the textual description of it.
Eu.ModSt.1261	As shown in *Figure 1258* and *Figure 1259* the operands of combined fragments may themselves contain combined fragments, and thus can be composed into a tree hierarchy.

ID	Requirement
Eu.ModSt.1258	Figure 1258 Operators "par" and "alt" with nested operators 

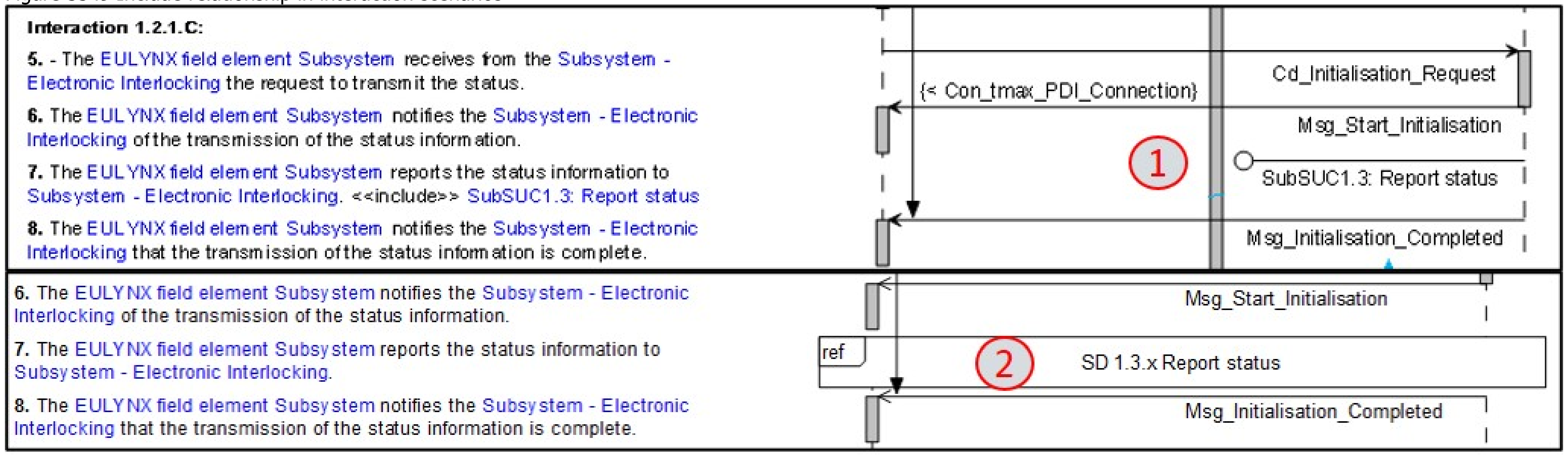
ID	Requirement
Eu.ModSt.1259	Figure 1259 Operator "loop" with nested operators 

ID	Requirement

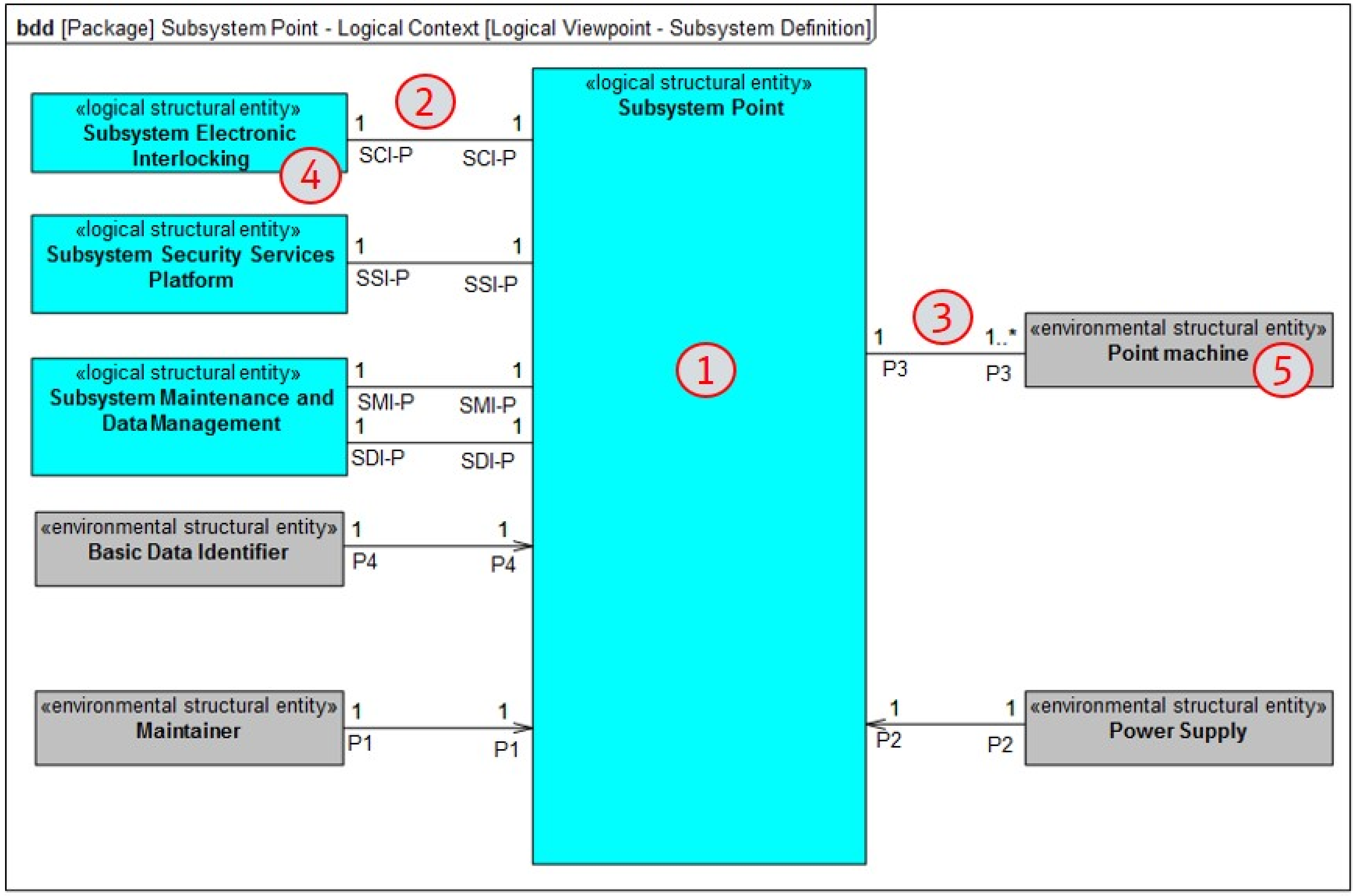
Eu.ModSt.7961	**Multiple instances of the same type**
	The behaviour of the SUS may interact with multiple instances of the same type of a subsystem, adjacent system or actor. One example is the subsystem point interacting with multiple point machines. In such cases, often either:
	1**) all instances** (or no instance) of a group of instances of the same type must fulfil a certain condition in order to induce a certain behaviour of the SUS
	2) **at least one instance** of a group of instances of the same type must (or must not) fulfil a certain condition in order to induce a certain behaviour of the SUS
	
	Multiple instances of the same type of a subsystem, adjacent system or actor, are modelled as described below:
	• Exactly two instances of the subsystem, adjacent system or actor are shown in the sequence diagram, labelled as '1st' and 'n-th'. In case of additional multiple sets, the 'n' is replaced by other available characters.
	• The two instances jointly represent a set of instances with multiplicity 2 or higher. All represented interactions for the two instances must be interpreted together. The instance represented as '1-st' does not represent any specific instance within the set.
	• In the interaction that refers to a condition of the instances, use a combined fragment as follows:
	    • If an "all instances" (or "no instances") condition shall be represented, use a "par" fragment with two legs testing the condition on the "1st" and on the "n-th" instance. The interactions with all instances of the set are to be interpreted as "AND"-connected.
	    • If an "at least one instance" condition shall be represented, use an "alt" fragment with two legs testing the condition on the "1st" and on the "n-th" instance. The interactions with all instances of the set are to be interpreted as an "OR"-connected.

Eu.ModSt.815	**Postcondition:** The postconditions positioned after the last interaction of a scenario representing the results of a UseCase are to be structured as follows: **Postcondition:** ***<Postcondition 1>.*** ***...*** ***<Postcondition n>***.
Eu.ModSt.816	**Example** (see Fig. 715): **Postcondition:** SubS LS indicates the commanded signal aspect.
Eu.ModSt.1222	Postconditions which equal preconditions are not to be stated.
Eu.ModSt.938	If there are no postconditions to be stated, three hyphens are to be depicted instead of them: **Postcondition:** **---**
Eu.ModSt.3547	**Include relationship** As shown in *Figure 3549* an <<include>> relationship can be used to jump from an interaction scenario to the interaction scenario of an included use case (e.g., SubSUC1.3: Report status). The text part and the include symbol **(1)** indicate which use case is to be accessed. After processing the included interaction scenario, the original interaction scenario is continued.
Eu.ModSt.3548	Alternatively to the include symbol **(1)** an "interaction use" **(2)** may be used to indicate which included interaction scenario is to be accessed. "Interaction uses" are shown as frames with the keyword "ref" in the frame label. The body of the frame contains the name of the referenced interaction scenario.
Eu.ModSt.7949	For each SD that is referenced in another SD, a notice must be inserted in the modelling tool (e.g. Properties ->Text->Description) that corresponds to a defined schema: • This SD is part of [referred SD].
Eu.ModSt.7950	The notice is to be transferred to "Requirements Part 2" of the specification document generated in the requirements management tool.

ID	Requirement
Eu.ModSt.3549	Figure 3549 Include relationship in interaction scenarios



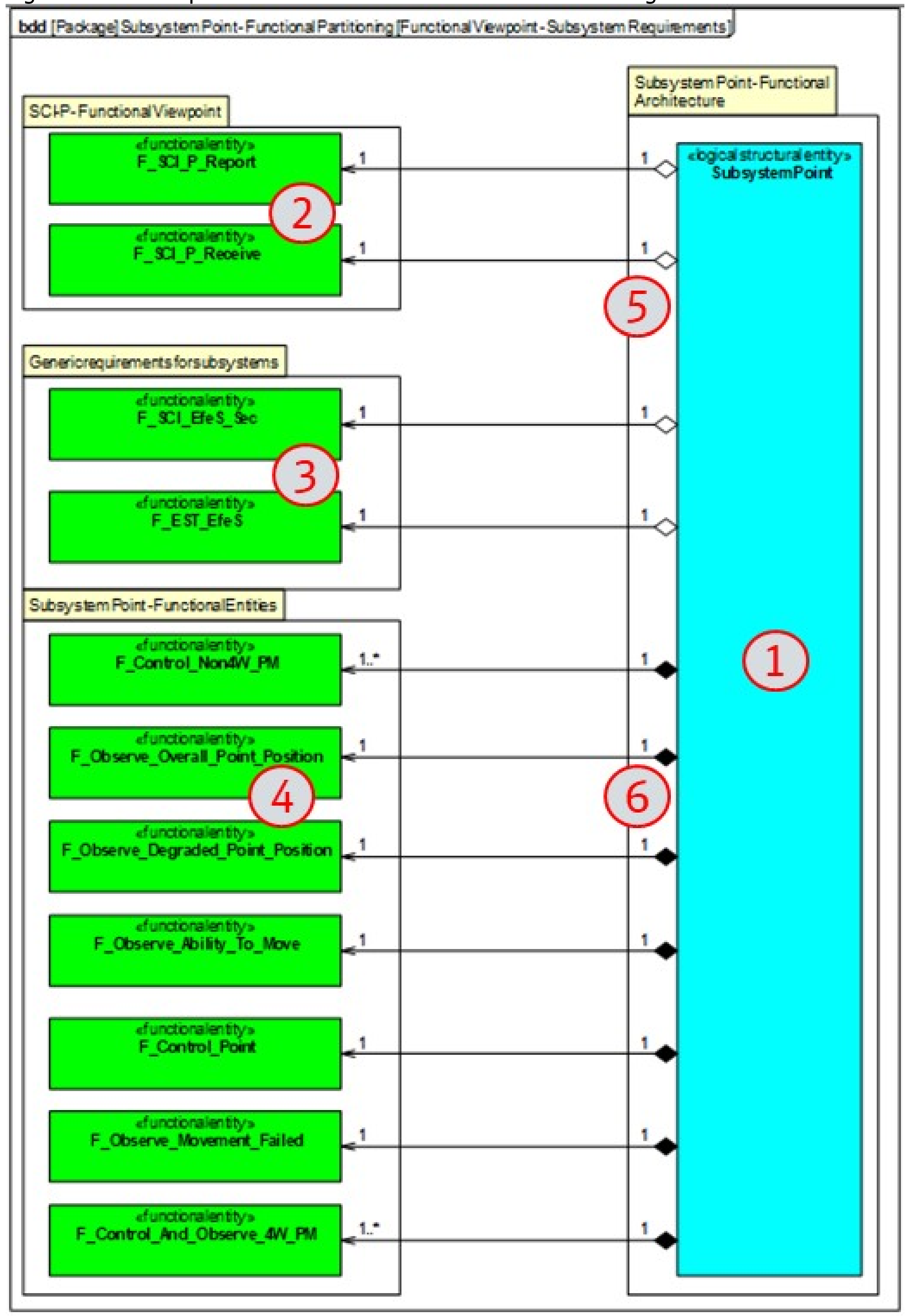
Eu.ModSt.7084	**8.3.4.3 Binding** (see *chapter 8.2.1*)
Eu.ModSt.7753	**Diagram of model view "Use case scenario"** has an "**Info**" binding if it is further specified in a refined model view (e.g. through a state machine).
Eu.ModSt.7938	**Diagram of model view "Use case scenario"** has a "**Req**" binding if it is not further specified in a refined model view.
Eu.ModSt.7942	The definitions of **time periodes** (e.g. Con_tmax_PDI_Connection) represented by block properties have "**Def**" bindings.
Eu.ModSt.7944	The **values of the defined time periods**, which are specified and linked separately in the requirements management tool, have "**Req**" bindings.
Eu.ModSt.2131	**8.3.5 Model View "Logical Context" of a SUS (AL1) - Description**
Eu.ModSt.2132	The model view "Logical Context" as shown in *Figure 2134* represents the environment of the SUS and provides initial information about the SUS boundaries and the relationships to the interaction partners. This diagram contains the following definitions relevant to implementation: • **Interaction partners:** the representation of the interaction partners as actors with whom the SUS concerned must be able to interact, • **Logical SUS interfaces:** - number of required logical interfaces represented by associations to interaction partners in the SUS environment defined by means of multiplicities at the association ends - possible directions of the interaction (uni- or bidirectional). - kinds of interfaces such as SCI-P, SMI-P and so on defined by means of roles at the association ends.
Eu.ModSt.2136	**Interaction partners** Interaction partners **(4, 5)** of the SUS **(1)** are represented by actors. An actor describes a person (for example "Maintainer") or another system (for example the "Subsystem - Electronic Interlocking) in the role of a user of services offered by the SUS concerned (here "Subsystem Point"). At the logical viewpoint actors are represented by logical structural entities if they are in the context of a system element belonging to the same overall system. If an actor in the context of a system element is outside of the overall system of this system element (adjacent system) it is represented by an environmental structural entity.
Eu.ModSt.7880	*Figure 2134* therefore includes for example the following related definitions: • system element "Subsystem Electronic Interlocking" represented by a logical structural entity (LSE) assumes the role of an actor in the environment of "Subsystem Point" belonging to the same overall system **(4)**. • system element "Point machine" represented by an environmental structural entity (ESE) assumes the role of an actor in the environment of "Subsystem Light Signal" not belonging to the same overall system **(5)**.
Eu.ModSt.2139	**Logical SUS Interfaces** The connection between the SUS (represented by a logical structural entity) and an actor represents a logical interface **(2, 3)**. It is depicted as an association that is a continuous line between the actor and the SUS. It represents the definition that the SUS must be able to interact with the connected actor through a corresponding logical interfaces.
Eu.ModSt.2140	The association also represents the possible interaction directions of the interface. No arrow heads means that the interaction is bidirectional. An arrow head on the other hand indicates that an interaction is only possible in the direction of the arrow.
Eu.ModSt.2141	On the side of the actor of the association, a multiplicity indication describes in more detail with how many of the respective actors the SUS concerned must be able to interact i.e., how many logical interfaces are required.
Eu.ModSt.2142	The definition of the quantity of each actor by means of multiplicities represents an important requirement regarding system development. It is obvious that it makes a difference, for example, whether the system depicted in *Figure 2134* requires an interface to one "Subsystem Electronic Interlocking" or to several.

ID	Requirement
Eu.ModSt.2143	The multiplicity "1" is defined at the SUS side of the association. The reason for this is that only requirements for the SUS concerned may be phrased in the respective requirements specification. However, according to the SysML syntax, a multiplicity indication at the SUS side would represent a statement for the actor.
Eu.ModSt.2144	Some examples for the representation of multiplicities and their meaning: 1 or blank exactly one 0..1 none or one * none or several 1..* one or several 2..4 at least two and at most four
Eu.ModSt.7881	*Figure 2134* therefore includes for example the following related definitions: • the "Subsystem Point" must be able to interact with exactly one "Subsystem Electronic Interlocking" as an actor, with the interaction possible in two directions. • the "Subsystem Point" must be able to interact with one or more actors "Point machine", with the interaction possible in two directions. • the "Subsystem Point" must be able to interact with exactly one "Basic Data Identifier" as an actor, with an interaction only possible from "Basic Data Identifier" to the "Subsystem Point".
Eu.ModSt.7745	Roles at the association ends represent the used "**Interface kind**" such as SCI-LS, SMI-LS and so on. In Figure 2134 "Subsystem Point" sees for example "Subsystem Electronic Interlocking" in the role of "SCI-P" and vice versa.
Eu.ModSt.7882	*Figure 2134* therefore includes for example the following related definitions: • the interface between "Subsystem Point" and "Subsystem Electronic Interlocking" must be implemented according to the specification of "SCI-P". • the interface between "Subsystem Point" and "Subsystem Maintenance and Data Management" must be implemented according to the specification of "SMI-P". • the interface between "Subsystem Point" and "Subsystem Maintenance and Data Management" must be implemented according to the specification of "SDI-P". • the interface between "Subsystem Point" and "Subsystem Security Services Platform" must be implemented according to the specification of "SSI-P".
Eu.ModSt.2134	Figure 2134 Example of SUS model view "Logical Context" 

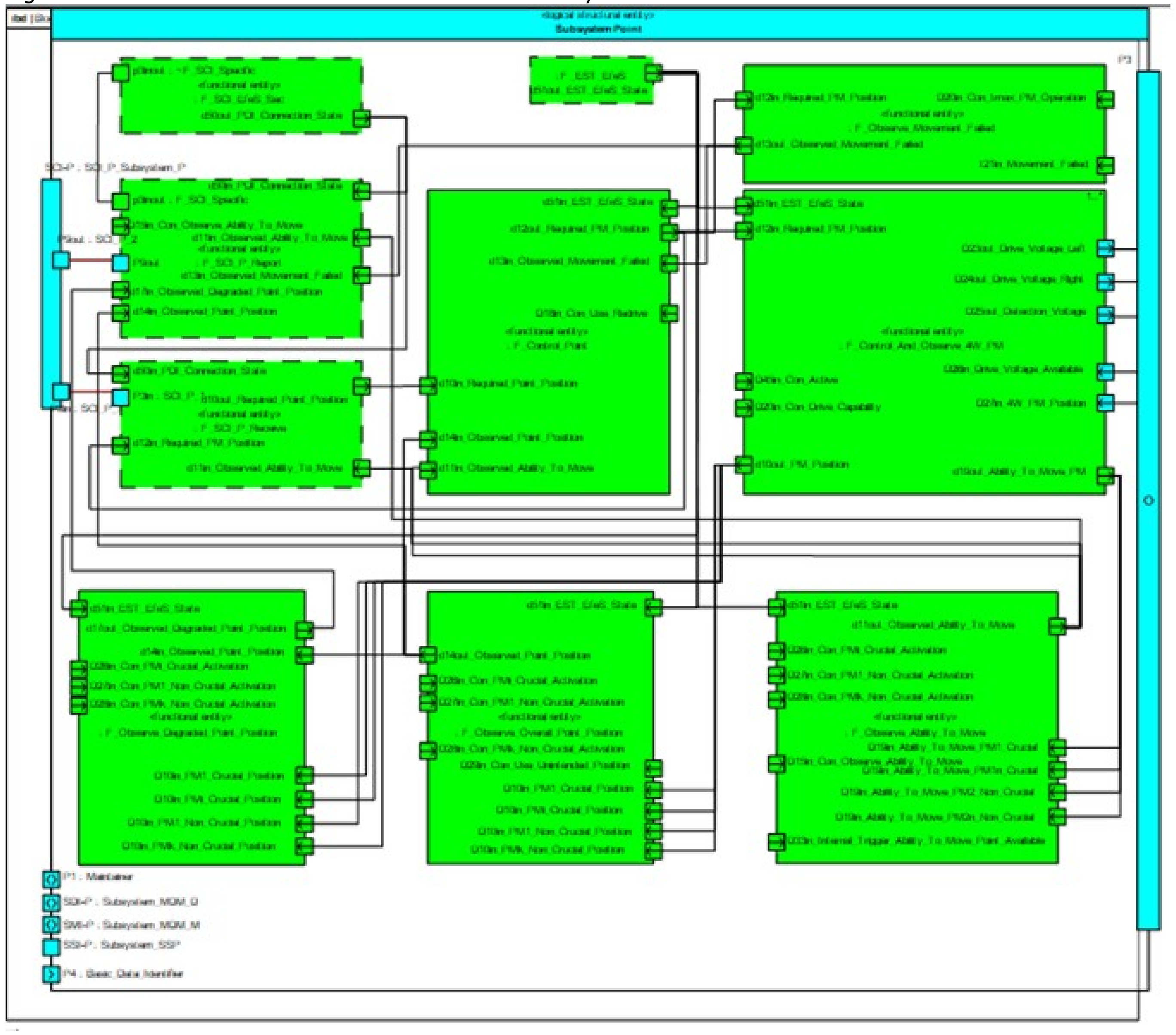
ID	Requirement
Eu.ModSt.377	**8.3.6 Model view "Logical Context" of a SUS (AL1) - Modelling rules**
Eu.ModSt.378	**8.3.6.1 SysML diagram**
Eu.ModSt.379	**Block definition diagram (BDD):** depicts the view "Logical System Context".
Eu.ModSt.3560	**Name of the Diagram:** *bdd[Package]<><System block signature><>-<>Logical Context<>[Logical Viewpoint<>-<>Subsystem Definition]*.
Eu.ModSt.383	**Example:** bdd [Package] Subsystem Light Signal - Logical Context [Logical Viewpoint - Subsystem Definition]
Eu.ModSt.385	**8.3.6.2 Model elements**
Eu.ModSt.890	The model elements basically used to describe the model view "Logical Context" are depicted in *Figure 2134*.
Eu.ModSt.386	**Block:** Modular unit of structure in SysML that is used to define the Logical Structural Entity (LSE) or Environmental Structural Entity (ESE) representing the logical view of the SUS or the actors at the uppermost level of abstraction.
Eu.ModSt.1184	**Naming conventions for blocks representing LSEs:** **<System block signature>** := <Abbr. System ID> | <System ID>
Eu.ModSt.1186	**<Abbr. System ID> :=** <Abbr. System type><><Abbr. System name>
Eu.ModSt.1212	**<Abbr. System type**> := "Sys" | "SubS" | "SysElem"
Eu.ModSt.1213	**<Abbr. System name**> := freely selectable
Eu.ModSt.1188	Examples: Sys ABB SubS LS SysElem 1
Eu.ModSt.1185	**<System ID> :=** <System type><><System name>
Eu.ModSt.1214	**<System type>** := "System" | "Subsystem" | "System Element"
Eu.ModSt.1215	**<System name**> := freely selectable
Eu.ModSt.1187	Example: System ABB Subsystem Light Signal System Element 1
Eu.ModSt.1252	If there are project-specific commitments, a deviating designation of **<System block signature>** may be used.
Eu.ModSt.1189	The modeller must ensure that the descriptions of the functional (Functional Viewpoint) and logical (Logical Viewpoint) representations of actors and SUS match.
Eu.ModSt.391	**Actor:** At the Functional Viewpoint (model view "Functional Context"), an actor may be a class of users, roles users can play, or other systems. Cockburn [22] distinguishes between primary and secondary actors.
Eu.ModSt.740	A primary actor is one having a goal requiring the assistance of the system.
Eu.ModSt.741	A secondary actor is one from which the system needs assistance.
Eu.ModSt.392	**Depiction of an actor:** At the logical viewpoint, however, the actors defined in the model view "Functional Context" are represented as parts of the logical overall system architecture. They are represented by logical structural entities if they are in the context of a system element belonging to the same overall system. If an actor in the context of a SUS is outside of the overall system of this SUS (adjacent system) it is represented by an environmental structural entity.
Eu.ModSt.394	**Association:** specifies the structural relationship between a block, i.e. the SUS and an actor. It represents a logical interface (see also *chapter 8.3.5*)
Eu.ModSt.395	Depending on the direction of the information flow, the association has to be stated bi-directional or uni-directional.
Eu.ModSt.396	At the actor's side of an association, the multiplicity that defines the required quantity of each actor and the name of the logical interface has to be stated.

ID	Requirement
Eu.ModSt.397	At the block's side of an association, the multiplicity "1" and the name of the logical interface has to be stated.
Eu.ModSt.1191	**Naming conventions for interfaces:** **<Interface kind>** := <Abbr. Type of interface>-<Interface ID>
Eu.ModSt.1192	**<Abbr. Type of interface>** := S*)CI | S*)Freely selectable | Freely selectable S*)CI: Communication interface S*)Freely selectable: Standardised Interface except SCI Freely selectable: any non-standardised interface *) "S" indicates that the interface is standardised
Eu.ModSt.1193	**<Interface ID> :=** Freely selectable designator (as far as a generic interface is concerned, "Gen" or "XX"is to be used as Interface ID)
Eu.ModSt.1194	Examples: SCI-P, SMI-LS, SDI-LS, SCI-Gen, SCI-XX
Eu.ModSt.1286	If the interface kind is used within the executable part of the model, where hyphens <-> are forbidden, an underscore <_> is to be used between <Abbr. Type of interface> and <Interface ID>.
Eu.ModSt.1287	Examples: SCI_P, SMI_LS, SDI_LS, SCI_Gen, SCI_XX
Eu.ModSt.1896	If there are project-specific commitments, a deviating designation of **<Interface kind>** may be used.
Eu.ModSt.7746	**8.3.6.3 Binding** (see *chapter 8.2.1*)
Eu.ModSt.7752	**Diagram of model view "Logical Context"** has a "**Def**" binding.
Eu.ModSt.7718	**8.3.7 Model view "Functional Partitioning" of a SUS (AL2) - Description**
Eu.ModSt.7721	The model view "Functional Partitioning" shown in *Figure 7723* describes the refinement of the SUS **(1)** by FEs.
Eu.ModSt.7849	The FEs **(2)** defined in the SIUS model view "Functional Partitioning" (see *chapter 8.4.3*), which represent the local behaviours of the PDI (see *chapter 8.2.4*), and the generic FEs **(3)** are referenced by the SUS through reference associations **(5)**. FEs which are assigned to the subsystem via reference associations (marked with a white diamond) are not part of the subsystem, but are only used there. They represent the local behaviour of the PDI of the corresponding SIUS and are part of it.
Eu.ModSt.7850	The SUS-specific FEs **(4)** are part of the SUS which is represented by composite associations **(6)**. FEs which are assigned to the subsystem via composite associations, i.e. so-called whole-part relationships (marked with a black diamond) are part of the subsystem. They represent the specific behaviour of the subsystem that influences more than one interface. This so-called "linking behaviour" is also used to link the behaviour assigned to the interfaces.
Eu.ModSt.7851	The model view "Functional Partitioning" forms the basis for the model view "Functional Architecture" (see *chapter 8.3.9*). It defines the FEs in their maximum quantity structure in the form of multiplicities. Within the framework of this quantity structure, the FE configurations required for the definition of the functional requirements are then created in the model view "Functional Architecture".

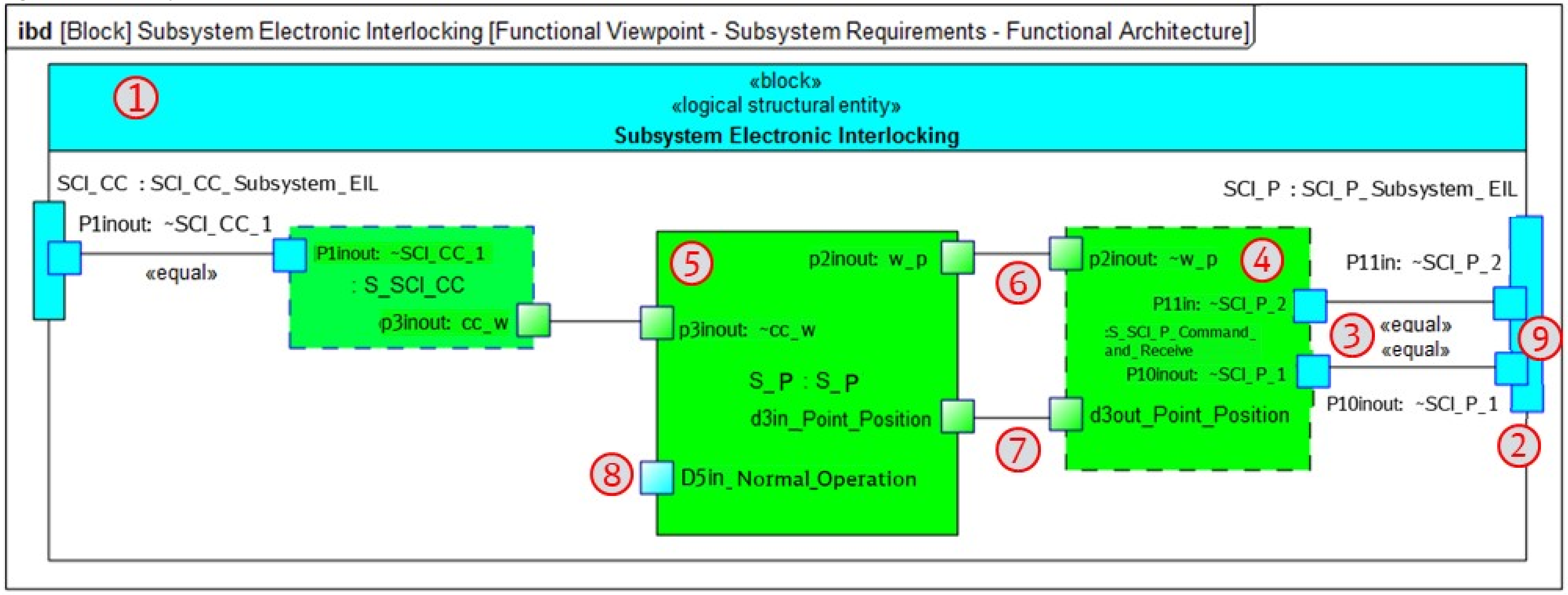
ID	Requirement
Eu.ModSt.7723	Figure 7723 Example of SUS model view "Functional Partitioning"



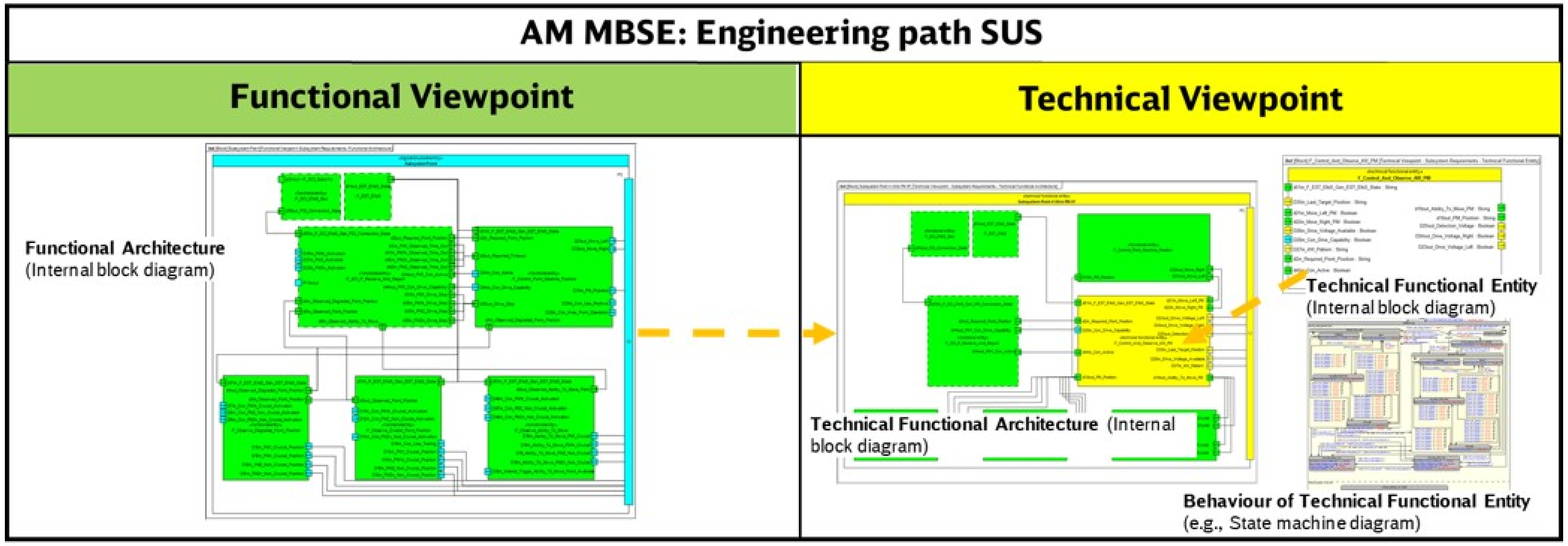
ID	Requirement
Eu.ModSt.7719	**8.3.8 Model view "Functional Partitioning" of a SUS (AL2) - Modelling rules**
Eu.ModSt.7780	**8.3.8.1 SysML diagram**
Eu.ModSt.7781	**Block Definition Diagram (bdd):** depicts the model view "Functional Partitioning".
Eu.ModSt.7782	**Diagram heading:** *bdd[Package]<><System block signature> <>->Functional Partitioning<> [Functional Viewpoint - Subsystem Requirements]*

ID	Requirement
Eu.ModSt.7783	**Example:** bdd [Package] Subsystem Point- Functional Partitioning [Functional Viewpoint - Subsystem Requirements]
Eu.ModSt.7811	**8.3.8.2 Model Elements**
Eu.ModSt.7812	*Remains free for the time being.*
Eu.ModSt.7843	**8.3.8.3 Binding** (see *chapter 8.2.1*)
Eu.ModSt.7852	**Diagram of model view "Functional Partitioning"** has a "**Def**" binding.
Eu.ModSt.7028	**8.3.9 Model view "Functional Architecture" of a SUS (AL2) - Description**
Eu.ModSt.7029	*Figure 7755* shows the model view "Functional Architecture" (FA) of Subsystem Point. It is created based on the in model view "Functional Partitioning" defined FEs.
Eu.ModSt.7755	Figure 7755 Model view "Functional Architecture" of Subsystem Point 
Eu.ModSt.7756	The model view "Functional Architecture" is explained in the following with a simple example as shown exemplarily in *Figure 7031*. It describes the external visible stimulus-response behaviour of a SUS **(1)** represented by a Logical Structural Entity (LSE) that is structured in a way that enables an interface centric specification approach as described in *chapter 8.2.4.* 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.

ID	Requirement
Eu.ModSt.7033	**Functional Entities** To describe the overall behaviour of an SUS observable externally in an FA structured, two different representations of the FEs **(4, 5)** are used: FEs with a solid border **(5)** and FEs with a dashed border **(4)**. Following the interface centric specification paradigm explained in *chapter 8.2.4*, a solid-bordered FE represents the directly specified behaviour of the SUS that is the "linking behaviour" (e.g. S_P : S_P). It is an inseparable part of the SUS behavioural model. 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 SUS by the directly specified SUS linking behaviour. The model view "Functional Entity" is described in *chapter 8.5* and *chapter 8.6*.
Eu.ModSt.7759	In *Figure 7031*, for example, the functional entity ":S_SCI_P_Command_and_Receive" is shown as a dashed block. This means that it is the local behaviour of the SCI-P protocol at application level, which is defined in the SCI-P specification (see *chapter 8.4*).
Eu.ModSt.7037	**Internal FE-coupling** Internal FE-couplings are implemented in two variants. In variant 1 **(6)**, communication between two FEs takes place by means of signals and in variant 2 **(7)**, permanent information is transmitted.
Eu.ModSt.7038	**Variant 1 (6):** 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) such as "w_p" can be exchanged. The information objects are represented by SysML signals (see *chapter 8.7.4* and *chapter 8.6.6.10.1*). 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.
Eu.ModSt.7039	Port name and port type are written in lower case. In addition, the ports are shown in the colour of the FEs.
Eu.ModSt.7040	**Variant 2 (7):** 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.
Eu.ModSt.7036	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.
Eu.ModSt.7885	**Please note:** In some cases, flow ports are still used to describe internal FE-couplings (see for example *Figure 7755*). However, these will gradually be replaced by proxy ports in the future.
Eu.ModSt.7041	Ports used for internal FE-coupling are defined as **functional ports**. Their names are written in lower case. In addition, the ports are shown in the colour of the FEs.
Eu.ModSt.7043	**External FE-coupling** The overall behaviour to be implemented by the manufacturers is connected to the **logical SUS interfaces (2)** via external FE-couplings **(3)**.
Eu.ModSt.7044	An external FE-coupling consists of a proxy port representing a logical SUS interface, located at the SUS outer boundary and labelled with the designator of the interface concerned (e.g. SCI_P : SCI_P_Subsystem_EIL). The proxy ports delegated from the FEs relevant to the interface using binding connectors **(3)** and representing the information flows (e.g. P11in : ~SCI_P_2 or P10inout : SCI_P_1) are embedded in it **(9)**.
Eu.ModSt.7860	In other words, the port (e.g. P10inout : ~SCI_P_1) at the FE is duplicated on the SUS outer boundary. Both ports are connected with a binding connector. The information flows and their direction remain unchanged in the interface block of the duplicated port.
Eu.ModSt.7045	The names of the proxy ports used in an external coupling (e.g. P11in or P10inout) designate the information flows assigned to the logical SUS interface. The port types (e.g. SCI_P_2 or SCI_P_1) define the information objects of the information flows that must be able to be exchanged via the respective interface.
Eu.ModSt.7861	The information objects defined in the information flows 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 object defined in an external FE-coupling only becomes a mandatory requirement in the context of its active use, e.g. in a transition.
Eu.ModSt.7884	**Please note:** In some cases, flow ports are still used to describe external FE-couplings (see for example interface P3 in *Figure 7755*). However, these will gradually be replaced by proxy ports in the future.
Eu.ModSt.7046	Ports used for external FE-coupling are defined as **logical ports**. Port name and port type are written in capital letters. In addition, the ports are shown in the colour blue.
Eu.ModSt.7049	**Open ports** Open ports **(8)** that is ports not associated to connectors define interfaces 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, as long as the information expected at the ports is provided or the information delivered via the ports is processed accordingly.
Eu.ModSt.7762	Ports used as open ports are defined as **logical ports**. Port name and port type are written in capital letters. In addition, the ports are shown in the colour blue.
Eu.ModSt.7050	Open ports are also used to configure the specified behaviour.
Eu.ModSt.7030	**Please note:** The FA is not to be understood as a specification for an internal architecture of the SUS, but as a descriptive structuring. The FEs in communication relationship represent the expected overall behaviour of a SUS, which must be fulfilled by the respective manufacturer in its entirety.

ID	Requirement
Eu.ModSt.7031	Figure 7031 Example of SUS model view "Functional Architecture" 
Eu.ModSt.1800	**8.3.10 Model view "Functional Architecture" of a SUS (AL2) - Modelling rules**
Eu.ModSt.1813	**8.3.10.1 SysML diagram**
Eu.ModSt.1832	**Internal Block Diagram (ibd):** depicts the model view "Functional Architecture".
Eu.ModSt.1833	**Diagram heading:** *ibd[Block]<><System block signature><>[Functional Viewpoint<>-<>System Requirements<>-<>Functional Architecture]*
Eu.ModSt.1834	**Example:** ibd[Block] Subsystem Electronic Interlocking[Functional Viewpoint-System Requirements-Functional Architecture]
Eu.ModSt.7758	**8.3.10.2 Model elements**
Eu.ModSt.7763	**Block:** Modular unit of structure in SysML that is used to define the Logical Structural Entity (LSE) representing the SUS at the Logical Viewpoint and the Functional Entities (FE) in the form of parts and reference properties.
Eu.ModSt.7764	**Part:** Parts **(5)** describe composition relationships between blocks. A composition relationship is also called a whole-part relationship. Thus, the parts used in a Functional Architecture of a SUS describe the composition relationships between the LSE and the corresponding FEs representing linking behaviour as introduced in *chapter 8.2.4*.
Eu.ModSt.7857	**Reference properties:** Reference properties **(4)** enable an instance of a block that contains the reference property to refer to an instance of the block which types the reference property. They can be used to describe a logical hierarchy that references blocks that are part of other composition hierarchies. Reference properties are depicted in a similar fashion to parts when shown on the internal block diagram, except that their box symbol has a dashed instead of a solid boundary. In the model view "Functional Architecture" of a SUS reference properties represent FEs which are references to the local behavioural parts of the interface application protocol as defined in model view "Functional Architecture" of the SIUS (see *chapter 8.4.5*).
Eu.ModSt.1137	**Part/reference property signature :=** *<Name of the part/reference property>:<FE_TFE block signature>*
Eu.ModSt.694	**Name of the part/reference property :=** 1) | 2) | 3) 1) A part/reference property is not named when the type (FE_TFE block signature) provides sufficient information to infer the role the part plays in the context of the Functional Architecture. 2) A part/reference property is given a name (Freely selectable designator) when the type (FE_TFE block signature) does not adequately describe the role the part plays in the context of the Functional Architecture. 3) A part/reference property is given a name (Freely selectable designator) when it is used within a SySim simulation.
Eu.ModSt.7858	**Example:** S_P : S_P :S_SCI_P_Command_and_Receive

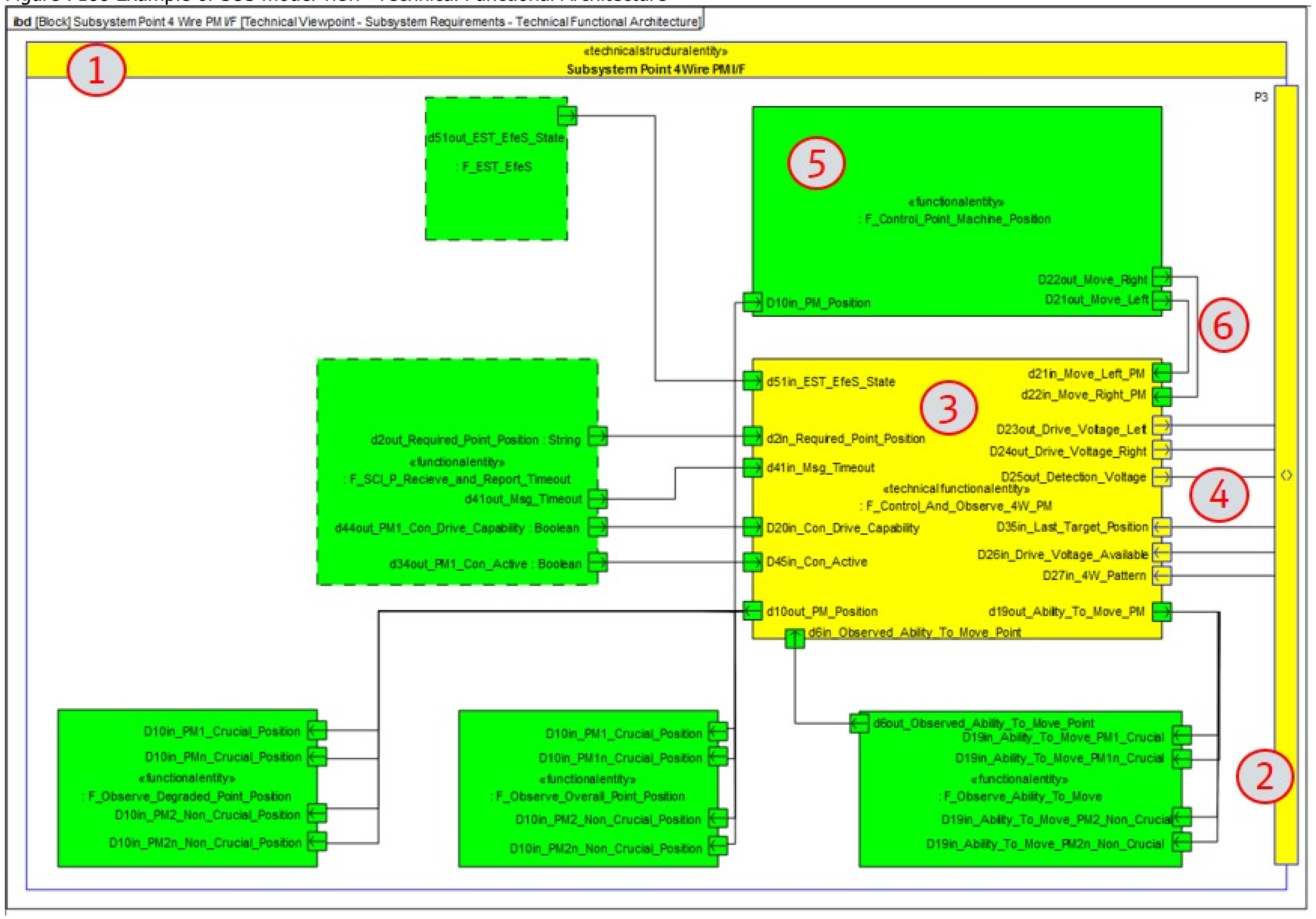
ID	Requirement
Eu.ModSt.7765	**Connector:** SysML connectors **(6,7)** are used to model the connections between parts or reference properties. Thus, they specify the communication-channels between the ports of FEs.
Eu.ModSt.7766	Whereas an out port of a FE may be connected to no connector or an infinite number of connectors, an in port may be connected to either no connector or only one connector, but must not be connected to more than one connector.
Eu.ModSt.7859	**Binding Connector:** A binding connector **(3)** is a special kind of connector that constrains its ends to have the same value. It is used, among other things, to bind proxy ports to parts or reference properties. For example, the value of the proxy port "P11in: ~SCI_P_2" **(9)** at the SUS interface **(2)** in *Figure 7031* corresponds to that of the port of the same name of the FE ":S_SCI_P_Command_and_Receive". A binding connector is shown using the connector notation, except that the connector path optionally has the keyword <<equal>> shown near its centre.
Eu.ModSt.7862	**Designator of a logical SUS interface** := *<Interface kind><>:<><Signature of Interface block aggregating information flows>*
Eu.ModSt.7863	*<Signature of Interface block aggregating information flows> := <Interface kind>_<System block signature>*
Eu.ModSt.7865	*<Interface kind>*: see *chapter 8.3.6.2* (Example: SCI_P)
Eu.ModSt.7867	*<System block signature>*: see *chapter 8.3.6.2* (Example: Subsystem_EIL)
Eu.ModSt.7864	**Example of a designator of a logical SUS interface:** SCI_P : SCI_P_Subsystem_EIL
Eu.ModSt.7868	**Designator of an Information flow** := *P<PNo><Port direction>_<Port information><>:<><Signature of Interface block aggregating information objects>*
Eu.ModSt.7869	*<PNo>, <Port direction>, <Port information>* are defined in *chapter 8.6.5.2*.
Eu.ModSt.7870	*<signature of Interface block aggregating information objects> := <Interface kind>_<IFNo>*
Eu.ModSt.7871	**Information flow number (IFNo):** natural number
Eu.ModSt.7872	**Example:** P11in : SCI_P_2 P10inout : SCI_P_1 P1inout : SCI_CC_1
Eu.ModSt.7948	**Please note:** Regarding the use of flow ports, flow specifications and flow properties see [**Eu.Doc.30]**.
Eu.ModSt.7760	**8.3.10.3 Binding** (see *chapter 8.2.1*)
Eu.ModSt.7761	**Diagram of model view "Functional Architecture"** has a "**Def**" binding.
Eu.ModSt.7197	**Ports** have a "**Def**" binding.
Eu.ModSt.7945	**Flow specifications** have an "**Info**" binding.
Eu.ModSt.7946	**FLow properties** of the flow specifications have a "**Def**" binding if they are further refined elsewhere (e.g. by linked telegram definitions in separate interface specifications or further requirements in chapter 5.X. of the subsystem requirements specification in the requirements management tool).
Eu.ModSt.7947	**FLowProperties** of the FlowSpecifications have a "**Req**" binding if they are not further refined elsewhere.
Eu.ModSt.7186	**8.3.11 Model view "Technical Functional Architecture" of a SUS (AL2) - Description**
Eu.ModSt.7193	*Figure 7194* shows the engineering path of the model views used to specify a SUS at the Technical Viewpoint on abstraction level AL2.

ID	Requirement
Eu.ModSt.7194	Figure 7194 Engineering path to specify a SUS at the Technical Viewpoint on abstraction level AL2 
Eu.ModSt.7767	The model view "Technical Functional Architecture" (TFA) supplements or substitutes the behaviour described in the model view "Functional Architecture", which is independent of technology, with behavioural components derived from technical requirements. In other words, the FEs interconnected in the model view "Functional Architecture" are either transferred to the model view "Technical Functional Architecture" or completely or partially replaced by Technical Functional Entities (TFE).
Eu.ModSt.7769	The SUS can either be described completely from a technical point of view (all FEs are replaced by TFEs) or only certain parts of it (interconnection of TFEs and transferred FEs).
Eu.ModSt.7192	*Figure 7188* shows an example of the transfer of the FES defined in the model view "Functional Architecture" to the model view "Technical Functional Architecture" of the SUS Subsystem Point. The SUS **(1)** is represented by a Technical Structural Entity (TSE). The transferred FEs **(5)** are supplemented with the TFE "F_Control_And_Observe_4W_PM" **(3)** that describes the functionality of the four-wire interface to a point machine based on technical requirements.
Eu.ModSt.7189	In model view "Technical Functional Architecture" TFEs are coupled with each other, with the already defined FEs **(6)** and with the environment **(4)** via **external technical functional interfaces (2)**.
Eu.ModSt.7886	The overall behaviour of a SUS structured by a TFA can be divided into several TFAs in the graphical representation.
Eu.ModSt.7887	**Technical Functional Entities** To describe the overall behaviour of an SUS observable externally in an TFA structured, two different representations of the TFEs are used: TFEs with a solid border **(3)** and TFEs with a dashed border. Following the interface centric specification paradigm explained in *chapter 8.2.4*, a solid-bordered FE represents the directly specified behaviour of the SUS that is the "linking behaviour". It is an inseparable part of the SUS behavioural model. TFEs 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 SUS by the directly specified SUS linking behaviour. The model view "Technical Functional Entity" is described in *chapter 8.5* and *chapter 8.6*.
Eu.ModSt.7888	**Internal TFE-coupling and external TFE-coupling** The definitions for internal FE-coupling and external FE-coupling in *chapter 8.3.9* apply accordingly.
Eu.ModSt.7889	Ports used for external TFE-coupling and internal TFE-coupling are defined as **technical functional ports**. They are shown in the colour yellow **(4)**.
Eu.ModSt.7890	Ports used for internal coupling of FEs with TFEs are **functional ports**. They are shown in the colour green **(6)**.
Eu.ModSt.7891	Ports representing **technical functional SUS interfaces (2)** can only be connected to technical functional ports **(4)**.
Eu.ModSt.7892	**Open ports** Open ports that is ports not associated to connectors define interfaces to specification parts not contained in the model, i.e. expected behaviour in the environment of the TFEs. This behaviour can be implemented proprietarily by each manufacturer, as long as the information expected at the ports is provided or the information delivered via the ports is processed accordingly.
Eu.ModSt.7893	Ports used as open ports are defined as **logical ports**. Port name and port type are written in capital letters. In addition, the ports are shown in the colour blue.

ID	Requirement
Eu.ModSt.7883	**Please note:** The TFA is not to be understood as a specification for an internal architecture of the SUS, but as a descriptive structuring. The TFEs or FEs in communication relationship represent the expected overall behaviour of a SUS, which must be fulfilled by the respective manufacturer in its entirety.

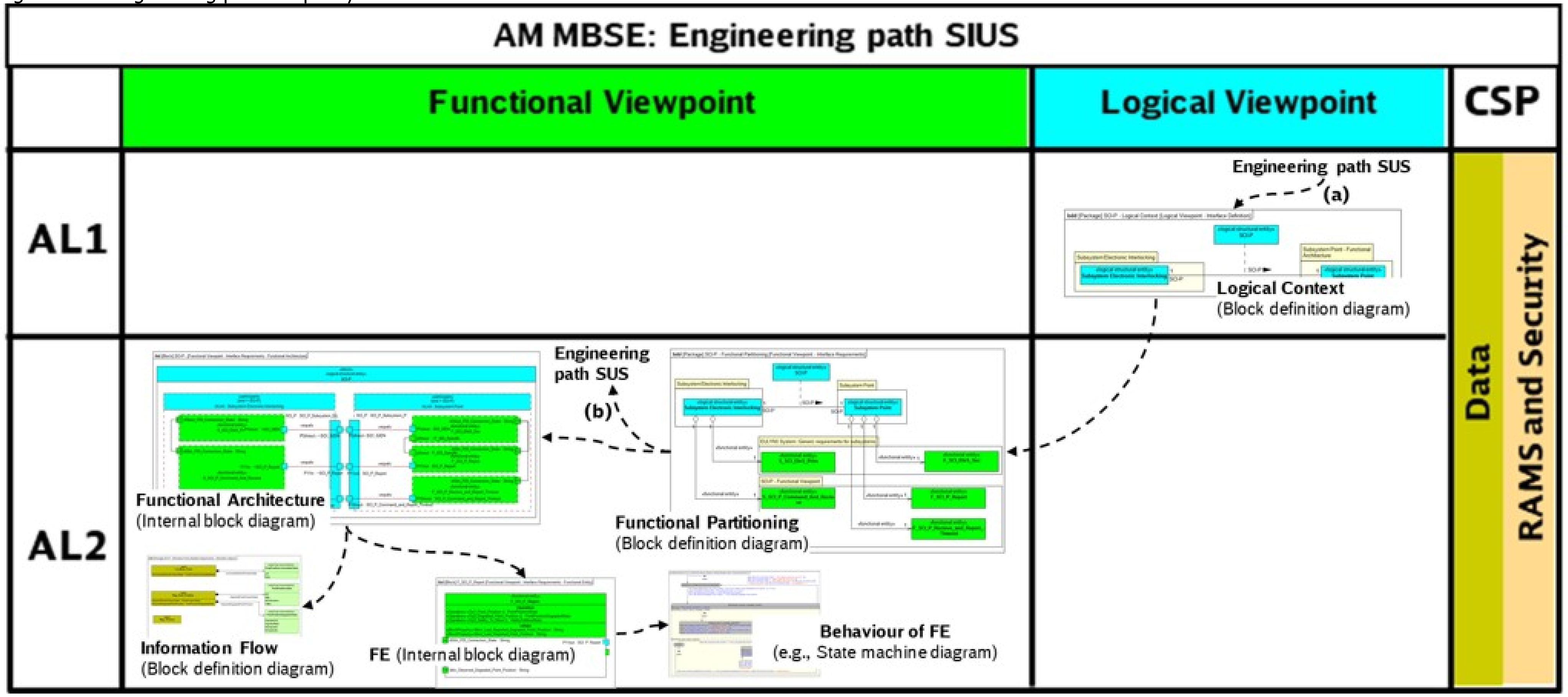
Eu.ModSt.7188

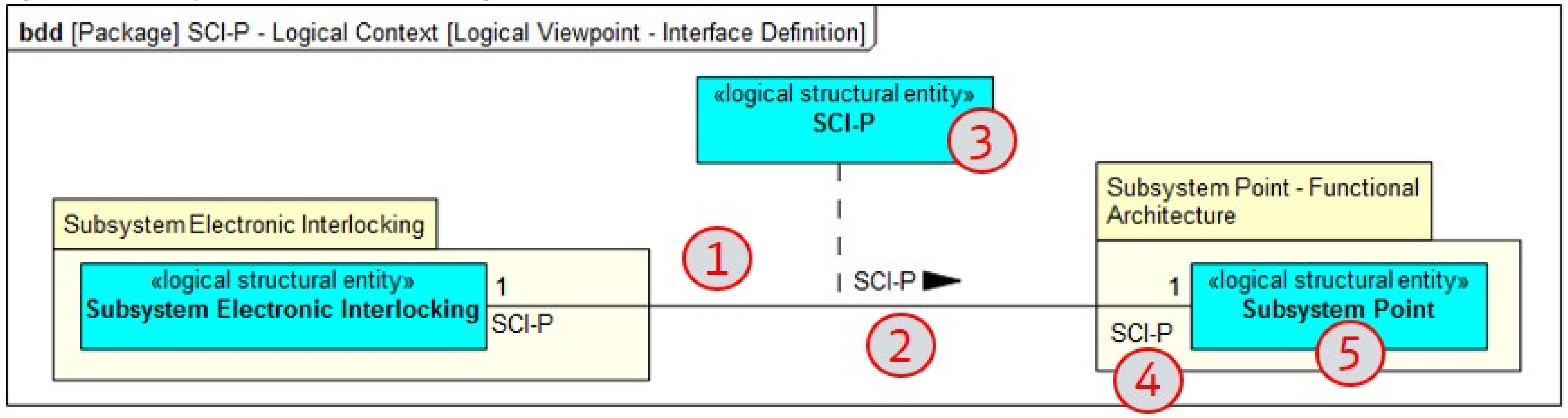
Figure 7188 Example of SUS model view "Technical Functional Architecture"

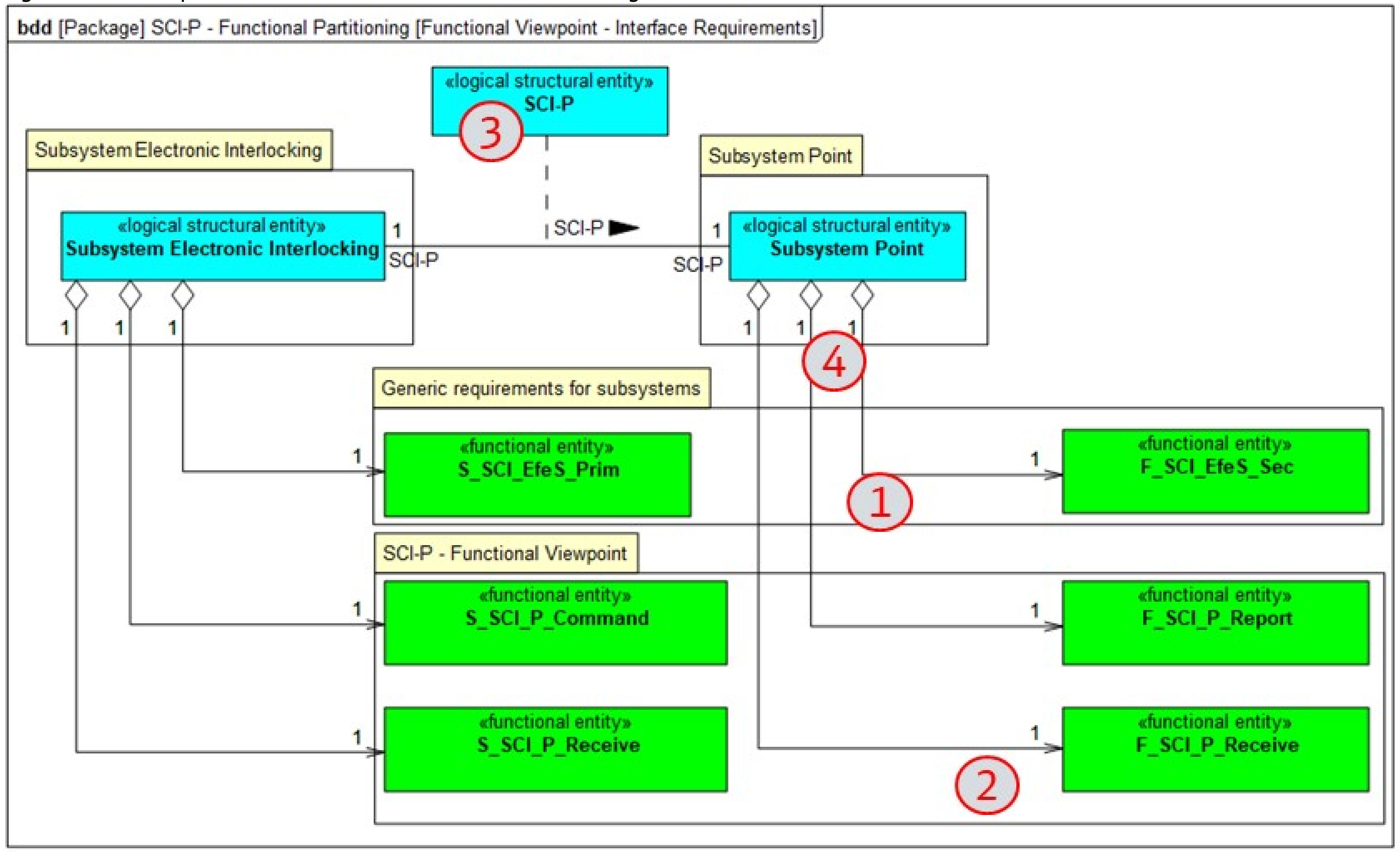


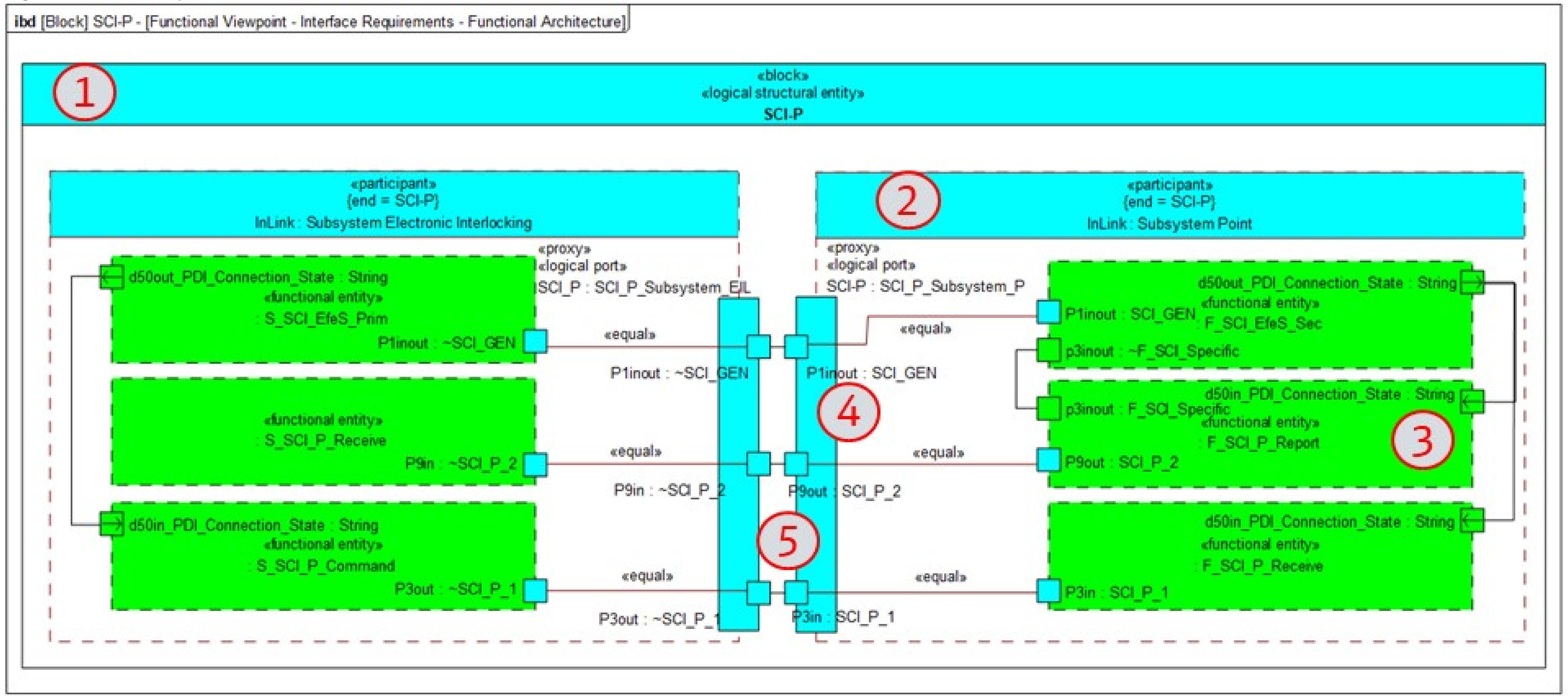
ID	Requirement
Eu.ModSt.7301	**8.3.12 Model view "Technical Functional Architecture" of a SUS (AL2) - Modelling rules**
Eu.ModSt.7303	**8.3.12.1 SysML diagram**
Eu.ModSt.7304	**Internal Block Diagram (ibd):** depicts the model view "Technical Functional Architecture"
Eu.ModSt.7305	**Diagram heading:** *ibd[Block]<><System block signature><>[Technical Viewpoint<>-<>Subsystem Requirements<>-<>Technical Functional Architecture]*
Eu.ModSt.7306	**Example:** ibd [block] Subsystem Point 4 Wire PM I/F [Technical Viewpoint - Subsystem Requirements - Technical Functional Architecture]
Eu.ModSt.7307	**8.3.12.2 Model elements**
Eu.ModSt.7308	**Block:** Modular unit of structure in SysML that is used to define the TSE representing the technical manifestation of the SUS.
Eu.ModSt.7310	**Please note:** For the remaining model elements, the definitions in *chapter 8.3.10.2* apply accordingly.

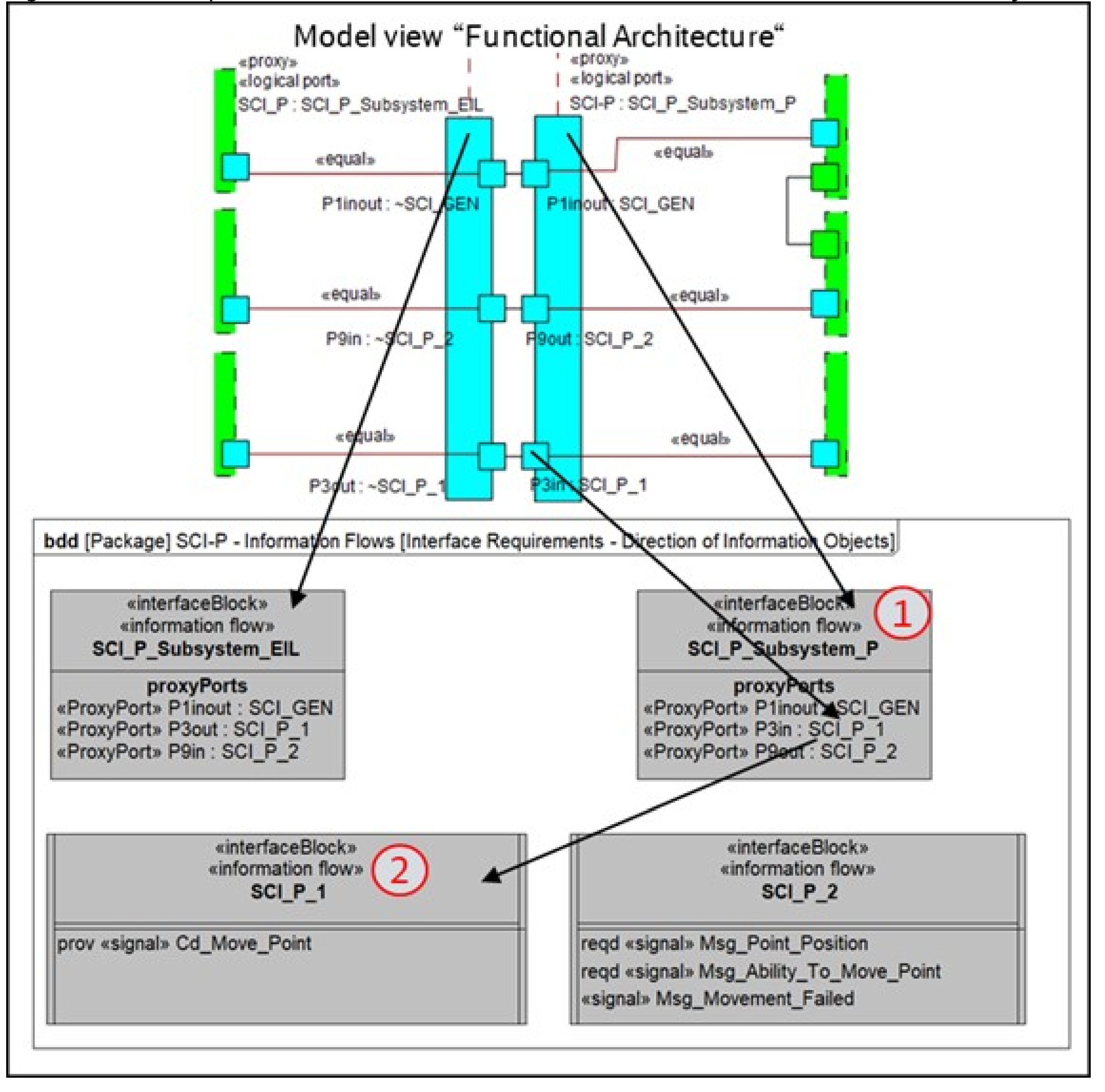
ID	Requirement
Eu.ModSt.7330	**8.3.12.3 Bindings** (see *chapter 8.2.1*)
Eu.ModSt.7333	**Diagram of model view "Technical Functional Architecture"** has a "**Def**" binding.
Eu.ModSt.7335	**Ports** have a "**Def**" binding.
Eu.ModSt.7336	**Technical functional SUS interface** has a "**Req**" binding if it is not further specified in a refined model view or in the form of a separate requirement.
Eu.ModSt.2486	**8.4 Model views used to specify EULYNX interfaces**
Eu.ModSt.2238	**Model view "Logical Context": Block Definition Diagram (bdd)** The model view "Logical Context" describes the logical view of an interface at the upper level of abstraction.
Eu.ModSt.2239	**Model view "Functional Partitioning": Block Definition Diagram (bdd)** The model view "Functional Partitioning" describes the refinement of the interface defined in model view "Logical Context" using Functional Entities.
Eu.ModSt.2240	**Model view "Functional Architecture": Internal Block Diagram (ibd)** The model view "Functional Architecture" defines the global behaviour of the application protocol (see *chapter 8.2.4*).
Eu.ModSt.2241	**Model view "Functional Entity": Internal Block Diagram (ibd) and State Machine (stm)** The model view "Functional Entity" encapsulates a subset of the functional requirements of an 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. As the model view "Functional Entity" is used for the specification of EULYNX system elements as well as for the specification of EULYNX interfaces it is described in the separate *chapter 8.5* and *chapter 8.6*.
Eu.ModSt.2242	**Model view "Information Flow": Block Definition Diagram (bdd)** The model view „Information Flow" describes the information objects to be exchanged via an interface which are further refined to telegrams at abstraction level AL3. At present, the telegrams are not yet described in a model-based way. They are defined in the interface specifications (e.g. Interface Specification SCI-P, [Eu.Doc.38]).
Eu.ModSt.2243	*Figure 2244* shows the engineering path of the model views used to specify a SIUS considering the Functional Viewpoint and the Logical Viewpoint. It describes the context of the model views, with the arrows indicating which model views are developed from which. Based on the definition of the logical SUS interfaces in model view "Logical Context" of the SUS (**a**: see *Figure 2129* in *chapter 8.3*) the model views "Logical Context" and "Functional Partitioning" of the corresponding SIUS are created. The model view "Functional Partitioning" in turn forms the basis for the creation of the model view "Functional Architecture" of the SIUS and the model view "Functional Partitioning" of the SUS (**b**: see *Figure 2129* in *chapter 8.3*). Subsequently, the model views "Information Flow" and "Functional Entity" are created.

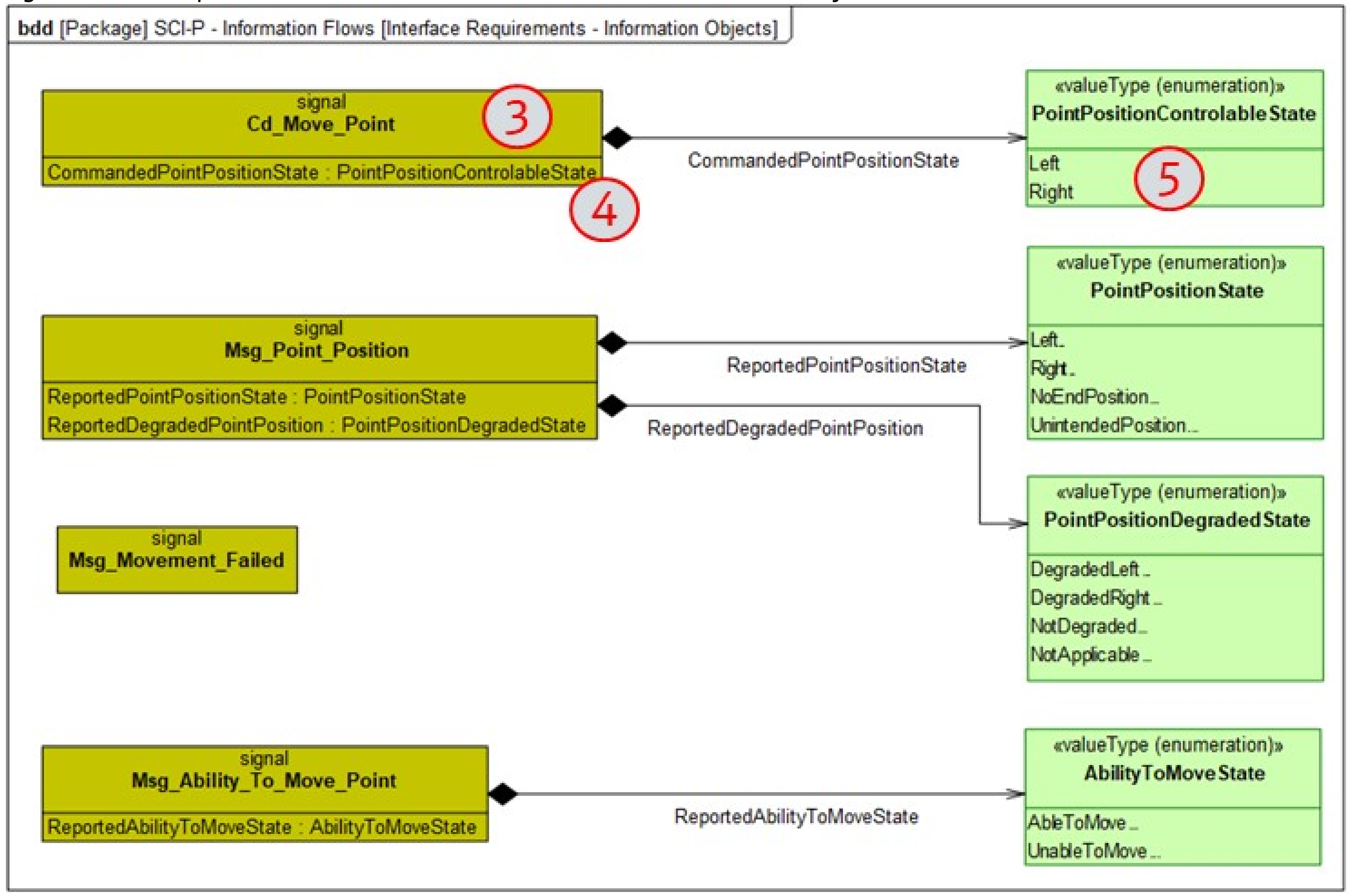
ID	Requirement
Eu.ModSt.2244	Figure 2244 Engineering path to specify a EULYNX interface 
Eu.ModSt.7229	**8.4.1 Model view "Logical Context" of a SIUS (AL1) - Description**
Eu.ModSt.7230	The model view "Logical Context" as shown in *Figure 7231* describes the logical view of an interface at the upper level of abstraction. In contrast to the logical context of a SUS in which the logical interfaces are also defined in terms of their number, an interface in its logical context is regarded as a one-to-one relationship.
Eu.ModSt.7233	An interface **(1)** is generally defined as a unique connection between two communication participants **(5)**. From the logical viewpoint at the upper level of abstraction an interface is represented by a SysML association **(1).** An association is depicted as a continuous line between the communication participants. It also represents the possible interaction directions of the interface. No arrow heads means that the interaction is bidirectional. An arrow head on the other hand indicates that an interaction is only possible in the direction of the arrow. It represents the requirement that the two communication participants must be able to interact with each other.
Eu.ModSt.7235	The logical interface represented by an association **(1)** is linked to a SysML association block **(3)**, which serves to refine the relationship. The global behaviour of the application protocol (Railway Control Protocol: RCP) is then specified in this later in the model view "Functional Architecture".
Eu.ModSt.7237	A defined set of information objects (information flow) is transmitted via the interface in a precisely defined temporal sequence (protocol) in many cases. An information flow and the corresponding definition of the temporal sequence can apply to different interfaces. These two properties of an interface are called interface kind **(4).** The interface kind is mapped at the association ends in the form of roles **(4).** This separation of interface and interface kind makes it possible to communicate in the same way via several different "unique relationships = interfaces". The interface kind represents the requirement that it is to be applied to a specific interface.
Eu.ModSt.7239	An interface is identified by a unique name **(2)** placed above or below the association **(1)** representing the interface.
Eu.ModSt.7240	The black arrow shown in connection with the association indicates the reading direction. The directional arrow specifies the top-level navigation through the interface model to improve readability. It is taken into account when refining the model, for example when defining the conjugation of information flows. Beyond that, it has no meaning for the model.
Eu.ModSt.7241	The interface name can be identical to the interface kind if it is certain that the interface kind is only applied to a specific interface and not to several different ones. If the interface name is the same as the interface kind, it may not be displayed.

ID	Requirement
Eu.ModSt.7231	Figure 7231 Example of SIUS model view "Logical context" 
Eu.ModSt.1730	**8.4.2 Model view "Logical Context" of a SIUS (AL1) - Modelling rules**
Eu.ModSt.1732	**8.4.2.1 SysML diagram**
Eu.ModSt.1733	**Block definition diagram (bdd):** depicts the view Technical Connection Domain Context.
Eu.ModSt.1734	**Diagram heading:** *bdd[Package]<><Interface name><>-<>Logical Context<>[Logical Viewpoint<>-<>Interface Definition].*
Eu.ModSt.1735	**Example:** bdd SCI-P - Logical Context [Logical Viewpoint - Interface Definition]
Eu.ModSt.7238	**8.4.2.2 Model elements**
Eu.ModSt.7784	**Block:** Modular unit of structure in SysML that is used to define the LSE representing the communication participants that is, the communicating subsystems **(5)**.
Eu.ModSt.1364	**Association block (3):** an association block is a combination of an association and a block, so it can relate two blocks together but can also have internal structure and other features. The internal structure can be used to decompose the connector that is typed by the association block. Association blocks are shown on block definition diagrams as an association path with a block symbol attached to it via a dashed line.
Eu.ModSt.7786	**8.4.2.3 Bindings** (see *chapter 8.2.1*)
Eu.ModSt.7787	**Diagram of model view "Logical Context"** has a "**Def**" binding.
Eu.ModSt.2260	**8.4.3 Model view "Functional Partitioning" of a SIUS (AL2) - Description**
Eu.ModSt.2261	The model view "Functional Partitioning" as shown in *Figure 2262* describes the refinement of the interface defined in model view "Logical Context" using Functional Entities. These Functional Entities specify the local behaviours of the communication protocol stack scaled-down to the application layer (PDI: Process Data Interface Protocol) at each side of the communicating system elements.
Eu.ModSt.2264	The specific **(2)** and generic **(1)** local behavioural parts of the application protocol defined by FEs are referenced by the communication partners via SysML reference associations **(4)**. Reference associations are marked with a white diamond and express that the FEs are not part of the subsystems, but are only used there. They are part of the PDI.
Eu.ModSt.7904	The FEs are used in the model view "Functional Architecture" to specify the global behaviour of the application protocol represented by the internal structure of the association block **(3)** associated with the association representing the interface.

ID	Requirement
Eu.ModSt.2262	Figure 2262 Example of SIUS model view "Functional Partitioning" 
Eu.ModSt.1359	**8.4.4 Model view "Functional Partitioning" of a SIUS (AL2) - Modelling rules**
Eu.ModSt.1361	**8.4.4.1 SysML diagram**
Eu.ModSt.1362	**Block Definition Diagram (bdd):** depicts the model view "Functional Partitioning".
Eu.ModSt.1405	**Diagram heading:** *bdd[Package]<><Interface name><>-<>Functional Partitioning<>[Functional Viewpoint<>-<>Interface Requirements]*
Eu.ModSt.1406	**Example:** bdd SCI-P - Functional Partitioning [Functional Viewpoint - Interface Requirements]
Eu.ModSt.1363	**8.4.4.2 Model elements**
Eu.ModSt.1407	*Remains free for the time being.*
Eu.ModSt.7796	**8.4.4.3 Bindings** (see *chapter 8.2.1*)
Eu.ModSt.7797	**Diagram of model view "Functional Partitioning"** has a "**Def**" binding.
Eu.ModSt.2265	**8.4.5 Model view "Functional Architecture" of a SIUS (AL2) - Description**
Eu.ModSt.2266	The model view "Functional Architecture" as shown in *Figure 2267* defines the global behaviour of the application protocol. The global behaviour is described by connecting the local behavioural components referenced by a communication partner with the corresponding ones of the neighbour via communication channels.
Eu.ModSt.2269	The description of the global behaviour of the application protocol is done by the internal structuring of the association block **(1)** defined in the model view "Functional Partitioning". In this process, the communication partners **(2)**, which in turn reference the local behavioural parts of the protocol represented by FEs **(3)**, are referenced in the form of SysML participant properties and connected via their logical SUS interfaces **(4)** with connectors **(5)**.

ID	Requirement
Eu.ModSt.2267	Figure 2267 Example of SIUS model view "Functional Architecture" 
Eu.ModSt.7203	**8.4.6 Model view "Functional Architecture" of a SIUS (AL2) - Modelling rules**
Eu.ModSt.1370	**8.4.6.1 SysML diagram**
Eu.ModSt.1371	**Internal Block Diagram (ibd):** depicts model view "Functional Architecture".
Eu.ModSt.1410	**Diagram heading:** *ibd[Block]<><Interface name><>[Functional Viewpoint<>-<>Interface Requirements<>-<>Functional Architecture]*
Eu.ModSt.1411	**Example:** ibd[Block] SCI-P [Functional Viewpoint - Interface Requirements - Functional Architecture]
Eu.ModSt.1372	**8.4.6.2 Model elements**
Eu.ModSt.1963	**Participant property:** Participant properties are placeholders that represent the blocks at the end of an association block, and are used when it is desired to decompose a connector. A participant property is depicted as a dashed box, like a reference property, but distinguished from other properties by the keyword <<participant>>.
Eu.ModSt.7802	**8.4.6.3 Bindings** (see *chapter 8.2.1*)
Eu.ModSt.7803	**Diagram of model view "Functional Architecture"** has a "**Def**" binding.
Eu.ModSt.7909	**Ports** have a "**Def**" binding.
Eu.ModSt.2270	**8.4.7 Model view "Information Flow" of a SIUS (AL2) - Description**
Eu.ModSt.2271	The model view "Information Flow" describes the information objects to be exchanged via an interface. It consists of the two sub-model views "Direction of Information Objects" and "Information Objects", which are shown in *Figure 7774* and *Figure 2272* respectively.
Eu.ModSt.7807	As shown in *Figure 7774*, the SUS interfaces such as SCI_P are depicted by proxy ports. These are typified with interface blocks such as SCI_P_Subsystem_P **(1)**, which represent information flows in the form of embedded proxy ports such as P10inout. The embedded proxy ports are typed with interface blocks **(2)**, which in turn contain the information objects (e.g. Cd_Move_Point).

ID	Requirement
Eu.ModSt.7774	Figure 7774 Example of SIUS model view "Information flow" - Direction of Information Objects 
Eu.ModSt.1376	As shown in *Figure 2272*, the information objects are represented by SysML signals such as "Cd_Move_Point" **(3)**. These signals can in turn have attributes such as "CommandedPointPositionState" **(4)** that represent parameters of the information objects. The attributes are typed with basic data types or for example enumerations such as "PointPositionControlableState" **(5)**.

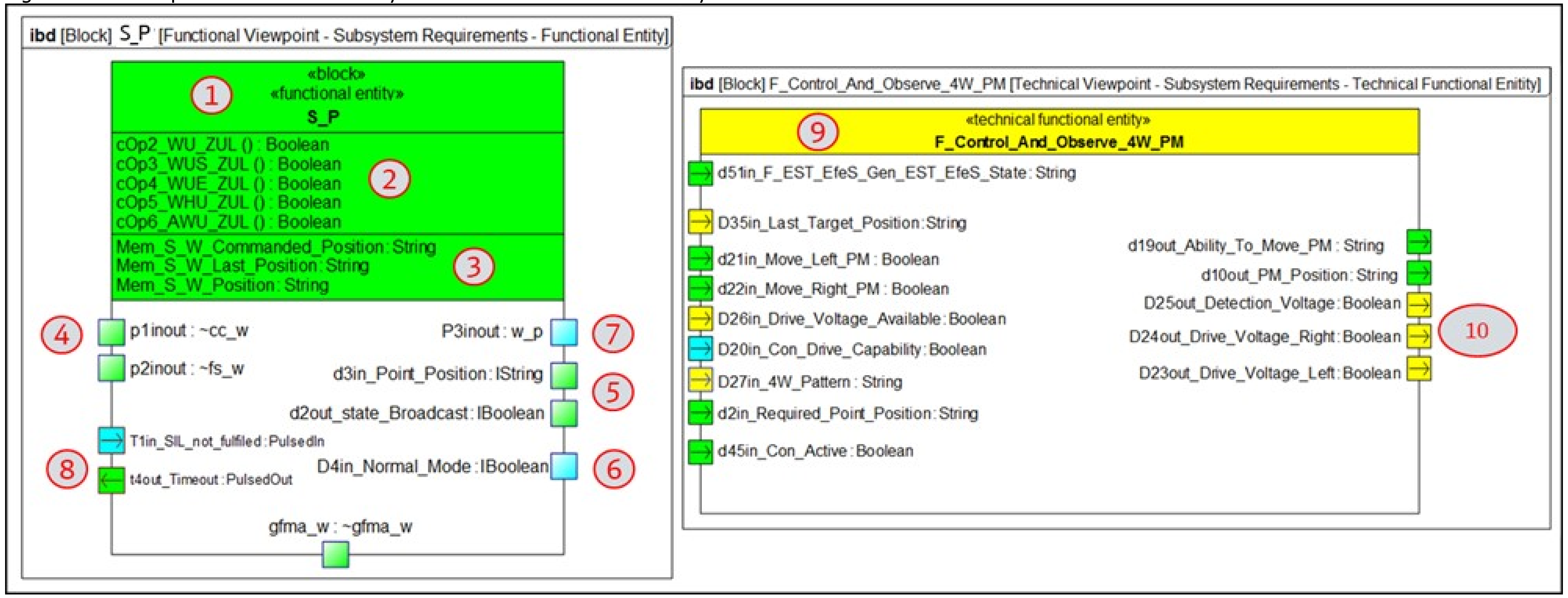
ID	Requirement
Eu.ModSt.2272	Figure 2272 Example of SIUS model view "Information flow" - Information Objects 

ID	Requirement
Eu.ModSt.7842	**Please note:** These model views can also be used in an adapted form to define the information flows for internal couplings between FEs or TFEs in a Functional Architecture or Technical Functional Architecture.
Eu.ModSt.7206	**8.4.8 Model view "Information Flow" of a SUS - Modelling Rules**
Eu.ModSt.1379	**8.4.8.1 SysML diagram**
Eu.ModSt.1380	**Block Definition Diagram (bdd):** depicts the sub-model views "Direction of Information Objects" and "Information Objects" of model view "Information Flow".
Eu.ModSt.1414	**Diagram heading (sub-model view "Direction of Information Objects"):** *bdd[Package]<><Interface name><>-<><Information Flows><>[Interface Requirements<>-<>Direction of Information Objects*
Eu.ModSt.1378	**Diagram heading (sub-model view "Information Objects"):** *bdd[Package]<><Interface name><>-<><Information Flows><>[Interface Requirements<>-<>Information Objects*
Eu.ModSt.1417	**Example:** bdd[Package] SCI-P - Information Flows [Interface Requirements - Direction of Information Objects] bdd[Package] SCI-P - Information Flows [Interface Requirements - Information Objects]
Eu.ModSt.1381	**8.4.8.2 Model elements**
Eu.ModSt.1416	*Remains free for the time being.*
Eu.ModSt.7099	**8.4.8.3 Bindings** (see *chapter 8.2.1*)
Eu.ModSt.7106	**Diagram of model view "Information Flows - Direction of Information Objects"** has a "**Def**" binding.
Eu.ModSt.7100	**Diagram of model view "Information Flows - Information Objects"** has a "**Def**" binding.

ID	Requirement
Eu.ModSt.7107	**Information Objects (Signals)** have a "**Def**" binding if they are further specified in a refined model view or in the form of a separate requirement.
Eu.ModSt.7905	**Information Objects (Signals)** have a "**Req**" binding if they are not further specified in a refined model view or in the form of a separate requirement.
Eu.ModSt.1249	**8.5 Model views "Functional Entity" and "Technical Functional Entity" - Description**
Eu.ModSt.7487	Within the EULYNX approach to specify model-based requirements the concept of Functional Entity (FE) and Technical Functional Entity (TFE) is used.
Eu.ModSt.7488	FE and TFE represent behavioural entities and encapsulate a subset of the functional requirements of a SUS or SIUS in the form of stimulus-response behaviour independent of any architectural constraints. While FEs define technology-independent functional requirements, TFEs describe technology-dependent ones.
Eu.ModSt.7489	**Please note:** FEs and TFEs are not to be interpreted as elements of the hardware- or software architecture.
Eu.ModSt.7490	The stimulus-response behaviour of FEs and TFEs is defined by SysML state machines (see *chapter 8.6.6*).
Eu.ModSt.7491	The principle structure of a Functional Entity and a Technical Functional Entity is shown in *Figure 7492.*

Eu.ModSt.7492

Figure 7492 Example of a Functional Entity and a Technical Functional Entity



ID	Requirement
Eu.ModSt.7493	Apart from state machines, FEs and TFEs may own • SysML block properties **(3)**, • SysML block operations **(2),** • SysML proxy ports used as atomic "in ports" and "out ports" **(5, 6)** or typed with an interface block in which the information objects to be exchanged via the port are defined **(4, 7)**, • SysML flow ports used as atomic "in ports" and "out ports" **(8, 10)**.
Eu.ModSt.7494	The description of a FE **(1)** contains the stereotype <<functional entity>> as well as the FE name (e.g. S_W).
Eu.ModSt.7495	The description of a TFE **(9)** contains the stereotype <<technical functional entity>> as well as the TFE name (e.g. F_Control_And_Observe_4W_PM).
Eu.ModSt.7808	**8.6 Model views "Functional Entity" and "Technical Functional Entity" - Modelling rules**
Eu.ModSt.7829	The numbers **(2)** to **(10)** added in the following descriptions refer to Figure 7492.
Eu.ModSt.7809	**8.6.1 SysML Diagram**
Eu.ModSt.7815	**Internal Block Diagram (ibd):** depicts model views "Functional Entity" and "Technical Functional entity".

ID	Requirement
Eu.ModSt.7816	**Diagram heading - FE:** *ibd[Block]<><FE_TFE block signature><>[Functional Viewpoint<>-<>Subsystem Requirements<>-<>Functional Entity*]
Eu.ModSt.7817	**Diagram heading - TFE:** *ibd[Block]<><FE_TFE block signature><>[Functional Viewpoint<>-<>Subsystem Requirements<>-<>Technical Functional Entity*]
Eu.ModSt.7818	**Example:** ibd[Block] S_Point [Functional Viewpoint - Subsystem Requirements - Functional Entity] ibd[Block] F_Control_And_Observe_4W_PM [Functional Viewpoint - Subsystem Requirements - Technical Functional Entity]
Eu.ModSt.7819	**8.6.2 Block**
Eu.ModSt.7820	**Block:** Modular unit of structure in SysML that is used to define a FE or TFE
Eu.ModSt.7821	**Block name:** *<FE_TFE block signature>*
Eu.ModSt.7822	**Example:** S_P F_Control_And_Observe_4W_PM
Eu.ModSt.906	**<FE_TFE block signature> :=** <Layer of LA modelling pattern >_ <Name of functionality>_<Operational entity>
Eu.ModSt.911	**<Layer of LA modelling pattern>** := C | S | F | "" **C**: Command control layer, **S**: Safety layer, **F**: Field layer "": if no layer is applicable See *chapter 8.2.2*
Eu.ModSt.916	**<Name of functionality>** := 1) | 2) | 3) | 4) | 5) | 6) 1) FE/TFE specifies the essential states of an operational entity (operating modes): **EST** 2) FE/TFE specifies the behaviour of an operational entity: **<description of the functionality>** (example: Control_And_Observe_4W_PM) 3) FE/TFE specifies local behaviour of the application protocol layer (RCP) assigned to a certain operational entity (see *chapter 8.2.4*): **<Interface name>** (example: SCI_P) or **<Interface name>_<description of the functionality>** (example: SCI_P_Report_Status) 4) FE/TFE specifies generic local behaviour of the application protocol layer (RCP): **<Abbr. Type of interface>_Gen (** example: SCI_Gen) **<Abbr. Type of interface>_<description of the functionality>_Gen** (example: SCI_Check_Version_Gen) 5) FE/TFE specifies generic local behaviour of the application protocol layer (RCP) assigned to a certain group of operational entities: **<Abbr. Type of interface>_<Operational entity_Operational entity_..._Operational entity>_Gen** (example: SCI_LS_P_Gen) or **<Abbr. Type of interface>_<Operational entity_Operational entity_..._Operational entity>_<description of the functionality>_Gen** (example: SCI_LS_P_Check_Version_Gen) 6) FE/TFE specifies generic local behaviour of the application protocol layer (RCP) assigned to a certain group of operational entities using a common designator: **<Abbr. Type of interface>_<group designator>_Gen (** example: SCI_EfeS_Gen) or **<Abbr. Type of interface>_<group designator>_<role of communication partner>** (SCI_EfeS_Prim) **<Abbr. Type of interface>_<group designator>_<description of the functionality>_Gen** (example: SCI_EfeS_Check_Version_Gen) **<Abbr. Type of interface>_<group designator>_<role of communication partner>_<description of the functionality>** (example: SCI_EfeS_Prim_Check_Version) **<group designator>** := Freely selectable common designator (example: FE for field elements) **<role of communication partner>** := freely selectable designator such as Prim (Primary) and Sec (Secondary)
Eu.ModSt.966	**<Operational entity>** := 1) | 2) | 3) | 4) | 5) 1) FE/TFE specifies the behaviour or the essential states of an operational entity: Name of the operational entity (vertical slice of the LA modelling pattern) Examples: LS, P, SOR (start of route), EOR (end of route) 2) FE/TFE specifies generic behaviour or the essential states of an operational entity: **Gen** 3) FE/TFE specifies generic behaviour or the essential states assigned to a certain group of operational entities: **<Operational entity_Operational entity_..._Operational entity>_Gen** (example: LS_P_Gen) 4) FE/TFE specifies generic behaviour or the essential states assigned to a certain group of operational entities using a common designator: **<group designator>_Gen** (example: EfeS_Gen) **<group designator>** := Freely selectable common designator (example: FE for field elements) 5) FE/TFE specifies the local behaviour of the application protocol layer (RCP): no operational entity
Eu.ModSt.7810	**8.6.3 Model elements - Block properties**

<table>
<tr><th>ID</th><th>Requirement</th></tr>
<tr><td>Eu.ModSt.7497</td><td>Block properties (3) are to be interpreted in the sense of variables or constants that store values. They are prefixed with "Mem".
Examples: Mem_last_Target_Requested, Mem_Current_Point_Position.</td></tr>
<tr><td>Eu.ModSt.534</td><td>Block properties are to be typed using the defined SySim value types.</td></tr>
<tr><td>Eu.ModSt.533</td><td>All SysML block properties have to be initialised. The initialisation must be carried out in an init-operation using ASAL. This SysML block operation is systematically named cOp1_init().</td></tr>
<tr><td>Eu.ModSt.7498</td><td>The initialisation can be carried out in the body of the init-block operation systematically named cOp1_init(). Alternatively it can be carried out directly in the transition effect of the transition outgoing from initial state of the state machine.
Example:
Mem_S_W_Position := "";
Mem_SW_Last_Position := "";
The assignments of values to the corresponding block properties are to be interpreted as definitions. They become mandatory requirements (binding character "Req") when they are used in a mandatory requirement, such as a transition of a state.</td></tr>
<tr><td>Eu.ModSt.536</td><td>Some reasons to use SysML block properties are given below. This is expressed by means of corresponding naming conventions:</td></tr>
<tr><td>Eu.ModSt.539</td><td>Defining configuration data: Con_data-name (e.g. Con_t_ini_max)</td></tr>
<tr><td>Eu.ModSt.540</td><td><blockpropertyname> ::= <Con><mark><propertyinformation>
<propertyinformation>::= <alphaNum><remaininginformation>
<remaininginformation> ::= „" | <alphaNum><remaininginformation>
<Con>::= Con
<alphaNum> ::= A | B | … | Z | a | b | … | _ | 0 | ... | 9
<mark>::= _</td></tr>
<tr><td>Eu.ModSt.897</td><td>Defining site data: Site_data-name</td></tr>
<tr><td>Eu.ModSt.898</td><td><blockpropertyname> ::= <Site><mark><propertyinformation>
<propertyinformation>::= <alphaNum><remaininginformation>
<remaininginformation> ::= „"|<alphaNum><remaininginformation>
<Site>::= Site
<alphaNum> ::= A | B | … | Z | a | b | … | _ | 0 | ... | 9
<mark>::= _</td></tr>
<tr><td>Eu.ModSt.537</td><td>Caching a value (except the value of a port): Mem_value-identifier (e.g. Mem_signal_aspect_to_be_indicated)</td></tr>
<tr><td>Eu.ModSt.541</td><td>Caching the value of a port: Mem_port-name (e.g. Mem_T6_Msg_defective)</td></tr>
<tr><td>Eu.ModSt.542</td><td><blockpropertyname> ::= <Mem><mark><port-name>
<Mem>::= Mem
<mark>::= _</td></tr>
<tr><td>Eu.ModSt.538</td><td><blockpropertyname> ::= <Mem><mark><propertyinformation>
<propertyinformation>::= <alphaNum><remaininginformation>
<remaininginformation> ::= „" | <alphaNum><remaininginformation>
<Mem>::= Mem
<alphaNum> ::= A | B | … | Z | a | b | … | _ | 0 | ... | 9
<mark>::= _</td></tr>
<tr><td>Eu.ModSt.7813</td><td>8.6.4 Model elements - Block operations</td></tr>
<tr><td>Eu.ModSt.7500</td><td>Block operations (2) are used in order to specify
• internal broadcast events or
• algorithms of data transformations defined in the operation body (call behaviour, time advance behaviour).</td></tr>
<tr><td>Eu.ModSt.7951</td><td>The content of an operation defined in the operation body shall always be displayed in the requirements management tool in "Requirements Part 1" and the name of the operation must be noted above it as a comment. The actual name of the operation, which comes from the model element, shall then be displayed in "Requirements Part 2".</td></tr>
<tr><td>Eu.ModSt.1011</td><td>8.6.4.1 Internal broadcast events</td></tr>
<tr><td>Eu.ModSt.545</td><td>Internal broadcast events are supposed to submit broadcasts within the state machine of a FE/TFE.</td></tr>
</table>

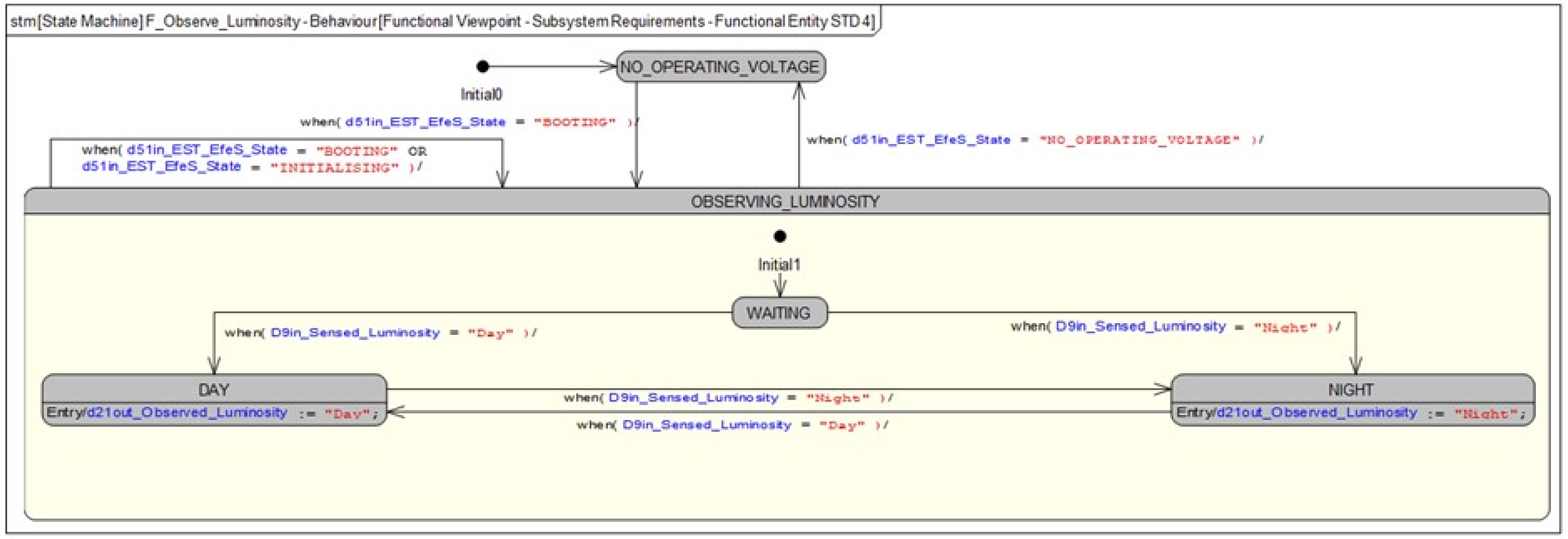
ID	Requirement
Eu.ModSt.550	**Naming of internal broadcast events** bc<Id>_<broadcast information>, Example: bc1_indicate_signal_aspect.
Eu.ModSt.969	**Id:** Natural number starting with 1
Eu.ModSt.548	**8.6.4.2 Definition of algorithms for data transformation**
Eu.ModSt.549	There are two types of behaviour that can be defined by means of SysML block operations: • **call behaviour** and • **time advance behaviour**.
Eu.ModSt.7823	**8.6.4.2.1 Call behaviour**
Eu.ModSt.7502	Block operations used to define call behaviour are prefixed with cOp<Id> where "Id" is a natural number starting with 1.
Eu.ModSt.7504	Call operations are used as • boolean expressions or parts of it in change events: e.g. when(cOp3_No_End_Position)/ • transition guards: e.g. when(cOp5_Trailed)[cOp7_Is_Trailable]/ • transition effects: e.g after(D5in_Con_tmax_Point_Operation/cOp12_Timeout();

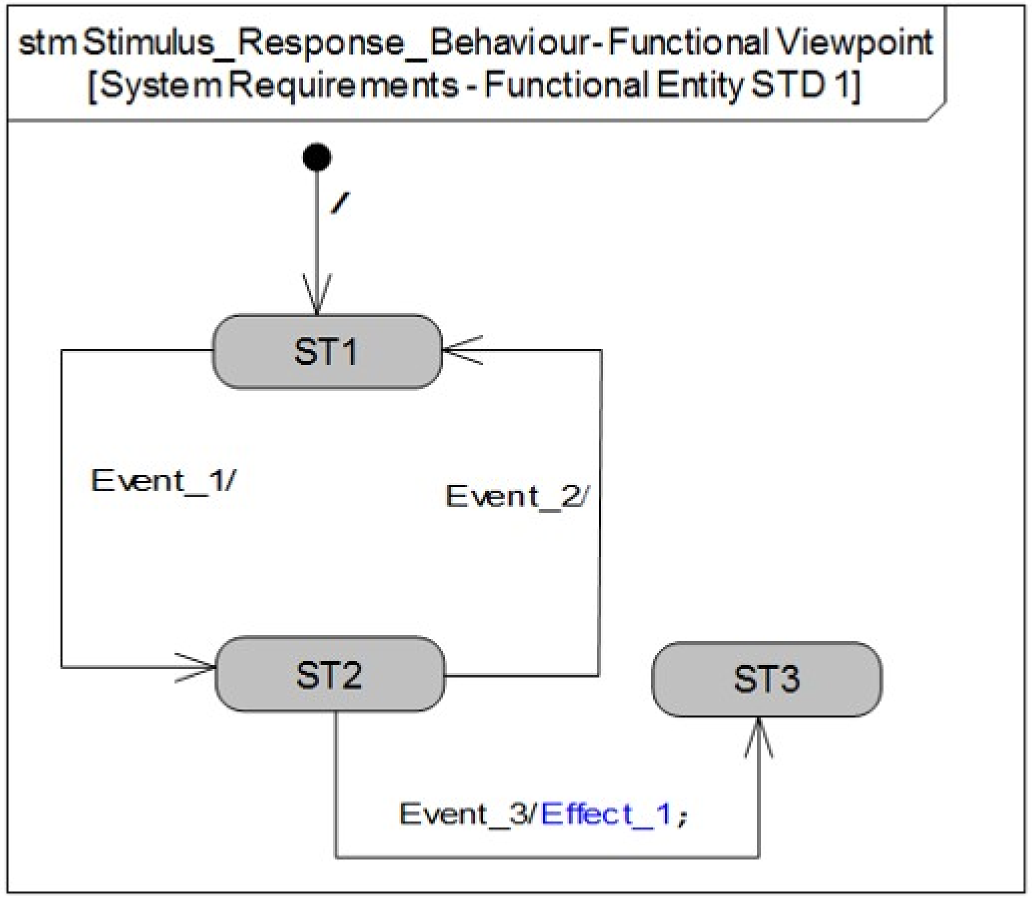
Eu.ModSt.7503	Call behaviour is invoked on demand, executed and terminated after execution. It is supposed to define event-driven data transformations. The algorithm of the data transformations is described in the body of the corresponding block operation using the Atego Structured Action Language (see *chapter 8.6.7*).
	
	Example: cOp2_All_Left
	if cOp8_Supports_Multiple_PMs() then
	return (
	(D21in_PM1_Position = "LEFT") and
	(D22in_PM2_Position = "LEFT" or D13in_PM2_Activation= "INACTIVE")
	);
	else
	return D21in_PM1_Position = "LEFT";
	end if

Eu.ModSt.7505	The call operation to initialise the block properties and Out Ports of a FE is named cOp1_init() systematically.
Eu.ModSt.7506	Call operations are to be interpreted as definitions. They become mandatory requirements (binding character "Req") when they are used in a mandatory requirement, such as a transition of a state.
Eu.ModSt.1014	**8.6.4.2.2 Time advance behaviour**
Eu.ModSt.1015	Time advance behaviour is invoked once during system activation and executes continuously. It is supposed to define continuous data transformation. The algorithm of the data transformations is to be described in the ***body*** of the corresponding block operation using the Atego Structured Action Language (see *chapter 8.6.8*).
Eu.ModSt.553	**Naming of time advance behaviour** tOp<Id>_<behaviour name> Example: tOp1_indicate_availability_ratio
Eu.ModSt.1017	**Id:** Natural number starting with 1
Eu.ModSt.7814	**8.6.5 Model elements - Ports**
Eu.ModSt.7507	**8.6.5.1 Atomic SysML in ports and out ports**
Eu.ModSt.7508	A FE features interfaces that define the stimuli consumed by the assigned state machine, represented by atomic in ports, and responses generated by the assigned state machine, represented by atomic out ports.
Eu.ModSt.7509	In ports and out ports are specified as SysML proxy ports or SysML flow ports of the SysML block representing the FE/TFE depicted in an internal block diagram (ibd).

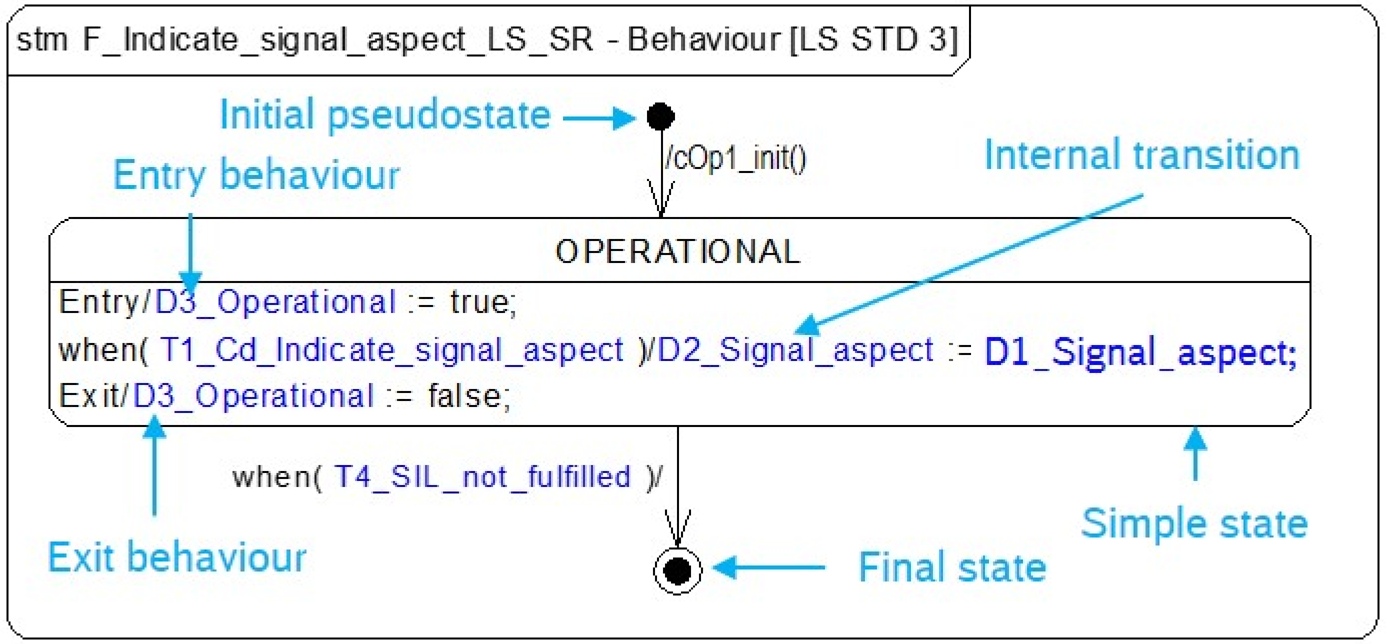
ID	Requirement
Eu.ModSt.7510	In ports and out ports are described according to the port definition schema below: *<Port information type><PNo><Port direction>_<Port information>:<Data type>*.
Eu.ModSt.7511	**Port information type** Used port information type: • **D** or **d**: data ports (D-Ports), • **T** or **t**: trigger ports (T-Ports).
Eu.ModSt.7512	Data ports and trigger ports start with a small letter (such as d3in_Point_Position or t4out_Timeout) if they are part of an internal connection between two FEs or between a FE and a TFE. In this case they are referred to as **functional ports** and have the colour green like the corresponding F E **(5)**.
Eu.ModSt.7513	**Data ports** and **trigger 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 or T1in_SIL_not_fulfiled). In this case they are referred to as **logical ports** and have the colour blue **(6)**.
Eu.ModSt.7514	Data ports and trigger ports which are part of a connection between TFEs or a TFE and the system environment (technical system interface) are referred to as **technical functional ports** and have the colour Yellow **(10)**. They start with a small letter if they are part of an internal connection between two TFEs and with a capital letter if they are part of an external connection between a TFE and the system environment (technical system interface).
Eu.ModSt.7515	**Data ports (5), (6)** Data ports are especially suited to indicate permanently available information. The value of a D-port only changes if it is explicitly changed.
Eu.ModSt.7516	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 (e.g configuration data: d21in_Con_Downgrade_Most_Restrict). Their values can be permanently regarded as valid.
Eu.ModSt.7517	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).
Eu.ModSt.7518	**Trigger ports (8)** Trigger ports are especially suited to indicate singular events. They have a Boolean value that always enters false and only briefly changes to true when the event occurs (data types PulsedIn or PulsedOut). Afterwards the value is automatically returned to false.
Eu.ModSt.7519	Trigger in ports are mainly used as arguments of Boolean expressions in change events.
Eu.ModSt.7520	**Port number (PNo)** For each port of a FE/TFE with the port information type "D or d" or "T or t", a unique PNo is to be assigned in the format of a natural number. The ports need not be numbered consecutively. For example port numbers like 1, 2, 3, 4, 5 are possible, but also 1, 3, 6.
Eu.ModSt.7521	**Port direction** The direction of the in Ports and out Ports are additionally defined, i.e. whether it is a stimulus or a response for the FE. • An "in" after the port number represents a stimulus or a permanently present value, • An "out" after the port number represents a response.
Eu.ModSt.7522	**Port information** The port information defines the information type and the semantic meaning of the information to be transmitted, e.g. "Cd_Indicate_signal_aspect". *<Port information> := <Information type> _<Information>*
Eu.ModSt.7523	**Information type:** Msg (message), Cd (command), Con (configuration data), Site (site data) or project-specifically determined information types.
Eu.ModSt.7524	**Information:** semantic meaning of the information to be transmitted, e.g. Indicate_signal_aspect.
Eu.ModSt.7525	**Data type** The data type which is assigned to any in port and out port is only shown on the diagram if it is necessary for a correct interpretation.
Eu.ModSt.7526	**Initialisation of out ports** All data out ports are initialised. The initialisation can be carried out in the body of the init-block operation systematically named cOp1_init(). Alternatively it can be carried out directly in the transition effect of the transition outgoing from initial state of the state machine. Trigger out ports are set to "FALSE" by default and are not explicitly initialised. Example: D25out_Redrive := FALSE; The assignments of values to the corresponding out ports are to be interpreted as definitions. They become mandatory requirements (binding character "Req") when they are used in a mandatory requirement, such as a transition of a state.

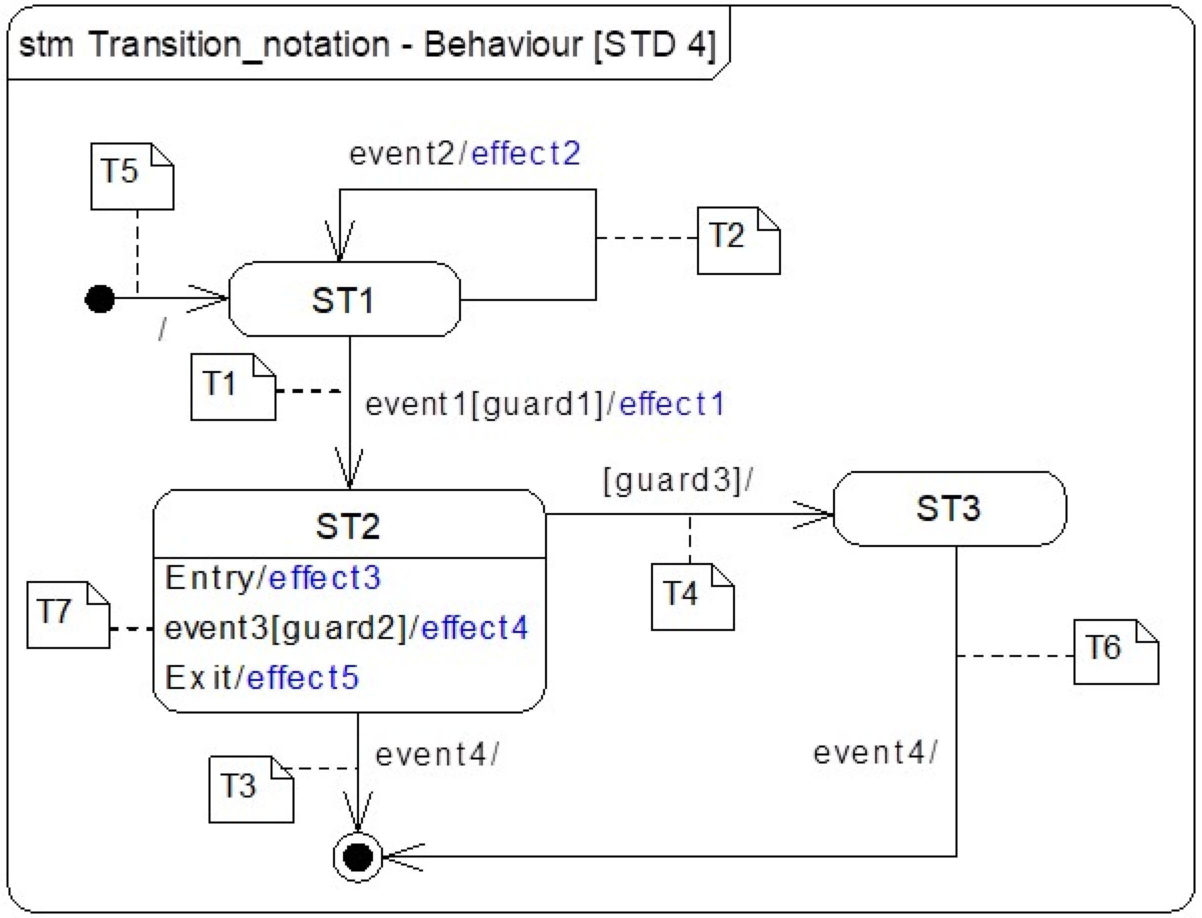
ID	Requirement
Eu.ModSt.7527	**8.6.5.2 SysML proxy ports to describe a signal-based communication**
Eu.ModSt.7528	A FE features interfaces that define event-driven in-flow of information consumed by the assigned state machine and event-driven out-flow of information generated by the assigned state machine.
Eu.ModSt.7529	The information flows are represented by **SysML proxy ports** typed with **SysML interface blocks (4, 7)**.
Eu.ModSt.7530	The information objects to be exchanged are represented by **signals**. The interface blocks define the **receptions** for these signals.
Eu.ModSt.7531	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.
Eu.ModSt.7824	Proxy ports to describe a signal-based information flow are described according to the port definition schema below: *<Port information type><PNo><Port direction>_<Port information>:<Signature of Interface block aggregating information objects>*.
Eu.ModSt.7825	**Port information type** Used port information type: **P** or **p**
Eu.ModSt.7532	Ports and their interface blocks are written in small letter (such as p1inout : ~cc_w) if they are part of an internal connection between two FEs. In this case they are referred to as **functional ports** and have the colour green like the corresponding FE **(4)**.
Eu.ModSt.7533	Ports and their interface blocks are written in capital letters if they are part of an external connection (system interface) between a FE and the system environment (such as P3inout : W_P) or if they are open ports. In this case they are referred to as **logical ports** and have the colour blue **(7)**.
Eu.ModSt.7534	Ports which are part of a connection between TFEs or a TFE and the system environment (technical system interface) are referred to as **technical ports** and have the colour yellow **(10)**. They start with a small letter if they are part of an internal connection between two TFEs and with a capital letter if they are part of an external connection between a TFE and the system environment (technical system interface) or if they are open ports.
Eu.ModSt.7535	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: p1inout : ~cc_w).
Eu.ModSt.7826	**Port number (PNo)** For each port of a FE/TFE with the port information type "P or p", a unique PNo is to be assigned in the format of a natural number. The ports need not be numbered consecutively. For example port numbers like 1, 2, 3, 4, 5 are possible, but also 1, 3, 6.
Eu.ModSt.7827	**Port direction** The direction of the ports are additionally defined ("in", "out", "inout").
Eu.ModSt.7828	**Port information** Freely selectable and optional.
Eu.ModSt.7536	**Signature of Interface block aggregating information objects** The information flow through a proxy port is represented by an interface block in which the receptions for the incoming and outgoing information objects are defined. The information objects are represented by signals. The use of interface blocks and signals is described in the *chapters 8.4.7* (Model view "Information Flow"), *8.6.6.9.4* (Signal event) and *8.6.6.10.1* (Event-driven responses using signals).
Eu.ModSt.7565	**8.6.6 Model elements - state machines**
Eu.ModSt.7566	In the following, the term "Functional Entity" and the corresponding abbreviation "FE" stand for both a FE and a TFE.
Eu.ModSt.7567	A FE is always in a state that abstracts a combination of values given in the FE. Events arriving at the FE lead to reactions - depending on the state - that change values of SysML out ports or SysML block properties, invoke a **l**ocal trigger or a call operation or send a signal via a port and result in new states.
Eu.ModSt.7568	The state machine diagrams (see *figure 7569*) are children of the state machine and illustrate its behaviour, i.e. they describe the stimulus-response behaviour of a FE. The state machine contains states and state transitions that are triggered by trigger in ports, data in ports, internal broadcast events, signal events as well as timing events. The state transitions represent the binding functional requirements of the system to be specified.
Eu.ModSt.7830	**State Machine Diagram (STD):** defines the behaviour of a FE.
Eu.ModSt.7934	For each STD, a description must be inserted in the modelling tool (e.g. Properties ->Text->Description) that corresponds to a defined schema: • The SUS or SIUS receives a stimulus and responds with the result to....

ID	Requirement
Eu.ModSt.7935	A possible application of the schema is shown below using the example of the subsystem LS: Information: This state machine diagram describes the requirements for the following functionalities: • receives the observed Signal Aspect and reports this to the Subsystem - Electronic Interlocking • receives the observed intentionally dark state and reports this to the Subsystem - Electronic Interlocking • receives the observed Luminosity and reports this to the Subsystem - Electronic Interlocking
Eu.ModSt.7936	The description is to be transferred to "Requirements Part 2" of the specification document generated in the requirements management tool.
Eu.ModSt.7832	**Diagram heading:** *stm[State Machine]<><FE_TFE block signature><>[Functional Viewpoint<>-<>Subsystem Requirements or Interface Requirements<>-<>Functional Entity or Technical Functional Entity<>STD<DiaNo>*]
Eu.ModSt.1128	**<DiaNo> :=** Natural number starting with 1
Eu.ModSt.7569	Figure 7569 Example of a state machine diagram 
Eu.ModSt.7570	**8.6.6.1 Region**
Eu.ModSt.7571	Each state machine contains at least one region, which itself can contain a number of states and pseudostates, as well as the transitions between them. During execution of a state machine, each of its regions has a single active state that determines the transitions that are currently viable in that region. A region must have an initial pseudostate and can have a final state that correspond to its beginning and completion, respectively.
Eu.ModSt.7572	If a state machine contains a single region, it is represented by the area inside the frame of the state machine diagram and it is not to be named. Multiple regions are named and shown separated by dashed lines. A state machine with multiple regions may describe some concurrent behaviour happening within the state machine's owning block.
Eu.ModSt.7573	**8.6.6.2 State**
Eu.ModSt.7574	The UML specification defines a state as „a situation during which some (usually implicit) invariant condition holds. The invariant may represent a static situation such as an object waiting for some external or internal event to occur". The „object", in the present case the FE, is waiting for a stimulus from its environment or for an internal stimulus such as a time event or a local trigger.
Eu.ModSt.7575	Thus, a state represents a "between stimuli" condition of the external observable stimulus-response behaviour of a FE. In other words, it specifies the responses to incoming stimuli.
Eu.ModSt.7576	It is helpful to use the analogy that a block, i.e. the FE, is controlled by a switch. Each state corresponds to a switch position. The state machine defines all valid switch positions (i.e. states) and transitions between switch positions (i.e. state transitions). If there are multiple regions, each region is controlled by its own switch with its switch positions corresponding to its states. The switch positions can be specified by a form of truth table - similar to how logic gates can be specified - in which the current states and transitions define the next state.
Eu.ModSt.7577	In the example depicted in *Figure 427*, the state ST2 represents a "between stimuli condition", i.e. it constitutes the precondition for triggering a response in the form of Effect_1. Following the analogy that the FE is controlled by a switch, the switch would be positioned to ST2. When Event_3 occurs Effect_1 is executed while the FE changes to state ST3.

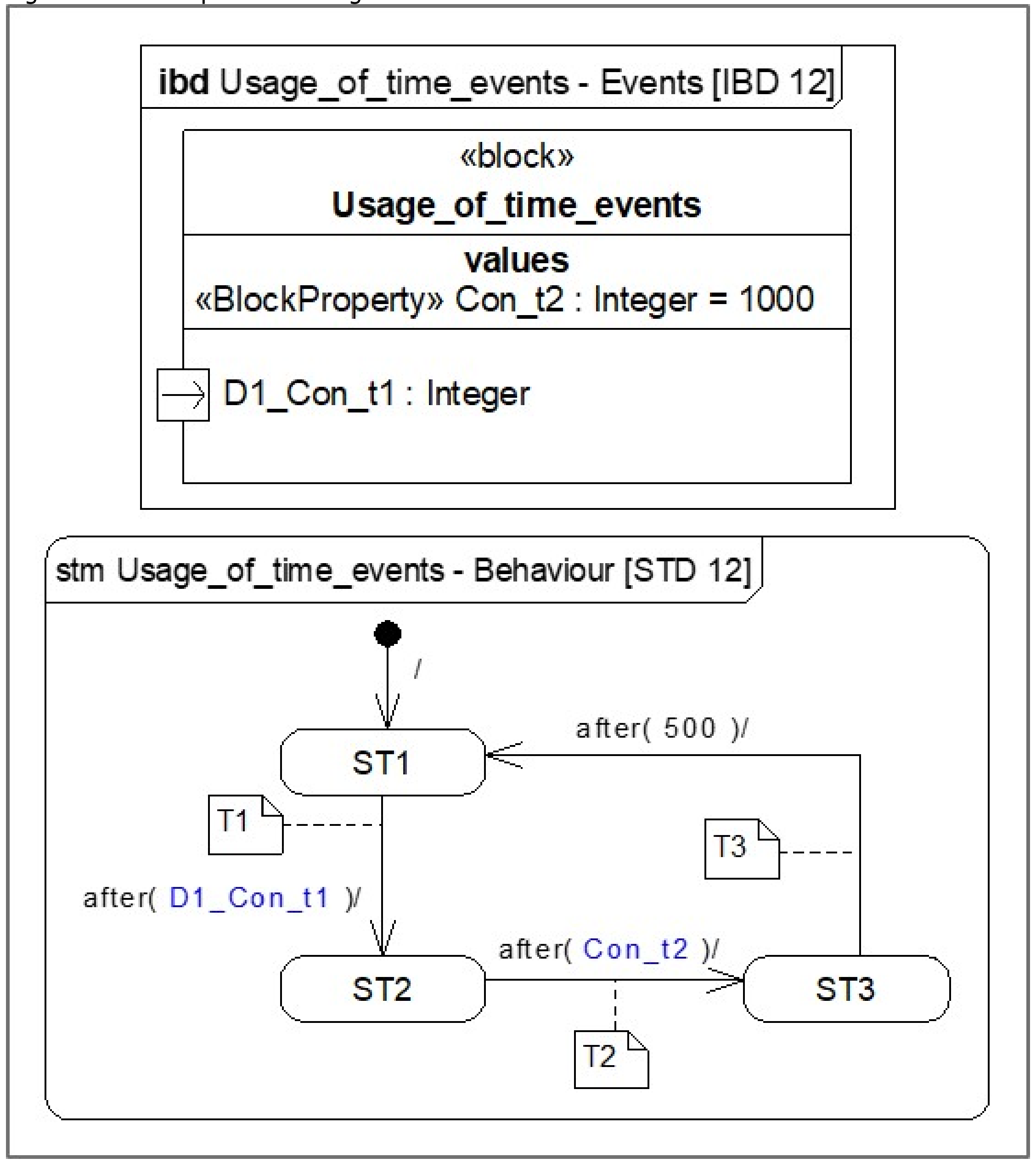
ID	Requirement
Eu.ModSt.7578	Figure 427 Example of a state specifying a response 
Eu.ModSt.7579	In the EULYNX requirements specification documents there are below the depicted state machine diagrams (as for example depicted in *figure 33*) the corresponding state transitions listed as atomic mandatory functional requirements: **Info | Initial** **Req | {Initial - ST1}** **Info | ST1** **Req | Event_1/{ST1 -ST2}** **Info | ST2** **Req | Event_2/{ST2 -ST1}** **Req | Event_3/Effect_1; {ST2 - ST3}** **Info | ST3**
Eu.ModSt.7580	A state is represented on the state machine diagram by a round-cornered box containing its name.
Eu.ModSt.7581	**Kinds of states:** The following three kinds of states are distinguished: • **simple state** (state with no regions and therefore without nested states), • **sequential state** (state with exactly one region) and • **concurrent state** (state with at least two regions)
Eu.ModSt.7582	Each state may contain entry and exit behaviour that are performed whenever the state is entered or exited respectively. Entry and exit behaviour are described as text expressions using the chosen action language preceded by the keywords entry or exit and a forward slash.
Eu.ModSt.7583	A state machine can contain transitions, called internal transitions, which do not effect a change in state. An internal transition has the same source and destination and, if triggered, simply executes the transition effect.
Eu.ModSt.7584	By contrast, an external transition with the same source and destination state - sometimes called a transition-to-self - triggers the execution of that state's exit and entry behaviour as well as the transition effect.

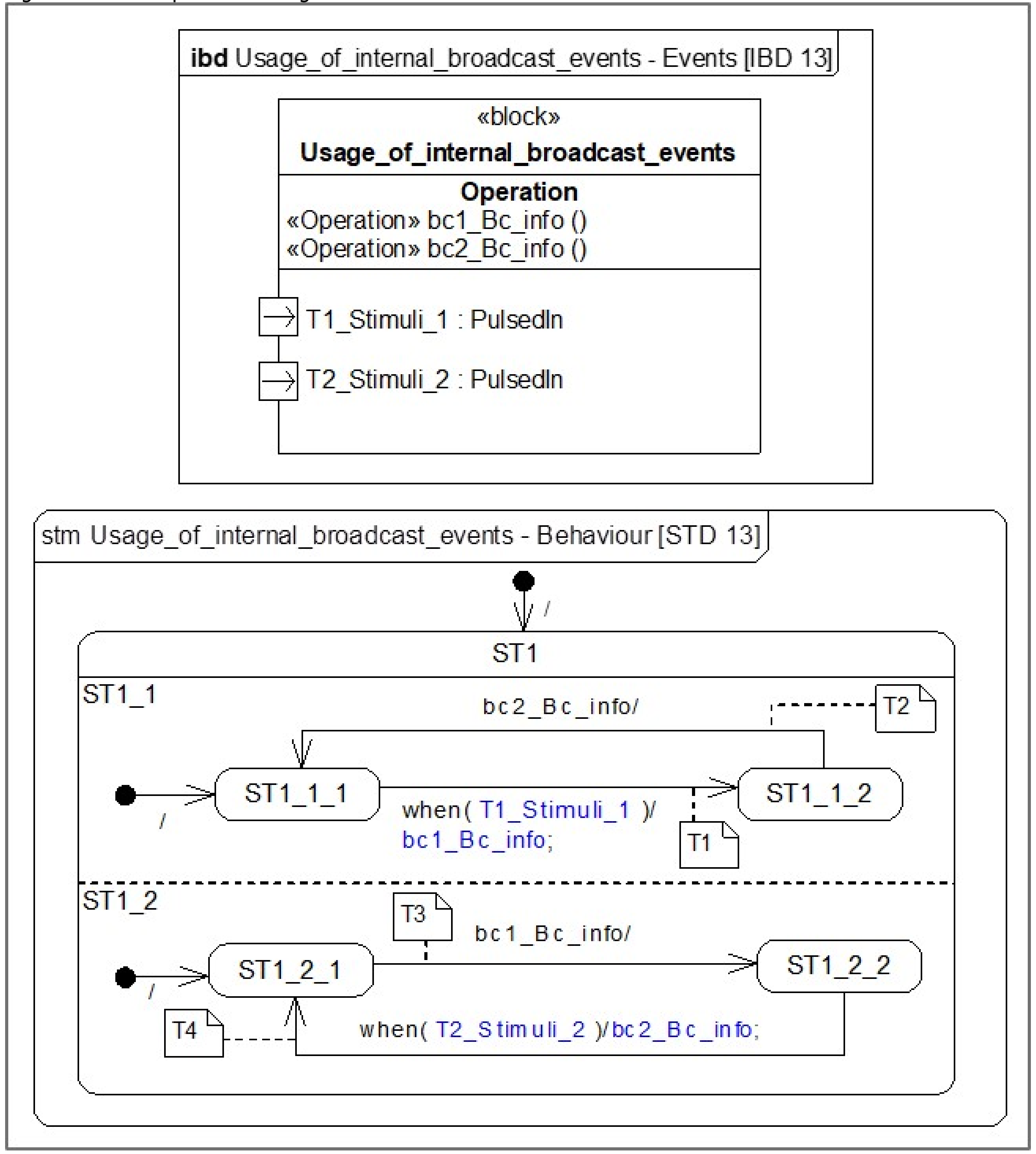
ID	Requirement
Eu.ModSt.7585	Additional to the states, SysML includes a number of pseudostates to provide additional semantics. The difference between a state and a pseudostate is that a region can never stay in a pseudostate, which merely exists to help determine the next active state.
Eu.ModSt.7586	The EULYNX methodology adopts the following SysML pseudostates: • **initial pseudostate**, • **final state**, • **choice pseudostate**, • **fork pseudostate** and • **join pseudostate**.
Eu.ModSt.7587	Pseudostates have a defined name, that may be visible on the diagrams.
Eu.ModSt.7588	**8.6.6.3 Initial pseudostate and final state**
Eu.ModSt.7589	An initial pseudostate is shown as a filled circle. It is used to determine the initial state of a region (see *Figure 7609*). The outgoing transition from an initial pseudostate may include an effect. Such effects are often used to set the initial values of properties used by the state machine (e.g. call operation cOp1_init() shown in *Figure 7609*).
Eu.ModSt.7590	A final state is shown as a bulls-eye (i.e. a filled circle surrounded by a larger hollow circle). It indicates that a region has completed execution. When the active state of a region is the final state, the region has completed, and no more transitions take place within it. Hence, a final state can have no outgoing transitions.
Eu.ModSt.7591	**8.6.6.4 Choice pseudostate**
Eu.ModSt.7592	A choice pseudostate is shown as a white diamond with one transition arriving and two or more transitions leaving. It is used to construct a compound transition path between states. The compound transition allows more than one alternative path between states to be specified, although only one path can be taken in response to any single event.
Eu.ModSt.7593	Multiple transitions may either converge on or diverge from the choice pseudostate. When there are multiple outgoing transitions from a choice pseudostate, the selected transition will be one of those whose guard evaluates to true at the time after the choice pseudostate has been reached. This allows effects executed on the prior transition to affect the outcome of the choice.
Eu.ModSt.7594	When a choice pseudostate is reached in the execution of a state machine, there must always be at least one valid outgoing transition. If not, the state machine is invalid.
Eu.ModSt.7595	If a compound transition contains choice pseudostates, any possible compound transition must contain only one trigger, normally on the first transition in the path.
Eu.ModSt.7596	**8.6.6.5 Fork pseudostate**
Eu.ModSt.7597	A fork pseudostate is shown as a vertical or horizontal bar with transition edges either starting or ending on the bar.
Eu.ModSt.7598	It has a single incoming transition and as many outgoing transitions as there are orthogonal regions in the target state. Unlike choice pseudostates, all outgoing transitions of a fork are part of the compound transition. When an incoming transition is taken to the fork pseudostate, all the outgoing transitions are taken.
Eu.ModSt.7599	Because all outgoing transitions of the fork pseudostate have to be taken, they may not have triggers or guards but may have effects.
Eu.ModSt.7600	**8.6.6.6 Join pseudostate**
Eu.ModSt.7601	A join pseudostate is shown as a vertical or horizontal bar with transition edges either starting or ending on the bar.
Eu.ModSt.7602	The coordination of outgoing transitions from a concurrent state is performed using a join pseudostate that has multiple incoming transitions and one outgoing transition. The rules on triggers and guards for join pseudostates are the opposite of those for fork pseudostates.
Eu.ModSt.7603	Incoming transitions of the join pseudostate may not have triggers or a guard but may have an effect. The outgoing transition may have triggers, a guard and an effect.
Eu.ModSt.7604	When all the incoming transitions can be taken and the join's outgoing transition is valid, the compound transition can occur. Incoming transitions occur first followed by the outgoing transition.
Eu.ModSt.7605	**8.6.6.7 Simple state**
Eu.ModSt.7606	As shown in the examples depicted in *Figure 427* (states ST1, ST2, ST3) and *Figure 7609* (state "OPERATIONAL"), a simple state has no regions and therefore no nested states.
Eu.ModSt.7607	A simple state may, like any kind of state, contain entry behaviour, that is executed immediately upon entering the state, exit behaviour, that is executed immediately before exiting the state, and behaviour executed during internal transitions. (see *Figure 7609*). All three kinds of behaviour are not interruptible.

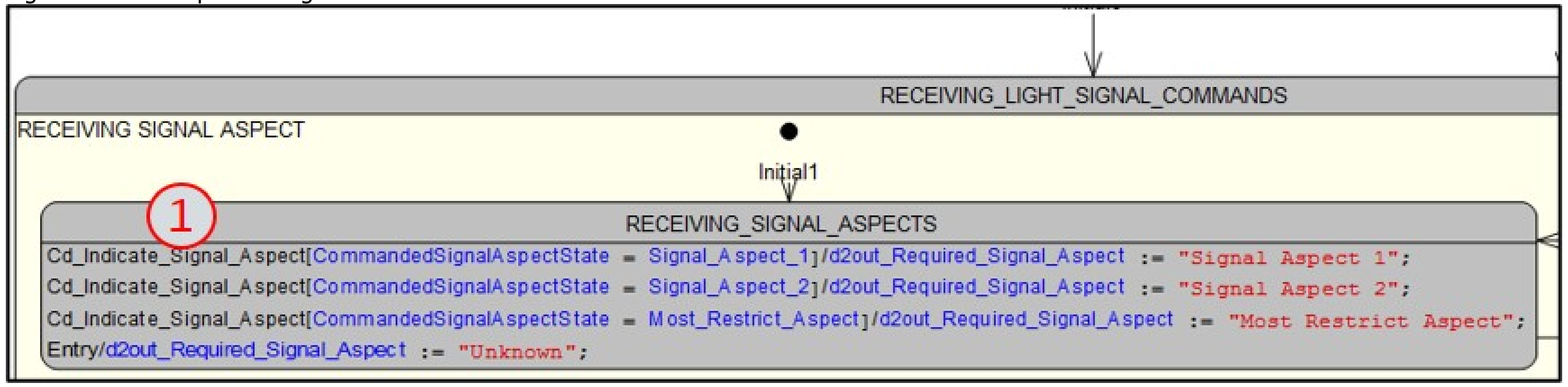
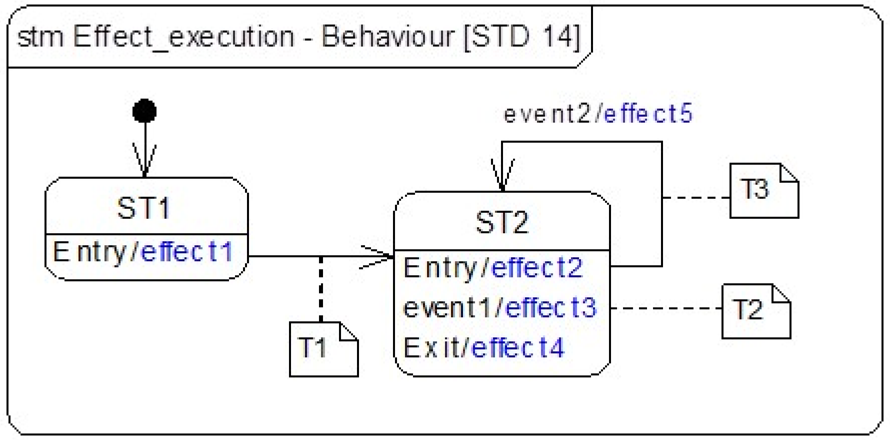
ID	Requirement
Eu.ModSt.7608	*Figure 34* shows a simple example of a FE defining the functionality "Indicate signal aspect" of a light signal (LS) with a single OPERATIONAL state in its single region. A transition from the region's initial pseudostate goes to the OPERATIONAL state. On entry, the light signal indicates that it is operational, setting the value of the out port "D3_Operational" to true, and on exit it indicates a non operational status, setting the value of "D3_Operational" to false. While the light signal is in the state OPERATIONAL, it may receive commands to indicate a transmitted signal aspect (T1_Cd_Indicate_signal_aspect) and indicate it (D2_Signal_aspect). When in the OPERATIONAL state, the internal trigger "T4_SIL_not_fulfiled" triggers a transition to the final state, and because there is only one single region, the state machine terminates.
Eu.ModSt.7609	Figure 7609 Example of a simple state 
Eu.ModSt.7610	**8.6.6.8 Transition**
Eu.ModSt.7611	A transition specifies a change of state within a state machine. It is a directed relationship between a source and a destination state, and defines an event (trigger) and a guard (condition) that both lead to the state transition, as well as an effect (behaviour) that is executed during the transition. Source and destination can be the same state (see T2 in *Figure 7626*).
Eu.ModSt.7612	**Run to completion:** State machines always run to completion, which means that they are not able to consume another event until the state machine has completed the processing of the current event. Thus, the next event will be consumed only if all effects (behaviour) of the previous event have been completed.
Eu.ModSt.7613	Run to completion does not mean that a state machine owned by a FE interconnected with neighbouring FE monopolises all FEs in this network until the run to completion step is complete. The preemption restriction only applies to the context of the corresponding FE.
Eu.ModSt.7614	An event that cannot be consumed, for example because there is no matching transition, is discarded.
Eu.ModSt.7615	**Transition notation:** A transition is shown as an arrow between two states, with the head pointing to the target state.
Eu.ModSt.7616	Transitions-to-self are shown with both ends of the arrow attached to the same state (see T2 in *Figure 7626*).
Eu.ModSt.7617	Internal transitions are not shown as graphical paths but are listed on separate lines within the state symbol (see T7 in *Figure 7626*).
Eu.ModSt.7618	The definition of the transition's behaviour is shown in a formatted string on the transition with the event first, followed by a guard in square brackets, and finally the transition effect preceded by a forward slash (event-effect block or even-action block). As shown in *Figure 7626*, any or all of the behavioural elements as event, guard and effect may be omitted. In T5 for example, all the behavioural elements are omitted. Transition **T3**, to give another example, is only triggered by an event without guard and effect.
Eu.ModSt.7619	**Event:** An event specifies some occurrence that can be measured with regard to location and time and causes a transition to occur. Descriptions of the triggering events are provided in *chapter 8.6.6.9 Event*.
Eu.ModSt.7620	**Guard:** The transition guard contains an expression that must evaluate true in the moment of the triggering event so that the transition is performed (see T1, T4 and T7 in *figure 35*). The guard is specified using a constraint which includes an expression formulated in the applied action language to represent the guard condition. If preceded by an event (see T1 and T7 in *Figure 7626*) and if the event satisfies a trigger, the guard on the transition is evaluated. If the guard evaluates to true, the transition is triggered; if the guard evaluates to false, then the event is consumed with no effect.

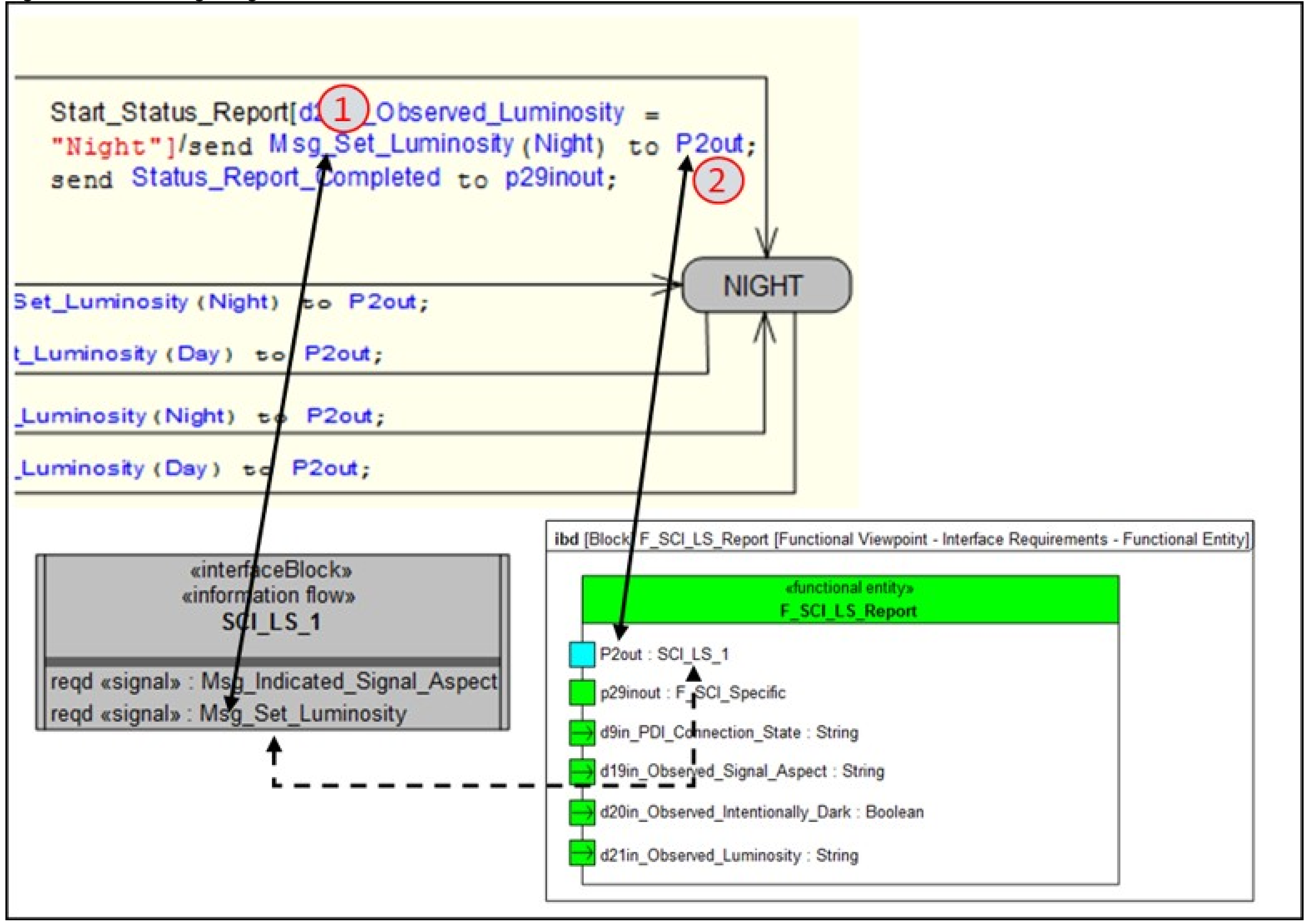
ID	Requirement
Eu.ModSt.7621	Transitions can also be triggered by internally generated completion events. For a simple state a completion event is generated when the entry behaviour (for example Entry/effect3 in *Figure 7626*) has completed.
Eu.ModSt.7622	Thus, where a guard is shown without a preceding event (see T4 in *Figure 7626*), the guard condition is evaluated immediately after entering the source state, i.e. after its entry behaviour has completed, and a transition takes place if true, triggered by the generated completion event of the source state.
Eu.ModSt.7623	**Please note:** if the guard condition of a transition without trigger changes to true while the state machine is already in the source state (for example in state ST2), the guard condition won't be evaluated and no transition will take place.
Eu.ModSt.7624	**Effect:** The effect is a behaviour executed when entering or exiting a state (entry and exit behaviour, respectively), during an internal transition (see T7 in *Figure 7626*) and during the external transition from one state to another (see T1 in *Figure 7626*). If an external transition is triggered, first the exit behaviour of the current (source) state, then the transition effect and finally the entry behaviour of the target state are executed. Descriptions of the effects used in the methodology underlying this Modelling standard are provided in chapter *8.6.6.10 Effect*.
Eu.ModSt.7625	A transition may also be formulated textually as atomic functional requirement: **Event [Guard]/Effect {Source state - Target state}**.
Eu.ModSt.7626	Figure 7626 Transition notation 
Eu.ModSt.7627	**8.6.6.9 Event**
Eu.ModSt.7628	An event specifies some occurrence that can be measured with regard to location and time and causes a transition to occur.
Eu.ModSt.7629	In the EULYNX methodology, the following types of events are used: • **Change event**, • **Time event** • **Internal broadcast event** • **Signal event**.
Eu.ModSt.7630	**8.6.6.9.1 Change event**

ID	Requirement
Eu.ModSt.7631	A change event indicates that some condition has been satisfied, that is, the value of a specified Boolean expression holds. A defined change event occurs during system operation each time the specified Boolean expression toggles from false to true. Change events are continuously evaluated.
Eu.ModSt.7632	According to the EULYNX methodology, the Boolean expression of a change event may contain the following arguments: • **Data In Port,** • **block property** • **block operation.**
Eu.ModSt.7633	**Notation of change events:** Change events use the term „when“ followed by the Boolean expression that has to be met in parenthesis. Like other constraint expressions, the Boolean expression is to be expressed in text using the applied action language: **when(boolean expression)[guard]/effect;**
Eu.ModSt.7634	**8.6.6.9.2 Time event**
Eu.ModSt.7635	A time event indicates that a given time interval has passed since the current state was entered.
Eu.ModSt.7636	**Notation of time events:** Time events use the term "after“ followed by the time period (in milliseconds by default) in parenthesis, e.g. after(D1_Con_t1) as depicted in *Figure 7638.*
Eu.ModSt.7637	"after" indicates that the time is relative to the moment the state is entered. The transition T1 shown in *Figure 7638* is, for example, triggered after the time D1_Con_t1 has expired. The time starts on entering the state ST1.

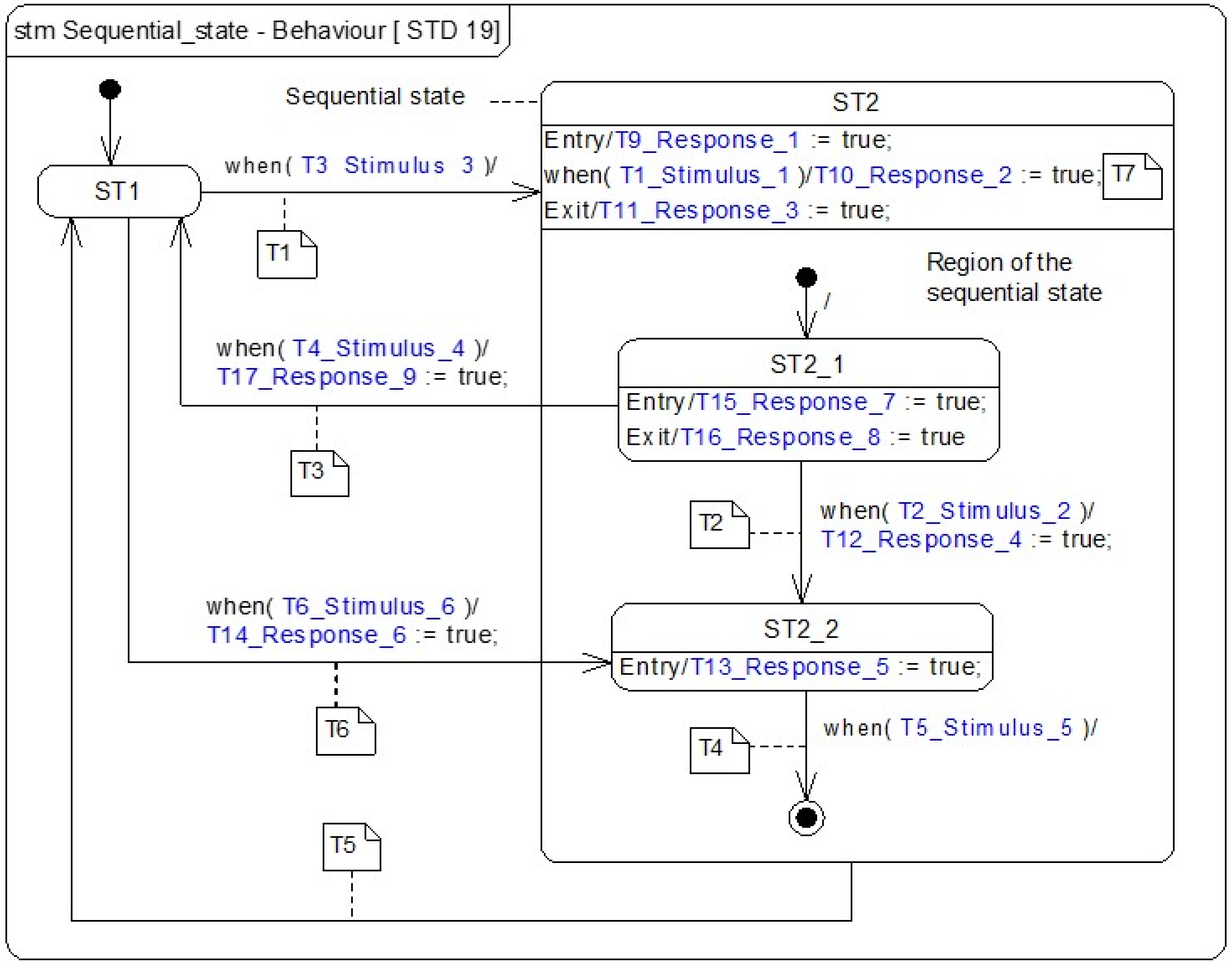
ID	Requirement
Eu.ModSt.7638	Figure 7638 Example of the usage of time events 
Eu.ModSt.7639	**8.6.6.9.3 Internal broadcast event**
Eu.ModSt.7640	Internal broadcast events occur when corresponding SysML block operations are invoked. They are supposed to submit broadcasts within the state machine of a FE.
Eu.ModSt.7641	In *Figure 7642* for example, the SysML block operations bc1_Bc_info() and bc2_Bc_info() represent internal broadcast events. During transition T1, the internal broadcast event bc1_Bc_info() is invoked in order to trigger transition T3. Furthermore, during transition T4, the internal broadcast event bc2_Bc_info() is invoked to trigger transition T2.

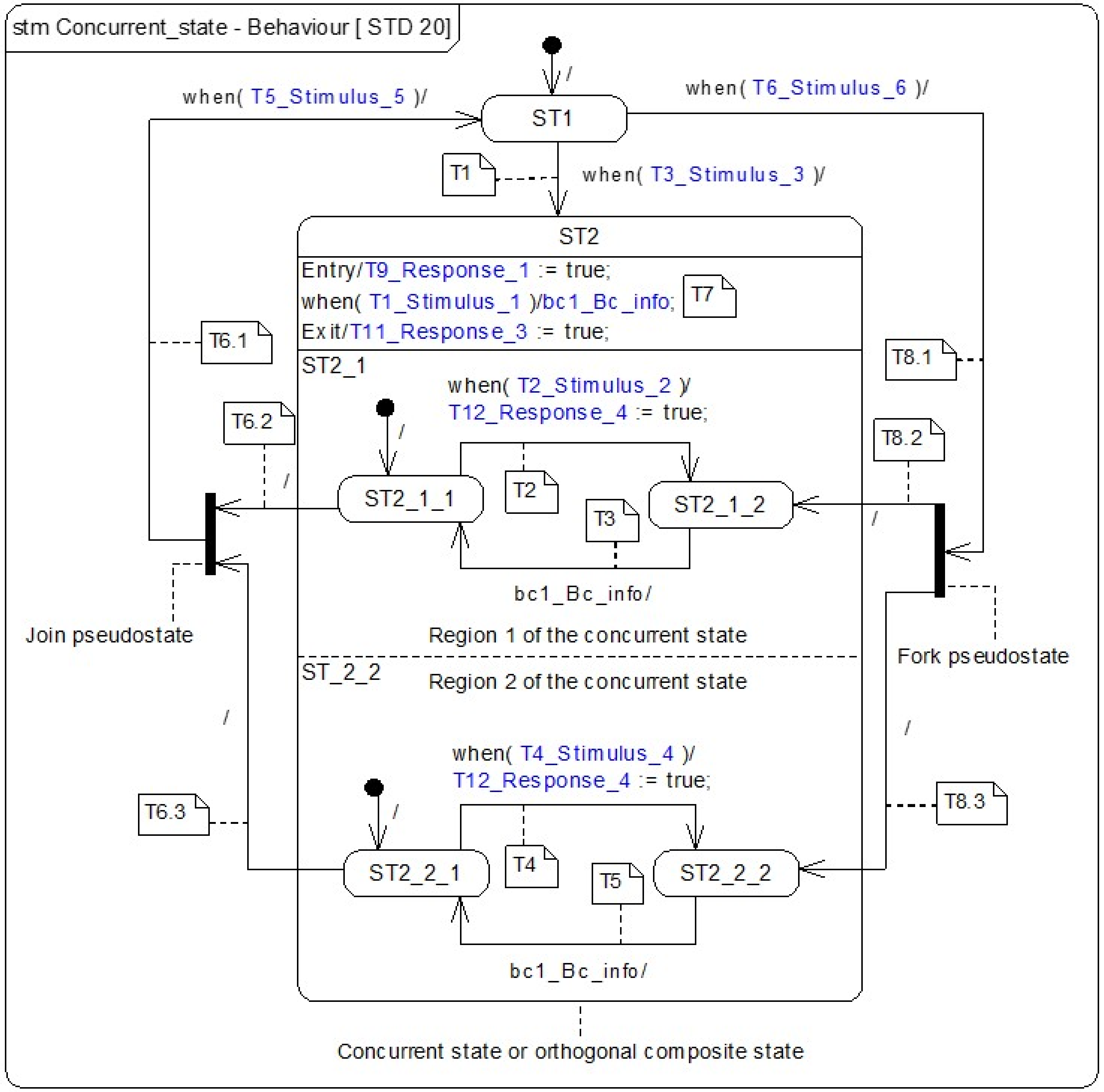
ID	Requirement
Eu.ModSt.7642	Figure 7642 Example of the usage of internal broadcast events 
Eu.ModSt.7643	**8.6.6.9.4 Signal event**
Eu.ModSt.7644	A signal event is generated when a reception of an interface block receives a signal. This is then used in the state model to trigger a state transition **(1)**.

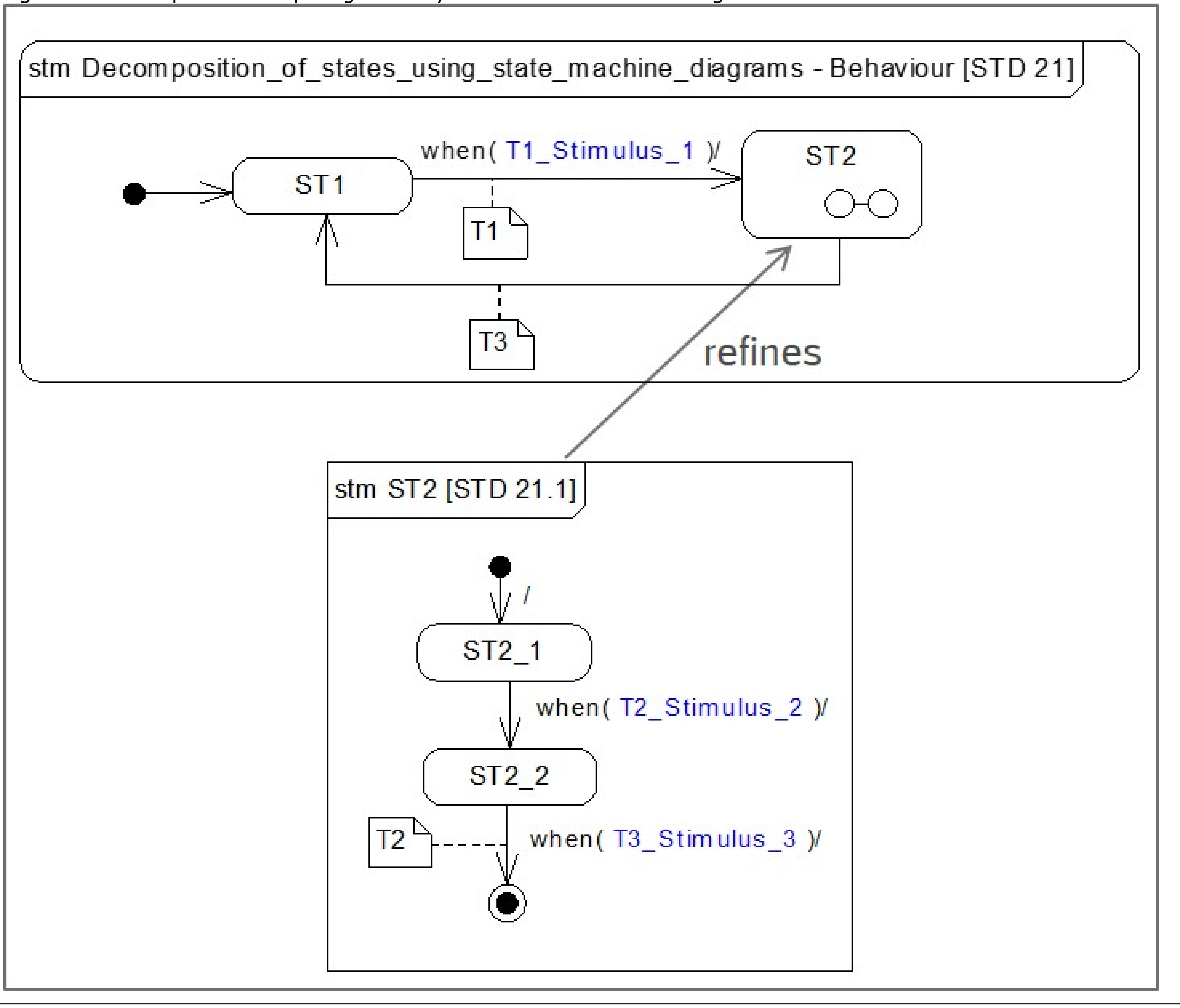
ID	Requirement
Eu.ModSt.7645	Figure 7645 Example of a signal event 
Eu.ModSt.7646	**8.6.6.10 Effect**
Eu.ModSt.7647	An effect is a behaviour executed when entering or exiting a state (entry and exit behaviour, respectively), during an internal transition or during an external transition from one state to another.
Eu.ModSt.7648	The sequence of effect execution is demonstrated in *figure 7649*. Transition T1 is taken immediately on completion of effect1. The sequence of effect execution when event2 occurs (T3) is: effect4, then effect5, then effect2. Event1 generates only one effect (T2): effect3.
Eu.ModSt.7649	Figure 7649 Sequence of effect execution 
Eu.ModSt.7650	The following elements of behaviour may be represented as **effect**: • **Event-driven responses using signals,** • **Responses in form of continuous flows,** • **Call behaviour.**
Eu.ModSt.7651	**8.6.6.10.1 Event-driven responses using signals**
Eu.ModSt.7652	As shown in *Figure 7652*, signals **(1)** are sent as an effect of a state transition or triggered in a block operation via the corresponding port **(2)** of the respective FE.

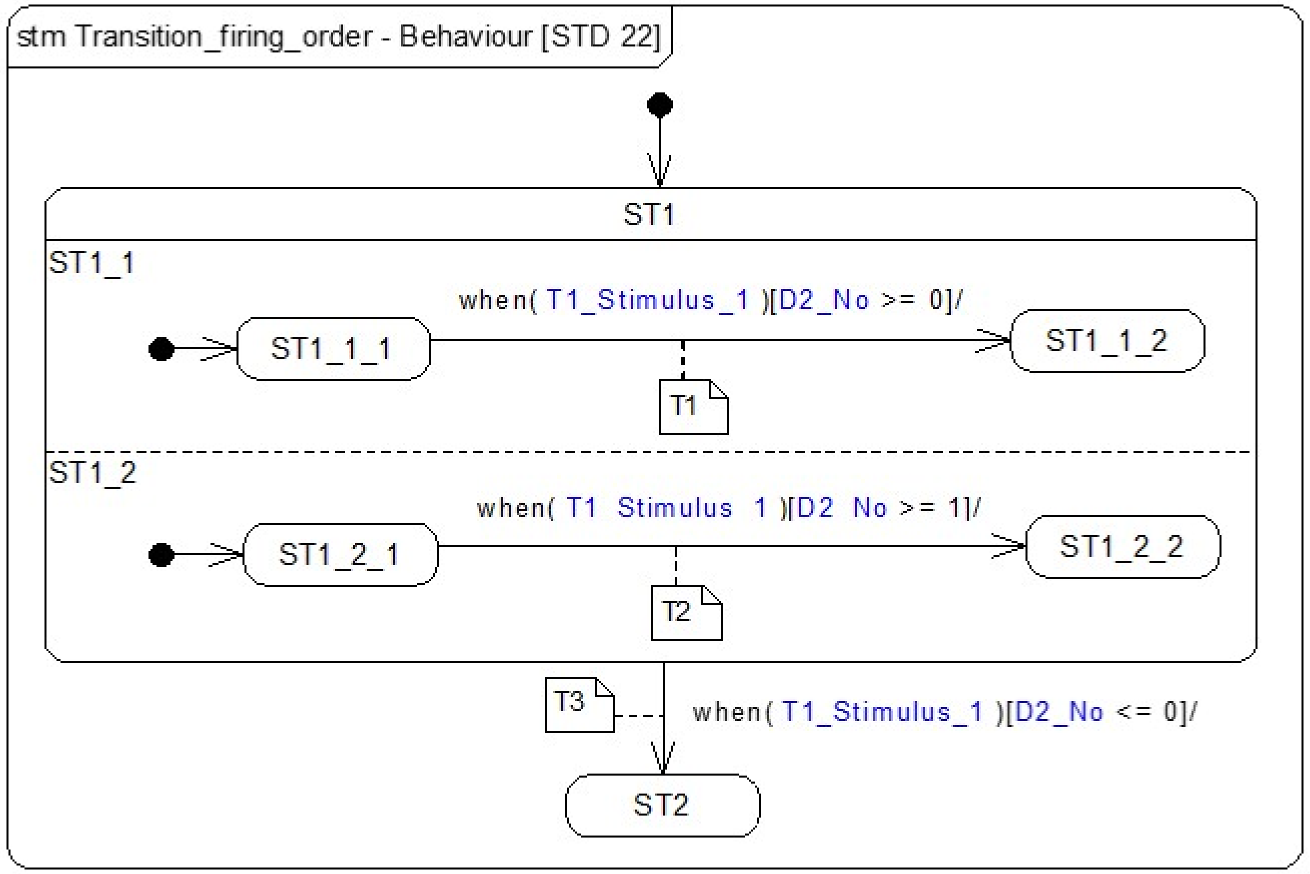
ID	Requirement
Eu.ModSt.7653	Figure 7653 Sending a signal 
Eu.ModSt.7654	**8.6.6.10.2 Responses in form of continuous flows**
Eu.ModSt.7655	A response is sent in form of a continuous flow by assigning the desired value to a data out port, e.g. D1out_Temperature := 40.
Eu.ModSt.7656	All out ports are initialised. The initialisation can be carried out in the body of the init-block operation systematically named cOp1_init(). Alternatively it can be carried out directly in the transition effect of the transition outgoing from initial state of the state machine.
Eu.ModSt.7657	Furthermore, the sender of a response must always configure the current value of the Data Out Port.
Eu.ModSt.7658	**8.6.6.10.3 Call behaviour**
Eu.ModSt.1013	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 to be described in the body of the corresponding block operation using ASAL (see *chapter 8.6.8*).
Eu.ModSt.551	**Naming of Call behaviour** cOp<Id>_<behaviour name>, Example: cOp2_establish_safe_state
Eu.ModSt.1016	**Id:** Natural number starting with 1
Eu.ModSt.552	The call behaviour to initialise the block properties and out ports of a FE is to be named **cOp1_init()** systematically.
Eu.ModSt.7660	**8.6.6.11 Composite state**

ID	Requirement
Eu.ModSt.7661	States can have regions. Such states are called composite states or hierarchical states. They allow state machines to scale to represent state-based behaviour of any complexity. A composite state may have one single region (sequential state) but also multiple orthogonal regions (concurrent state or orthogonal composite state).
Eu.ModSt.7662	Instead of using a region to decompose the behaviour of a state, a state machine diagram may be assigned to the corresponding state alternatively, defining its behaviour.
Eu.ModSt.7663	Each region or state machine diagram assigned to a state has a set of mutually exclusive disjoint subvertices and a set of transitions. In other words, it typically will contain an initial pseudostate and a final state, a set of pseudostates, and a set of substates, which may themselves be composite states.
Eu.ModSt.7664	Any state enclosed within a region of a composite state is called a substate of that composite state.
Eu.ModSt.7665	**8.6.6.12 Sequential state**
Eu.ModSt.7666	A sequential state, such as ST2 shown in the example depicted in *Figure 7674,* is a composite state that has one region.
Eu.ModSt.7667	*Figure 7674* shows the decomposition of the state ST2 into the substates ST2_1 and ST2_2. On entry to the state ST2, two entry behaviours are executed: the entry behaviour of ST2, T9_Response_1 := true and then the entry behaviour of ST2_1, T15_Response_7 := true. This is because on entry, as indicated by the initial pseudostate, the initial substate of ST2 is ST2_1.
Eu.ModSt.7668	When in state ST2_1, T2_Stimulus_2 will cause the transition T2 to the state ST2_2 and will successively process T16_Response_8 := true, T12_Response_4 := true and T13_Response_5 := true. If T5_Stimulus_5 is received while in state ST2_2, the change event will trigger the transition T4 to the final state. A completion event is generated when the final state is reached, triggering the transition T5 to state ST1. When leaving ST2, T11_Response_ 3 := true is executed.
Eu.ModSt.7669	A composite state (sequential state or concurrent state) may be porous, which means transitions such as transition T3 and T6 shown in *Figure 7674* may cross the state boundary, starting or ending on states within its regions.
Eu.ModSt.7670	In the case of a transition ending on a nested state, such as transition T6 shown in *Figure 7674*, the behaviours are executed in this order: **1.** the effect T14_Response_6 := true of the transition T6, **2.** the entry behaviour T9_Response_1 := true of the composite state, **3.** the entry behaviour T13_Response_5 := true of the transition's target nested state.
Eu.ModSt.7671	In the opposite case, such as transition T3 shown in *Figure 7674*, the behaviours are exited in this order: **1.** the exit behaviour T16_Response_8 := true of the source nested state, **2.** the exit behaviour of the composite state T11_Response_3 := true is executed, **3.** the transition effect T17_Response_9 := true.
Eu.ModSt.7672	In the case of more deeply nested state hierarchies, the same rule can be applied recursively to all the composite states whose boundaries have been crossed.
Eu.ModSt.7673	If T1_Stimulus_1 is received while in state ST2, the change event will trigger the internal transition T7 and the effect T10_Response_2 := true will be executed without a change of state.

ID	Requirement
Eu.ModSt.7674	Figure 7674 Example of a sequential state 
Eu.ModSt.7675	**8.6.6.13 Concurrent state**
Eu.ModSt.7676	A concurrent state as shown in *Figure 7683*, sometimes also called an orthogonal composite state, contains at least two regions.
Eu.ModSt.7677	When a concurrent state is active, each region has its own active state that is independent of the others, and any incoming event is independently analysed within each region.
Eu.ModSt.7678	A transition that ends on the concurrent state, such as transition T1 in *Figure 7683*, will trigger transitions from the initial pseudostate of each region, so there must be an initial pseudostate in each region for such a transition to be valid.
Eu.ModSt.7679	Similarly, a completion event for the concurrent state will occur when all the regions are in their final state.
Eu.ModSt.7680	When an event, as for example the internal broadcast event bc1_Bc_info shown in *Figure 7683*, is associated with triggers in multiple orthogonal regions, the event may trigger a transition in each region (e.g. transitions T3 and T5), assuming the transition is valid based on the other usual criteria.
Eu.ModSt.7681	**Please note:** a transition can never cross the boundary between two regions of the same concurrent state.
Eu.ModSt.7682	In addition to transitions that start or end on the concurrent state, such as transition T1 in *Figure 7683*, transitions from outside the concurrent state may start or end on the nested states of its regions. In this case, one state in each region must be the start or end of one of a coordinated set of transitions. This coordination is performed by a fork pseudostate in the case of incoming transitions, such as T8.1, T8.2 and T8.3 in *Figure 7683*, and a join pseudostate for outgoing transitions, such as T6.1, T6.2 and T6.3 in *Figure 7683*.

ID	Requirement
Eu.ModSt.7683	Figure 7683 Example of a concurrent state 
Eu.ModSt.7684	**8.6.6.14 Decomposition of states using state machine diagrams**
Eu.ModSt.7685	Instead of decomposing the behaviour of a state within a region of a sequential state or multiple regions of a concurrent state, the behaviour may alternatively be specified by a state machine diagram assigned to the corresponding state (see *Figure 7689*).
Eu.ModSt.7686	The region of the corresponding state machine diagram typically will contain an initial pseudostate and a final state, a set of pseudostates, and a set of substates, which themselves may be decomposed by state machine diagrams.
Eu.ModSt.7687	As illustrated in *Figure 7689*, a transition (e.g. transition T1) ending on a state (e.g. state ST2) that is refined by a state machine diagram will trigger the transition from the initial pseudostate of the diagram to its initialising state (e.g. state ST2_1).

ID	Requirement
Eu.ModSt.7688	Similarly, when the behaviour specified on the state machine diagram completes (e.g. the final state is entered after triggering the transition T2), it will generate a completion event that can trigger transitions (e.g. transition T3) whose source is the state (e.g. state ST2) the state machine diagram is assigned to.
Eu.ModSt.7689	Figure 7689 Principle of decomposing states by means of state machine diagrams 
Eu.ModSt.7690	**8.6.6.15 Transition firing order in nested state hierarchies**
Eu.ModSt.7691	The same event may trigger transitions at several levels in a state hierarchy, and with the exception of concurrent regions, only one of the transitions can be taken at a time. Priority is given to the transition whose source state is innermost in the state hierarchy.
Eu.ModSt.7692	Suppose the state machine depicted in *Figure 7695* is in its initial state (i.e. in state ST1_1_1 and ST1_2_1). The stimulus T1_Stimulus_1 is associated with the triggers of the transitions T1, T2 and T3, each with guards based on the value of D2_No.
Eu.ModSt.7693	The following list shows the transitions that will fire upon receipt of T1_Stimulus_1 based on values of D18_No from -1 to 1 if the system is in the states ST1_1_1 and ST1_2_1: • D2_No equals -1: transition T3 will be triggered because it is the only transition with a valid guard; • D2_No equals 0: transition T1 will be triggered because, although transition T3 also has a valid guard, state ST1_1_1 is the innermost of the two source states; or • D2_No equals 1: both transitions T1 and T2 will be triggered because both their guards are valid.
Eu.ModSt.7694	The normal rules for execution of exit behaviour apply, so, before the transition from state ST1 to state ST2 can be taken, any exit behaviour of the active nested states of state ST1, as well as the exit behaviour of state ST1, must be executed.

ID	Requirement
Eu.ModSt.7695	Figure 7695 Illustration of transition firing order 
Eu.ModSt.1078	**8.6.6.16 Interaction between state machines**
Eu.ModSt.1082	State machines in different blocks, may interact with one another by sending stimuli and returning responses. For example, the state machine of one block can send a stimulus to another block as part of a transition effect or state behaviour. The event corresponding to the receipt of this stimulus by the receiving block can trigger a state transition in its state machine.
Eu.ModSt.1083	Thus, different behaviour, each specifying a certain functionality of the system, may be encapsulated in blocks and interconnected with each other in a network of FEs or TFEs, i.e. in a Functional or Technical Functional Architecture.
Eu.ModSt.7831	**8.6.7 Bindings** (see *chapter 8.2.1*)
Eu.ModSt.7833	**Diagram of model view "Functional Entity" (ibd and stm)** has a "**Req**" binding.
Eu.ModSt.7834	**Diagram of model view "Technical Functional Entity" (ibd and stm)** has a "**Req**" binding.
Eu.ModSt.7837	The algorithm defined in a **time advanced operations** has a "**Req**" binding. The algorithm defined in a time advanced operation represents the mandatory externally visible behaviour of a FE or TFE in place of or in cooperation with a state machine.
Eu.ModSt.7839	**Transitions, states, ports, block operations and block properties** have "**Def**" bindings.
Eu.ModSt.7537	**8.6.8 Action language**
Eu.ModSt.7538	The EULYNX methodology follows the objective of creating executable specification models. In order to specify the necessary executable behaviours in a target language independent way, the Atego Structured Action Language (ASAL) is used.
Eu.ModSt.7539	ASAL is used to specify block operations or Event Action Blocks that define the transition effects on state machine diagrams.
Eu.ModSt.1940	A description of data types, logical operators and basic statements of the Atego Structured Action Language (ASAL) is provided below.
Eu.ModSt.7541	**8.6.8.1 Logical operators**

ID	Requirement
Eu.ModSt.7542	• Greater than: **>** • Less than: **<** • Greater than or equal: **>=** • Less than or equal: **<=** • Equal: **=** • Not equal: **<>** • Conjunction: **AND** • Disjunction: **OR** • Negation: **NOT** • Exclusive disjunction: **XOR**
Eu.ModSt.7840	The logical operators "AND", "OR", "NOT" and "XOR" are to be written in capital letters.
Eu.ModSt.7543	**8.6.8.2 Data types**
Eu.ModSt.7544	As the EULYNX specification approach follows the objective of creating executable specification models, the range of data types is limited to data types the simulation tool SySim supports (SySim value types).
Eu.ModSt.294	Only the SySim value types, including the redefined data types "PulsedIn" and "PulsedOut" may be used for the specification of systems requirements : • Boolean • DateTime • Single • String • Decimal • Double • Long • Integer • Timespan • PulsedIn • PulsedOut
Eu.ModSt.7546	The data types "PulsedIn" and "PulsedOut" represent redefinitions of the data type Boolean and are exclusively reserved to be assigned to Trigger Ports (T-Ports). That is, a Trigger In Port is typed with the data type "PulsedIn" and a Trigger Out Port with the data type "PulsedOut".
Eu.ModSt.7547	Outgoing data typed with "PulsedOut" (as default false) that are set to true (for example, T1out_Cd_indicate_signal_aspect := true) automatically change back to false after a defined time. The defined time frame is sufficient to trigger a transition in a receiving state machine.
Eu.ModSt.7548	Incoming data at receiver side typed with "PulsedIn" apply the behaviour of the corresponding outgoing data at sender side typed with "PulsedOut".
Eu.ModSt.7906	For the typing of proxy ports, the specially adapted interface blocks are to be used: • IBoolean • IDateTime • IDecimal • IDouble • IInteger • ILong • ISingle • IString
Eu.ModSt.7907	The data types "PulsedIn" and "PulsedOut" can only be used with flow ports but not in connection with proxy ports.
Eu.ModSt.269	**8.6.8.3 Declaring variables**
Eu.ModSt.270	The Declare statement declares local variables. The syntax is as follows: declare <variable list> : <type> ; Where: · <variable list> - specifies a list of variables that are being declared. For each variable an optional initial value can be set through the ':=' assignment operator. · <type> - specifies the type of the variables that are being declared.

ID	Requirement

	Example:
	declare A : Boolean;
	declare B := False : Boolean;
	declare C, D := 0 : Integer;

Eu.ModSt.7549	**8.6.8.4 Reading the value of a port**
Eu.ModSt.7550	The value of a port may be read using the name of the port on its own: The syntax is as follows: <A> := <port>; Where: <port> specifies the port whose value is being read. <A> specifies for example the value property the value of the port is to be assigned to. **Example:** Mem_D1_Signal_aspect := D1_Signal_aspect;
Eu.ModSt.7551	**8.6.8.5 Setting the value of a port**
Eu.ModSt.7552	The value of a port may be set using the name of the port: The syntax is as follows: <port> := <value>; Where: · <port> - specifies the port whose value is being set. · <value> - specifies the value that is being set for the port. **Example:** T1_Cd_Indicate_signal_aspect := true;
Eu.ModSt.7553	**8.6.8.6 Calling an operation**
Eu.ModSt.7554	To call an Operation item in ASAL, reference the Operation with its default (the default is 'This'). You must use parentheses for the operation, even if there are no parameters to pass. The syntax is as follows: <operation> ([<parameters>]); Where: · <operation> - specifies the operation that is being called. By default, the Operation is called against 'This'. · <parameters> - specifies any parameter values that are passed to the operation that is being called. **Examples:** MyOperation(True); OperationWithNoParameters();
Eu.ModSt.7555	**8.6.8.7 Assigning values to variables**
Eu.ModSt.7556	Values can be assigned to variables. The syntax is as follows: <variable> := <expression> ; Where: · <variable> - specifies the variable that is being assigned. · <expression> - specifies the value that is being assigned, which can be defined through an expression. **Example:** Mem_ped_wait := False;
Eu.ModSt.7557	**8.6.8.8 Conditional execution of code**

ID	Requirement

Eu.ModSt.7558	The if, then, else statements provide a mechanism for conditional execution of code.
	The syntax is as follows:
	if <condition> then
	... //code to execute
	elseif <condition> then
	... //code to execute
	else
	... //code to execute
	end if
	Where:
	· <condition> - specifies the condition that is being tested.
	
	Example:
	if A < 100 then
	A := A + 1;
	elseif B < 100 then
	B := B + 1;
	else
	NowStop := True;
	end if

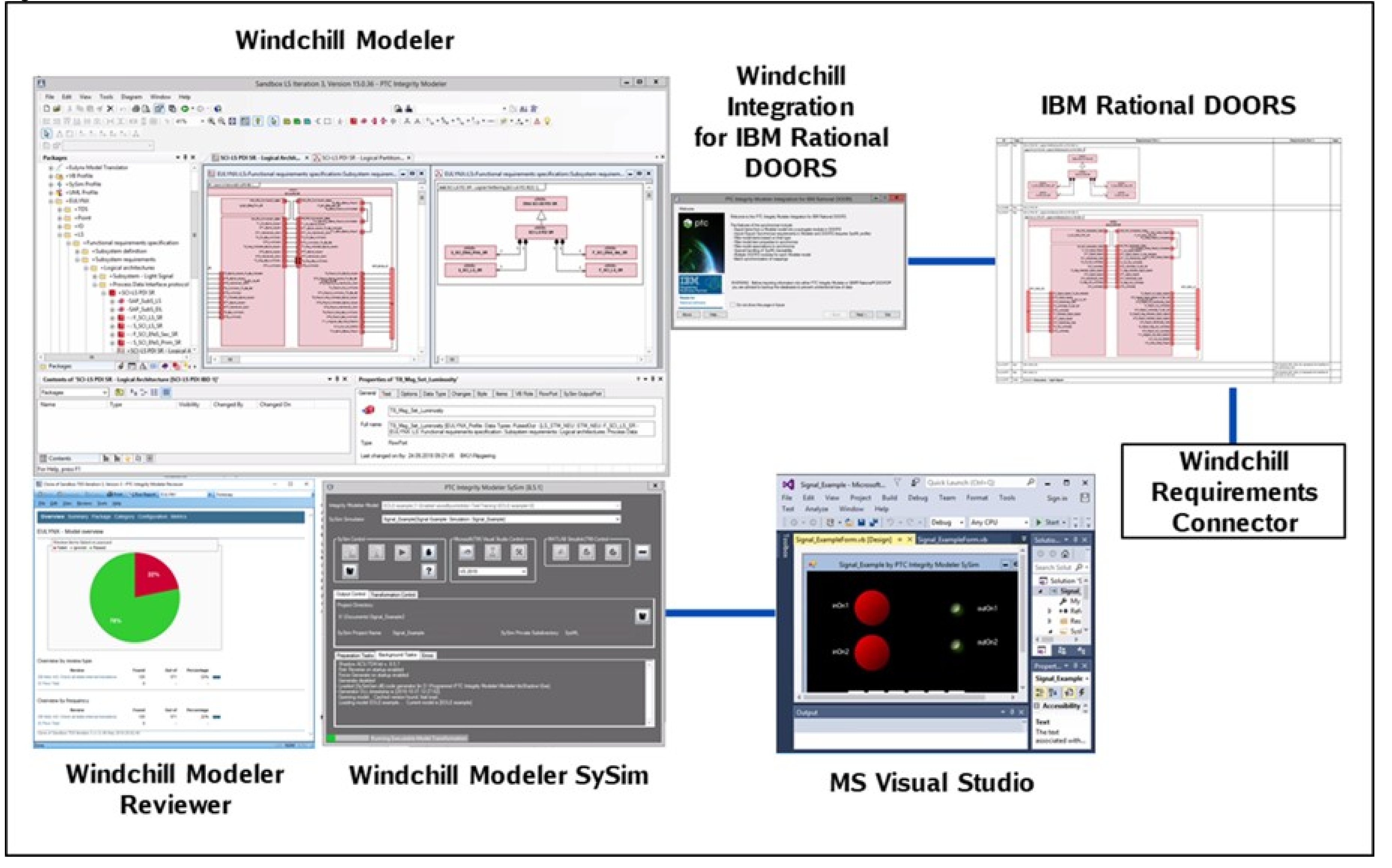
Eu.ModSt.7559	**8.6.8.9 While loops**
Eu.ModSt.7560	The while loop provides a mechanism for executing code while a condition is true. The syntax is as follows: while <condition> ... //code to execute end while Where: · <condition> - specifies the condition that is being tested. **Example:** while A < 100 A := A + 1; end while
Eu.ModSt.7561	**8.6.8.10 Case selection**
Eu.ModSt.7562	The case selection provides a mechanism for executing code when a case is true. The syntax is as follows (note that there can be many cases): select case <condition> case <condition>: ... //code to execute case else: ... //code to execute end select Where: · <condition> - specifies the condition that is being tested. **Example:** select case A + B case 200: ResultIs200 := True; case else: ResultIs200 := False; end select
Eu.ModSt.7563	**8.6.8.11 Return statement**

ID	Requirement
Eu.ModSt.7564	The Return statement can return the result of an expression. The syntax is as follows: return <expression> ; Where: · <expression> - specifies the expression that returns the result. **Example:** return A + B;
Eu.ModSt.287	**8.6.8.12 Comments**
Eu.ModSt.288	The Comment statement specifies text that is ignored by the target language. The syntax is as follows for single line comments: // <text> Where: · <text> - specifies the text that is generated as a comment.
Eu.ModSt.289	Example: // return the sum of A + B
Eu.ModSt.290	**8.6.8.13 Example program written in ASAL**

Eu.ModSt.291	This is an example program that is written in ASAL.
	declare A := 0, B: Integer; // Former declared variable initialized, latter is not. Both share the same type
	declare GoOn := True : Boolean;
	declare NowStop := False : Boolean;
	B := 0; // Assignment
	NowStop := not B = 0 AND (GoOn or NowStop); // Assignment (it's False) using a logical expression
	while GoOn AND NOT NowStop do // While loop
	if A < 100 then // Condition … if
	A := A + 1;
	elseif B < 100 then // Condition, elseif
	B := B + 1;
	else // Condition, else
	NowStop := True;
	end if // end of condition.
	end while
	declare TestOk : Boolean;
	select case A + B // Selection statement. It's similar to C/C++ switch (but no "break", only one case is executed at most)
	case 199 + (A + B) / (A + B): // Case expression, equates to 200
	TestOk := True;
	case else: // Default case
	TestOk := False;
	end select
	return A + B; // Return statement

Eu.ModSt.825	**9 References**
Eu.ModSt.826	[1] OMG Systems Modeling Language (OMG SysML ™),https://sysml.org/.res/docs/specs/OMGSysML-v1.6-19-11-01.pdf
Eu.ModSt.827	[2] OMG Unified Modeling Language TM (OMG UML), https://www.omg.org/spec/UML/2.5.1/PDF
Eu.ModSt.828	[3] KnowGravity Inc., RIAL Risk Analysis 10, 30.11.2014
Eu.ModSt.829	[4] M. Broy, K. Stølen, Specification and development of interactive systems, Focus on streams, interfaces, and refinement, Springer-Verlag New York, Inc, 2001
Eu.ModSt.830	[5] T. Weilkiens, Systems Engineering with SysML/UML, Modeling, Analysis, Design, dpunkt.verlag GmbH, Heidelberg, Germany, 2006
Eu.ModSt.831	[6] J. Braband, B-E. Brehmke, S. Griebel, H. Peters, K-H Suwe, The CENELEC-Standards regarding Functional Safety, Eurailpress, 2006
Eu.ModSt.832	[7] H. König, Protocol Engineering, B. G. Teubner Verlag, 2003

ID	Requirement
Eu.ModSt.833	[8] R. J. Wieringa, Design methods for reactive systems, Elsevier Sience (USA), 2003
Eu.ModSt.834	[9] KnowGravity Inc., Modelling Interlocking Requirements using UML, A Guideline, UIC/Euro-Interlocking, 2002
Eu.ModSt.835	[10] M. Debbabi, F. Hassaine, Y. Jarraya, A. Soeanu, L. Alawneh, Verification and Validation in Systems Engineering, Springer-Verlag Berlin Heidelberg, 2010
Eu.ModSt.836	[11] T. Koch, Rechnergestützter Sicherheitsnachweis: Ein Verfahren zum Ausschluss gefährlicher Systemzustände in rechnergesteuerten Eisenbahn- Sicherungsanlagen, Dresden: Technische Universität, Fakultät Verkehrswissenschaften „Friedrich List“, Diss. 1997
Eu.ModSt.837	[12] M. Burkhard, M. Hoeft, Bericht Zielarchitektur ESTW, Deutsche Bahn AG Systemverbund Bahn VTZ 12, Stand 0.92, 18.01.2009
Eu.ModSt.838	[13] Atego Modeler, Help
Eu.ModSt.839	[14] http://de.ptc.com/application-lifecycle-management/integrity/modeler
Eu.ModSt.840	[15] http://de.ptc.com/application-lifecycle-management/integrity/modeler/sysim
Eu.ModSt.841	[16] http://de.ptc.com/application-lifecycle-management/integrity/modeler/reviewer
Eu.ModSt.842	[17] CENELEC: EN 50126, Railway Applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS)
Eu.ModSt.843	[18] J. Ernst, Das Sp Dr S60-Stellwerk, Eisenbahn-Fachverlag, Heidelberg/Mainz, 1978
Eu.ModSt.844	[19] O. Lemke, A. Harhurin, K-D Sievers, Systems-Engineering-Prozess für LST-Anwendungen unter Berücksichtigung aktueller Normen, DB Netz AG- I.NVT 31, 2011
Eu.ModSt.845	[20] I. Jacobson, M. Griss and P. Jonsson, Software Reuse: Architecture Process and Organisation for Business Success, Addison-Wesley 1997
Eu.ModSt.846	[21] Shane Sendall and Alfred Strohmeier, From use cases to system operation specifications, In Stuart Kent and Andy Evans, editors, UML 2000 - The Unified Modeling Language: Advancing the Standard, Third International Conference, volume 1939 of LNCS. Springer, 2000
Eu.ModSt.847	[22] Alistair Cockburn, Writing Effective Use Cases, Addison Wesley, 2001.
Eu.ModSt.848	[23] Heldal Rogardt, Use Cases are more than System Operations, Chalmers University of Technology, Gothenburg, Schweden.
Eu.ModSt.889	[24] Sanford Friedenthal, Alan Moore, Rick Steiner, A Practical Guide to SysML, Third Edition: The Systems Modeling Language, The MK/OMG Press, 2015.
Eu.ModSt.1495	[25] Klaus Pohl, Harald Hönninger, Reinhold Achatz, Manfred Broy, Model-Based Engineering of Embedded Systems, The SPES 2020 Methodology, Springer-Verlag Berlin Heidelberg 12.
Eu.ModSt.1469	[26] Klaus Pohl, Manfred Broy, Heinrich Daembkes, Harald Hönninger, Advanced Model-Based Engineering of Embedded Systems, Extensions of the SPES 2020 Methodology, Springer International Publishing AG 2016.
Eu.ModSt.1477	[27] Christer Löfving, Arne Borälv, Formal Methods Taxonomy and Survey, X2Rail-2 Deliverable D5.1, 16.05.2018
Eu.ModSt.1502	[28] https://de.mathworks.com/products/simulink.html
Eu.ModSt.1517	[29] The Institute of Electrical and Electronics Engineers, Inc.: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Std. 1471-2000. New York, 2000
Eu.ModSt.1568	[30] Osamu Shigo, Atsushi Okawa, Daiki Kato: Constructing Behavioral State Machine using Interface Protocol Specification, 13th Asia Pacific Software Engineering Conference (APSEC'06), IEEE 2006
Eu.ModSt.3552	[31] https://www.eulynx.eu/
Eu.ModSt.7072	[32] Deliverable D10.2 Proposed extension of specification approach to meet needs of RCA, WP 10, X2Rail-5, 13.01.2022
Eu.ModSt.7911	**10 Appendix A - Reference Tool Chain**
Eu.ModSt.7916	A tool chain that fully supports the EULYNX MBSE process is shown below and is intended to be a reference for the use of alternative tools. When using alternative tools, make sure that they have the same capabilities.
Eu.ModSt.302	The EULYNX MBSE process is currently supported by a toolchain as illustrated in Figure 3553. 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.

ID	Requirement
Eu.ModSt.3553	Figure 3553 EULYNX Tool chain 
Eu.ModSt.303	**10.1 Windchill Modeler**
Eu.ModSt.304	Windchill Modeler [14], an all-in-one integrated collaborative development tool suite, is used to create the EULYNX SysML specification models. It provides systems and software modelling and component-based development targeted for technical systems and provides comprehensive notation support for the leading industry standards, including OMG SysML, OMG UML, UPDM (DoDAF and MODAF), OVM, data modelling, and architectural frameworks.
Eu.ModSt.3554	**10.2 IBM Rational DOORS**
Eu.ModSt.3557	Requirements management tool IBM Rational DOORS is used to organise the specification contents in a format conforming to classical requirements management (atomic requirements with unique identifiers and allocated bindingness). The requirements are structured in the form of DOORS-objects in the DOORS-modules representing the specification documents. The specification models created in the Windchill Modeler are represented as surrogates in the DOORS-modules structured in the form of atomic requirements.
Eu.ModSt.3555	**10.3 Windchill Integration for IBM Rational DOORS**
Eu.ModSt.3558	Windchill Modeler is connected to the requirements management tool IBM Rational DOORS via the Windchill Integration for IBM Rational DOORS. This connection enables the creation and synchronisation of surrogates of the specification models in the requirements management tool.
Eu.ModSt.3556	**10.4 Windchill Requirements Connector**
Eu.ModSt.3559	Windchill Requirements Connector is used to transform DOORS-modules into Requirements Interchange Format (ReqIF) and retransform ReqIF files into DOORS format.
Eu.ModSt.305	**10.5 Windchill Modeler SySim**

ID	Requirement
Eu.ModSt.306	Windchill Modeler SySim [15] is used together with Windchill modeler and MS Visual Studio to create executable specifications (virtual prototypes) from SysML specification models and validate their behaviours by means of simulation-based testing. That way it is ensured that the corresponding specification model is consistent and formally correct without the need to focus on lower-level details such as code generation or target environments.
Eu.ModSt.307	Furthermore, Windchill Modeler SySim allows the generation of appropriate and intuitive simulation graphics. Graphical components are automatically prepared in an MS Visual Studio toolbox, from which they can be dropped onto a form to create each user interface, for a given simulation scenario. Predefined graphical components are also provided for the most common functions, such as input and output. Developing new graphical components is also made easy, using the de-facto standard Microsoft .NET platform.
Eu.ModSt.308	**10.6 MS Visual Studio**
Eu.ModSt.309	MS Visual Studio is applied to create graphical user interfaces used to play through simulation scenarios and build executables from simulation code generated by Windchill Modeler SySim.
Eu.ModSt.310	**10.7 Windchill Modeler Reviewer**
Eu.ModSt.311	Windchill Modeler Reviewer [16] provides a quick way of reviewing items in a model using provided and optionally user-defined reviews. EULYNX SysML specification models can quickly be checked for completeness, correctness and consistency using the corresponding reviews. Summary reports may be created that provide statistical analysis of review failures and metrics relating to items in a model. Furthermore, user-defined reviews may be created to include in reports.