

Rail to Digital automated up to autonomous train operation

D30.2 – Validation Approach Based on a Hybrid Testbed to Perform X-in-the-Loop Testing

Due date of deliverable: 31/12/2025

Actual submission date: 15/11/2025

Leader/Responsible of this Deliverable: DB

Reviewed: Y

Document status		
Revision	Date	Description
01	16/06/2025	Added table of contents and template.
02	31/07/2025	Added Chapter 6, Updated to the new deliverable template
03	31/08/2025	Added Chapters 1, 4
04	23/10/2025	Added Chapters 2, 3 and 5
05	01/11/2025	Added Chapters 8, 9 and 10; version for WP internal peer review
06	13/11/2025	Added Chapter 7; Revised chapters 4, 5 and 8; Edits to address review comments from DLR, ADIF; Added Executive summary; Updated figures, tables and cross-references; Finalised issue of the deliverable
07	15/11/2025	Addressed review comments from DLR and DB; Minor edits to the deliverable; Finalised the deliverable for TMT review
08	16/12/2025	Finalised the deliverable for SteCo review; SteCo review started on 08/12/2025
09	05/08/2026	Addressed specific issues review comments; Submitted PDF version for upload on SYGMA

Project funded from the European Union's Horizon Europe research and innovation programme

Dissemination Level

PU	Public	X
SEN	Sensitiv – limited under the conditions of the Grant Agreement	

Start date: 01/12/2022

Duration: 37 months

ACKNOWLEDGEMENTS



This project has received funding from the Europe's Rail Joint Undertaking (ERJU) under the Grant Agreement no. 101102001. The JU receives support from the European Union's Horizon Europe research and innovation programme and the Europe's Rail JU members other than the Union.

REPORT CONTRIBUTORS

Name	Company	Details of Contribution
Stalin Chriss Castelino	DB	Primary author, contributed to chapters 1, 2, 3, 4, 5, 7, 8, 9, 10. Review of chapter 6.
Daniel Schwencke	DLR	Co-author, contributed to chapters 6, 7, 8, 9, 10. Review of all chapters.
Abdul Rasheeq	DB	Co-author, contributed to chapters 1, 2, 3, 9, 10. Review of all chapters.
Sai Aditya Bommakanti	DB	Contributed to chapters 2, 3. Contributed to the formatting of the deliverable. Review of all chapters.
Tawonga Tsokodayi	DB	Contributed to chapters 4, 5
Akhil Parimi	DB	Contributed to chapters 4, 5
Miguel Fernandez Elorriaga	ADIF	Review of all chapters.
Ignacio Alguacil Ventas	ADIF	Review of all chapters.

Disclaimer

The information in this document is provided “as is”, and no guarantee or warranty is given that the information is fit for any particular purpose. The content of this document reflects only the author's view – the Joint Undertaking is not responsible for any use that may be made of the information it contains. The users use the information at their sole risk and liability.

EXECUTIVE SUMMARY

Railway signalling systems across Europe are undergoing a major transformation toward digitalisation and interoperability. The EULYNX initiative, now integrated into Europe's Rail System Pillar, provides standardised specifications using SysML-based models. However, validating real-world implementations against these specifications remains a challenge. Deliverable D30.2 addresses this gap by introducing a **Hybrid Testbed (HTB)** approach for X-in-the-Loop testing, enabling automated conformity checks and reducing reliance on manual verification. This work is part of Task 30.1.2 under WP30 of the FP2 R2DATO project, supporting Europe's vision for a Single European Railway Area (SERA) [31].

This deliverable documents the development of a validation approach that integrates models within a Hybrid Testbed (HTB). The primary goal is to verify the conformity of hardware implementations against EULYNX SysML-based specification models. To achieve this, the HTB is designed to support a plug-and-play architecture, enabling modularity in railway interlocking systems and fostering participation from multiple suppliers. In addition to the HTB, a versatile test component has been developed to facilitate systematic validation. The case study focuses on the EULYNX Subsystem Point, demonstrating how model-based requirements can be tested for preliminary compliance in a controlled environment.

The deliverable proposes a modular and scalable HTB that integrates SysML models with hardware and software components. Two complementary strategies were implemented:

- **White Box Approach:** Direct integration of SysML models with hardware prototypes to validate internal consistency and behaviour.
- **Black Box Approach:** Functional validation of subsystem behaviour at the SCI-P interface using Python-based simulation and GUI-driven testing.

A versatile **Point Test Component (TC)** was developed to simulate EULYNX-compliant behaviour, supporting fault injection and scenario-based testing. The HTB and TC were integrated via secure UDP communication, enabling remote validation in near-real conditions.

Key findings and results:

- Achieved Technology Readiness Level (TRL) 3, demonstrating proof of concept in a laboratory environment.
- Validated EULYNX Baseline 3 and Baseline 4 specifications through automated and manual test cases.
- Demonstrated real-time bidirectional communication between interlocking and field element subsystems.
- Identified gaps in current specifications (e.g., event scheduling in SysML statecharts) and proposed mitigation strategies.

This deliverable provides a **replicable framework for model-based validation**, reducing cost and time-to-market for signalling solutions. It supports standardisation efforts under Europe's Rail, enabling automated authorisation processes and fostering supplier diversity. The HTB approach strengthens confidence in interoperable signalling systems, contributing to safer and more efficient railway operations across Europe.

ABBREVIATIONS AND ACRONYMS

AL	Abstraction Level
AM MBSE	Architecture Model MBSE (EULYNX-specific)
API	Application Programming Interface
ASAL	Atego Structured Action Language
ATO	Automatic Train Operation
bdd	Block Definition Diagram
BL3R8	EULYNX Baseline 3 Release 8
BL4R2	EULYNX Baseline 4 Release 2
CCS	Control Command and Signalling
CENELEC	European Committee for Electrotechnical Standardisation
CSP	Cross-cutting System Properties
DLR	Deutsches Zentrum für Luft- und Raumfahrt e.V.
EfeS	EULYNX field element subsystem
EIL	Electronic Interlocking
ERJU	Europe's Rail Joint Undertaking
FE	Functional Entity
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HDMI	High-Definition Multimedia Interface
HTB	Hybrid Testbed
I/O	Input/Output
ibd	Internal Block Definition (diagram)
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IF	Interfaces
IM	Infrastructure Manager
INCOSE	International Council on Systems Engineering
IoT	Internet of Things
IP	Internet Protocol
ISO	International Organization for Standardisation
IXL	Interlocking

LAB	Logical Architecture Blank (diagram)
LED	Light Emitting Diode
MBSE	Model-based Systems Engineering
MBSE-SF	MBSE Specification Framework (EULYNX-specific)
OMG	Object Management Group
OS	Operating System
P3 IF	Interface between Point Controller and Point Simulation
PCB	Printed Circuit Board
PDI	Protocol Data Interface
PM	Point Machine
PTC	Parametric Technology Corporation
R2DATO	Rail to Digital automated up to autonomous train operation
RAMS	Reliability, Availability, Maintainability, and Safety
RaSTA	Rail Safe Transport Application
REST	Representational State Transfer
SCI	Standard Communication Interface
SCI-P	EULYNX interlocking to point machine interface
sd	Sequence Diagram
SDI	Standard Diagnostic Interface
SERA	Single European Railway Area
SIUS	System Interface under Specification
SMI	Standard Maintenance Interface
stm	State Machine (diagram)
SUS	(Sub)System Under Specification
SUT	System Under Test
SysML	Systems Modelling Language
TACS	Trackside Asset Control and Supervision
TC	Test Component
Traffic CS	Traffic Control and Supervision
TRL	Technology Readiness Level
uc	Use Case (diagram)
UDP	User Datagram Protocol
VPN	Virtual Private Network

WiFi	Wireless Fidelity
WP	Work Package
XMI	XML Metadata Interchange
XML	Extensible Markup Language

TABLE OF CONTENTS

Acknowledgements.....	3
Report Contributors.....	3
Executive Summary	4
Abbreviations and Acronyms	5
Table of Contents.....	8
List of Figures	11
List of Tables	13
1 Introduction	14
1.1 Motivation and Challenges	15
1.1.1 Hybrid Testbed – Authorisation of Railway Signalling Systems.....	16
1.1.2 Test Component - Testing Standardised Modular Signalling Systems	16
1.2 Overview of the Deliverable	18
2 The EULYNX Initiative and Methodology.....	19
2.1 EULYNX Specifications	19
2.1.1 Baseline 3.....	20
2.1.2 Baseline 4.....	20
2.1.3 EULYNX Input Documents	20
2.2 EULYNX MBSE	21
2.2.1 Viewpoint.....	21
2.2.2 Abstraction Level	22
2.2.3 View.....	22
2.2.4 Engineering Path	22
2.2.5 SysML	23
3 Reference model(s).....	25
3.1 SysML Modelling in EULYNX	25
3.1.1 Block.....	25
3.1.2 Port.....	26
3.1.3 Block Definition Diagram (BDD/bdd).....	27
3.1.4 Internal Block Definition (IBD/ibd)	29
3.1.5 Use Case Diagram (uc)	29
3.1.6 Sequence Diagram (seq/sd)	29
3.1.7 State Machine Diagram (stm)	30
3.1.8 Connector	30
3.1.9 State	30
3.1.10 Transition.....	31

3.1.11	Use Case.....	31
3.1.12	Actor.....	31
3.1.13	Interaction.....	31
3.1.14	Operation.....	31
3.1.15	Signal	31
3.1.16	Stereotype	31
4	Testbed – White Box Approach.....	32
4.1	Testbed Development – SysML Hardware Integration.....	32
4.1.1	Languages and Tools	32
4.1.2	Model Simulation	33
4.1.3	Model – IoT Platform Connection.....	34
4.1.4	Hardware Development	36
4.1.5	Hardware – IoT Platform Connection.....	36
4.1.6	Hybrid Testbed Integration.....	36
4.1.7	Testing.....	37
4.2	Testbed Development – SysML Virtual Sampler Integration	38
4.2.1	System Under Specification (Hybrid Testbed).....	39
4.2.2	Design definition process.....	42
4.2.3	HTB components	43
4.2.4	Validation Procedure	46
4.3	Limitations	47
5	Testbed – Black Box Approach	48
5.1	Hybrid Testbed Architecture	48
5.2	Hybrid Testbed Implementation	50
5.2.1	Languages and Tools	50
5.2.2	Testbed Behaviour Reference	50
5.2.3	Testbed GUI	55
5.2.4	Testbed Remote Connection	55
5.2.5	Testbed Telegram Communication	55
5.2.6	Test Suite	58
5.3	Hybrid Testbed Validation.....	58
6	A Versatile Point Test Component.....	60
6.1	Basic Idea: Use Cases and User Requirements	60
6.2	Concept: Realisation of the EULYNX Subsystem Point	60
6.2.1	Scope and Architectural Overview	60
6.2.2	Existing and Made-Up Specifications.....	62

6.2.3	Hardware and Software	63
6.3	Design and Implementation: From Configurable and Manipulable Functions To Statechart Code 63	
6.3.1	Design	63
6.3.2	Implementation	65
6.4	Validation: Testing the Test Component	67
6.4.1	Validation Set-Up at RailSiTe® Lab	67
6.4.2	Component Testing	67
7	Testbed – Test Component Integration	68
7.1	Testbed Integration Architecture	68
7.2	Integration Testing	69
7.2.1	Define Scope and Objectives	69
7.2.2	Prepare the Testing Environment	70
7.2.3	Initialise Communication and Interface Validation	70
7.2.4	Execute Test Cases	70
7.2.5	Log, Monitor and Analyse Results	70
7.2.6	Validate and Document Outcomes	70
8	Test cases	72
8.1	Test Cases for White Box Approach Hybrid Testbed	72
8.2	Test Cases for Black Box Approach Hybrid Testbed	77
8.3	Test Cases for Point Test Component at RailSiTe® Lab	79
8.4	Test Cases for Hybrid Testbed – Test Component Integration	81
8.4.1	Test Run Evidence	86
9	Results	90
9.1	Results – White Box Testbed Validation Approach	90
9.2	Results – Black Box Testbed Validation Approach	91
9.3	Results – Point Test Component Validation at RailSite Lab	91
9.4	Results – Integration of Hybrid Testbed and Test Component	92
9.5	Results – Test Case Execution	92
10	Conclusions	93
10.1	Deploying a Hybrid Testbed for Validation	93
10.2	How to Successfully Create a Versatile Test Component	94
10.3	Future Work	94
10.4	Final Remarks	95
	References	96

LIST OF FIGURES

Figure 1: Overview of the EULYNX hybrid testbed development.....	14
Figure 2: Example set-up for testing (in red) in a system-of-systems context (in blue)	17
Figure 3: Countries and IMs involved in EULYNX [© EULYNX]	19
Figure 4: Typical control loop of a EULYNX subsystem [© EULYNX].....	21
Figure 5: EULYNX System Definition [© EULYNX]	22
Figure 6 : EULYNX MBSE Methodology [© EULYNX].....	23
Figure 7: Relationship between SysML and UML and their respective diagrams	24
Figure 8: SysML Diagrams.....	24
Figure 9: HTB represented by a SysML block	26
Figure 10: IBD of a block containing a proxy port and a flow port.....	26
Figure 11: Example of an EULYNX SysML block (Baseline 4)	27
Figure 12: SysML Block Definition Diagram	27
Figure 13: SysML internal block diagram	28
Figure 14: SysML use case diagram	28
Figure 15: SysML sequence diagram.....	29
Figure 16: SysML state machine diagram	30
Figure 17: Hybrid Testbed – White Box Approach Setup	32
Figure 18: Virtual Prototype Simulator.....	33
Figure 19: ibd of Functional Entity for simulation	34
Figure 20: stm of Functional Entity for simulation	34
Figure 21: Simulation of virtual prototype	35
Figure 22: SysML model integrated into hybrid testbed.....	35
Figure 23: EULYNX-based hardware prototype for Subsystem Point.....	36
Figure 24: Hardware integrated into hybrid testbed	37
Figure 25: Integrated hybrid testbed.....	37
Figure 26: IBD showing relevant parts of the SCI-P and the Sampler Block.....	39
Figure 27: Use Case Diagram for HTB.....	40
Figure 28: Sequence Diagram of HTB system components executing validation	41
Figure 29: HTB Architecture.....	41
Figure 30: IBD of sampler block	42
Figure 31: SCI-P_Sampler virtual entity properties.....	43
Figure 32: Services belonging to Control_Value virtual entity.....	44
Figure 33: Binding the Validation_Test service to the Validate button	44
Figure 34: Javascript snippet from the service Validation_Test.....	45
Figure 35: HTB Mashup GUI.....	45

Figure 36: GUI for simulating the SCI-P	46
Figure 37: Sequence diagram of the validation process	47
Figure 38: Hybrid Testbed – Black Box Approach Setup	48
Figure 39: Hybrid Testbed System Architecture	49
Figure 40: [LAB] Set point position (Signaller) [Control loop] [© ERJU System Pillar]	49
Figure 41: EULYNX Functional Entities considered for HTB implementation	50
Figure 42: IBD of S_SCI_EfeS_Prim functional entity	51
Figure 43: IBD of EULYNX Point S_SCI_P_Command FE	51
Figure 44: stm of EULYNX Generic EIL S_SCI_EfeS_Prim functional entity	52
Figure 45: stm of EULYNX Point S_SCI_P_Command FE	52
Figure 46: IBD of EULYNX Point S_SCI_P_Receive FE	53
Figure 47: stm of EULYNX Point S_SCI_P_Receive FE	53
Figure 48: HTB User GUI	55
Figure 49: HTB and components in Tailscale network for remote connection and validation	56
Figure 50: Generic SCI information flows communicated as telegram with ISO/IEC 8859-1:1998	56
Figure 51: SCI-P information flows communicated as telegram with ISO/IEC 8859-1:1998	57
Figure 52: Overview of EULYNX Telegram Encoding/Decoding according to ISO:8859-1:1998 ...	57
Figure 53: HTB test suite with test driver integrated as test runner	58
Figure 54: Overview of the TC's structure and IF	61
Figure 55: Statechart for observing the PM position as part of the PM simulation specification	62
Figure 56: Example statechart transition, redrawn from [20]	65
Figure 57: Screenshot of the first tab of the TC's GUI for overall monitoring and control	66
Figure 58: Overview of validation approach with HTB integration	68
Figure 59: [LAB] Hybrid Testbed Integration Architecture	69
Figure 60: Refined Hybrid Testbed Integration Architecture	69
Figure 61: GUI of HTB displaying results of a successful validation test	74
Figure 62: GUI of HTB displaying results of a failed validation test	76
Figure 63: Automatic test case execution in test suite of HTB with test driver	78
Figure 64: Automatic test case execution in test suite of HTB with HTB GUI	79
Figure 65: Results sheet saved after automatic test case execution by HTB	79
Figure 66: TC_Eu.Gen-SCI.444.01 test case description	81
Figure 67: TC_Eu.Gen-SCI.444.01 test case	82
Figure 68: TC_Eu.P.5787.01 test case description	83
Figure 69: TC_Eu.P.5787.01 test case	83
Figure 70: TC-004 test case	84
Figure 71: TC_Eu.P.6714.01 test case description	84

Figure 72: TC_Eu.P.6714.01 test case	85
Figure 73: Telegram exchanges on the SCI-P IF between HTB and TC viewed from HTB (TC_Eu.Gen-SCI.444.01, TC_Eu.P.5789.01 and TC-004)	86
Figure 74: Fault-injection scenario with TC only reacting to correct telegram from HTB (TC-004) ..	86
Figure 75: Telegram exchanges on the SCI-P IF between HTB and TC viewed from TC (TC_Eu.Gen-SCI.444.01).....	86
Figure 76: TC with point status reported and PDI connection established	87
Figure 77: HTB with point status LEFT reported from TC and PDI connection established	87
Figure 78: SCI-P telegram exchanges to move point from left position to right position viewed from TC	88
Figure 79: Log excerpt of test run (TC_Eu.P.6722.01)	88
Figure 80: HTB with point position RIGHT, NOT_APPLICABLE after commanding TC to move point to RIGHT position (NOT APPLICABLE: Degraded point position not applicable)	89
Figure 81: EULYNX-based hardware prototype for Subsystem Point.....	90

LIST OF TABLES

Table 1: TC 1 – Command RIGHT	74
Table 2: TC 2 – Command LEFT	75
Table 3: TC 3 – Command RIGHT fault	76
Table 4: TC 4 – Command LEFT fault.....	77
Table 5: Black Box HTB automatically executed test cases	78
Table 6: Log excerpt from a test run.....	81

1 INTRODUCTION

This document constitutes the Deliverable 30.2 “Validation Approach Based on a Hybrid Testbed to Perform X-in-the-Loop Testing” in the framework of the WP30 of Flagship Project 2 R2DATO. The project covered by this deliverable is the work on Task 30.1.2 in WP30 and enables the concept for automated authorisation described by WP30.

Deliverable D30.2 introduces a comprehensive validation approach designed to test and verify the compliance of EULYNX implementations with the EULYNX BL4R2 specifications. This deliverable presents a case study centred on the development of a hybrid testbed, which serves as a proof of concept for establishing an automated testing and authorisation environment for EULYNX subsystem implementations. D30.2 aims to demonstrate how systematic validation and testing can ensure the correct functional behaviour of EULYNX components while supporting automated conformity assessment. The deliverable outlines the proposed methodology, describes the test environment, and illustrates how the approach can be applied to validate subsystem performance against EULYNX requirements. Figure 1 shows an overview of the EULYNX hybrid testbed development.

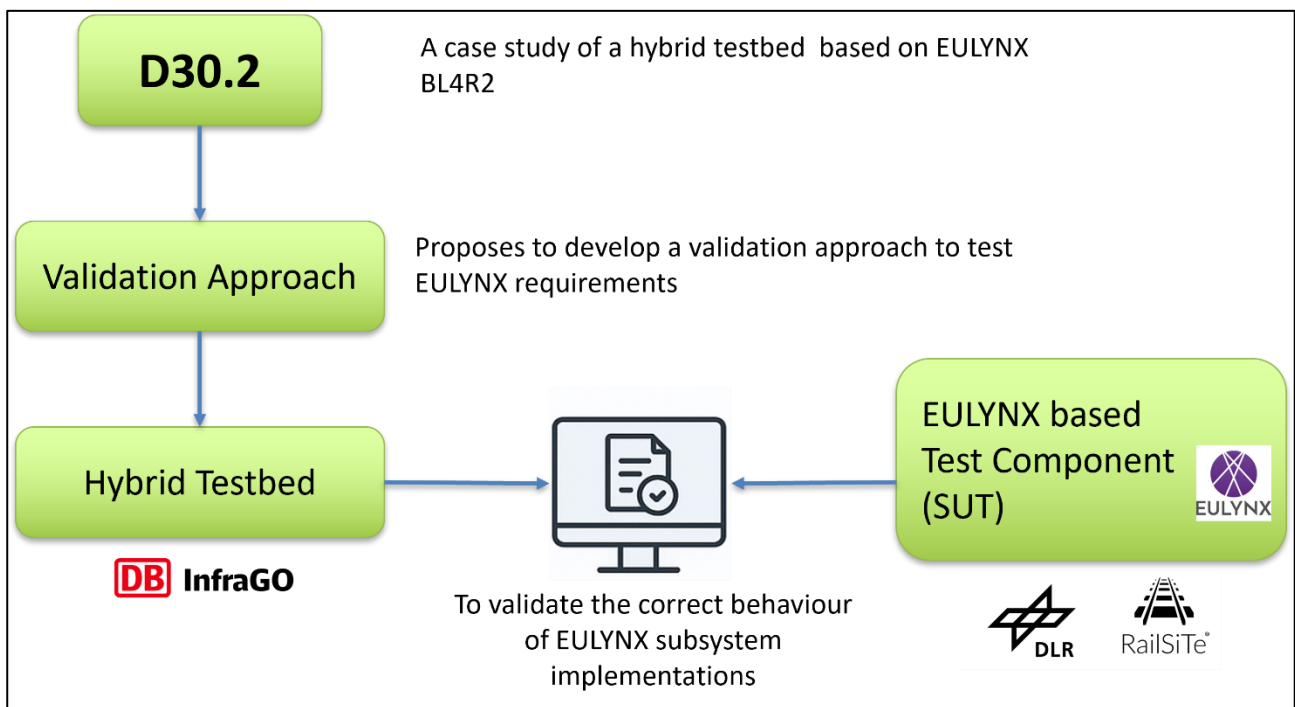


Figure 1: Overview of the EULYNX hybrid testbed development

Achieving Technology Readiness Level (TRL) 3 for the approach based on a hybrid testbed to perform X-in-the-Loop testing signifies the successful validation of key components of the proposed hybrid testbed and testing methodology as a proof-of-concept. At this stage, the project has demonstrated the integration of SysML-based EULYNX specification models with prototype subsystems through a functional hybrid testbed capable of executing X-in-the-Loop tests. The validation activities include the implementation and execution of basic test cases and interface validation based on the EULYNX requirements, focusing on the EULYNX SCI-P interface and point test component.

The development of the validation approach and hybrid testbed described in this deliverable is based on input documents and specifications from the EULYNX initiative. As of 2023, the EULYNX specifications are integrated in Europe's Rail, bringing a part of the EULYNX development under technical authority of the Europe's Rail System Pillar [1]. Key input documents are EULYNX requirement specifications and interface specifications mentioned in Section 2.1.3.

1.1 MOTIVATION AND CHALLENGES

The EULYNX initiative [2] aims to standardise and harmonise railway signalling interfaces across Europe using model-based systems engineering and formalised SysML specification models. These specification models capture requirements, interfaces and expected behaviours in a precise and structured way, thereby enabling the systematic development and verification of interoperable signalling solutions. As products developed from the EULYNX specifications are starting to enter the market, it becomes increasingly important to validate their implementation against the SysML-based specification models to ensure conformance and robustness in realistic operating conditions.

A required step in the automated authorisation of EULYNX railway assets is to be able to test the implementation of EULYNX prototypes (hardware) against the SysML-based specification models of EULYNX.

Currently, using documents, human effort and judgement are needed to manually check the conformity of the requirements specified in the specification models. A hybrid testbed that connects the SysML-based specification models and the implementation (Hardware) together automates the conformity of the requirements specified in the specification models of EULYNX by performing various tests that ensure the validation of the concerned EULYNX model-based requirements.

By combining rigorous EULYNX SysML-based specification models with an executable validation environment, this deliverable enables that EULYNX components are tested for functional conformance, robustness and compliance. The hybrid testbed serves as a key enabler for future railway digitalisation efforts, fostering confidence in interoperable solutions across European infrastructure managers and suppliers.

The motivation is to integrate hardware and software models in a simple, seamless and modular way using the hybrid testbed and the test component. The development of the hybrid testbed is broken down into multiple phases that produce constrained problems, challenges and solutions documented towards milestones. These milestones and completion of phases are in chronological order and are integrated into the solution of the hybrid testbed.

While the EULYNX SysML-based specification models provide a rigorous foundation, a gap exists between the SysML specification models and the implemented prototypes. Traditional testing approaches are often insufficient to capture the complexity and variability inherent in railway systems, particularly when it comes to handling fault conditions, degraded modes and interoperability scenarios. Without a systematic validation environment, it is difficult to assess whether implemented components fully satisfy the requirements specified in the models or how implemented components behave under unexpected conditions.

The primary objective of this deliverable is to describe and present a validation approach that bridges the gap between the EULYNX SysML-based specification models and test component implementations through the development of a hybrid testbed. This hybrid testbed will allow the execution of X-in-the-Loop testing (e.g., Model-in-the-Loop, Hardware-in-the-Loop or Software-in-

the-Loop), enabling conformance checks against the EULYNX specifications and validation techniques such as fault injection testing and scenario-based demonstrations.

The aim of the validation approach is:

1. To provide a structured methodology for connecting SysML models directly with test component implementations.
2. To design and implement validation workflows that ensure compliance with EULYNX requirements.
3. To establish foundation for scalable and repeatable testing that can evolve alongside future developments in EULYNX, Europe's Rail System Pillar and related railway standardisation efforts.

1.1.1 Hybrid Testbed – Authorisation of Railway Signalling Systems

To address the identified challenges, this deliverable proposes a hybrid testbed architecture that integrates SysML-based models with actual component implementations under test. The testbed will act as a validation environment where model-based requirements can be automatically or semi-automatically tested against prototype behaviour. Fault injection testing is embedded into the setup, allowing systematic exploration of resilience and safety aspects. The hybrid testbed enables different X-in-the-Loop configurations, offering flexibility for various validation stages.

This approach represents a novel contribution by systematically linking formal SysML models with executable validation environments. Prior research and existing EULYNX documents provide a foundation for modelling and specification; the work done to produce this deliverable advances the state-of-the-art by introducing an integrated validation framework that operationalises the EULYNX specifications into executable test scenarios.

Nevertheless, additional challenges have been identified, including automatic test case generation from SysML models, the scalability of the hybrid testbed to a system-of-systems context and the incorporation of cybersecurity considerations into fault injection testing. These challenges represent promising direction for future research and development, building upon the foundation established in this deliverable.

1.1.2 Test Component - Testing Standardised Modular Signalling Systems

For several years, European railway initiatives have been developing standardised modular architectures for control, command and signalling (CCS) systems. The EULYNX initiative is one such initiative working on the standardisation of railway trackside assets as described in chapter 2. Whereas previously modularisation, the design of interfaces (IF) and the detailed behaviour of CCS subsystems largely was a supplier-internal matter, with the new architectures they are now prescribed to a greater extent. This leads to exchangeable subsystems, potentially provided by many different suppliers. This in turn induces an increased need for verification and validation, to prove – to the external system integrator and to authorities – that the subsystems will actually interact towards realisation of the intended system of systems behaviour. In the first place, conformity to the standardised specification is to be verified, and ideally, this would already imply all kinds of desirable properties incorporated into the specification. However, in practice this is usually not sufficient; also, safety, robustness, the successful integration of subsystems, etc. will require further verification. More and more, the standardisation initiatives care about standardised verification: With Baseline 4 Release 1, EULYNX has started to provide “certification test cases” for its “subsystems”, i.e. for those

The diagram illustrates a Test Environment. It is divided into two main sections by a vertical red line: the SUT (System Under Test) on the left and the Test Environment on the right. The SUT section contains three blue rounded rectangles labeled A, B, and C. A, B, and C are connected by a horizontal line. A, B, and C are also connected to a horizontal line below them, which then branches into three lines connecting to D, E, and F. D, E, and F are blue rounded rectangles. D is labeled 'Extended' in a red box, E is labeled 'Nominal' in a red box, and F is labeled 'Nominal' in a red box. The Test Environment section contains three blue rounded rectangles labeled G, H, and I. G is labeled 'Reduced' in a red box, H is labeled 'Reduced' in a red box, and I is labeled 'Test Driver' in a red box. A, B, and C are connected to a horizontal line below them, which then branches into three lines connecting to D, E, and F. D, E, and F are blue rounded rectangles. D is labeled 'Extended' in a red box, E is labeled 'Nominal' in a red box, and F is labeled 'Nominal' in a red box. A red line connects the Test Driver (I) to the Test Environment section. A red line also connects the Test Driver (I) to the Test Environment section. A red line also connects the Test Driver (I) to the Test Environment section.

05/08/2026

The test component has been connected, controlled and integrated into the hybrid testbed as an application of the validation approach.

1.2 OVERVIEW OF THE DELIVERABLE

This deliverable consists of ten chapters.

The **first chapter** titled “**Introduction**” introduces the deliverable to the reader, its relation to the project task in the work package of the Europe’s Rail initiative and provides relevant background information.

The **second chapter** titled “**The EULYNX Initiative and Methodology**” provides an insight into the EULYNX consortium, initiative, methodology and specifications along with its integration into Europe’s Rail System Pillar.

The **third chapter** titled “**Reference Model(s)**” describes the baselines and versions of EULYNX models and specification considered in this deliverable.

The **fourth chapter** titled “**Testbed – White Box Approach**” presents the implementation of a hybrid testbed according to the white box approach using a SysML model of EULYNX, a SysML modelling tool, an IoT platform and corresponding hardware. A detailed analysis is conducted to test the implementation, describe the challenges faced and identify the limitations of the white box approach.

The **fifth chapter** titled “**Testbed – Black Box Approach**” presents the implementation of a hybrid testbed according to the black box approach using the model-based requirements specified in the EULYNX documents, a programming language, a graphical user interface (GUI) library, EULYNX telegram communication and remote connection using UDP. This hybrid testbed enables implementations or systems under test (SUTs) to be connected and integrated into it. The hybrid testbed is able to communicate and perform the relevant tests to validate the specified requirements.

The **sixth chapter** titled “**A Versatile Point Test Component**” describes the development of a EULYNX test component, starting from the idea, continuing with the concept, design and implementation and closing with a basic validation of the component. The test component covers the behaviour of the EULYNX point controller as well as of point machines and provides plenty of options for configuration and manipulation. In the context of WP 30, it plays the role of a system under test to be tested using the hybrid testbed (using the black box approach, see seventh, eighth and ninth chapter).

The **seventh chapter** titled “**Testbed – Test Component Integration**” integrates the hybrid testbed with a test component with remote connection, behaviour reference, telegram communication and establishing an interface.

The **eighth chapter** titled “**Test Cases**” describes the test cases derived and executed directly from the model as well as executing traditional test cases that have been developed to validate a field element subsystem.

The **ninth chapter** titled “**Results**” explains the results of the implementation of the hybrid testbed, the test component and the respective test cases for integration.

The **tenth chapter** titled “**Conclusions**” summarises the efforts and accomplishments of the task for the deliverable. The contributions and challenges of the various efforts to develop the validation approach along with the challenges faced are noted. Potential future work that could further develop and progress this approach has also been described.

2 THE EULYNX INITIATIVE AND METHODOLOGY

EULYNX is a European initiative by currently 20 railway Infrastructure Managers (IMs) with the aim of standardising interfaces and elements of railway signalling systems [2]. These IMs form the EULYNX consortium as shown in Figure 3 below.

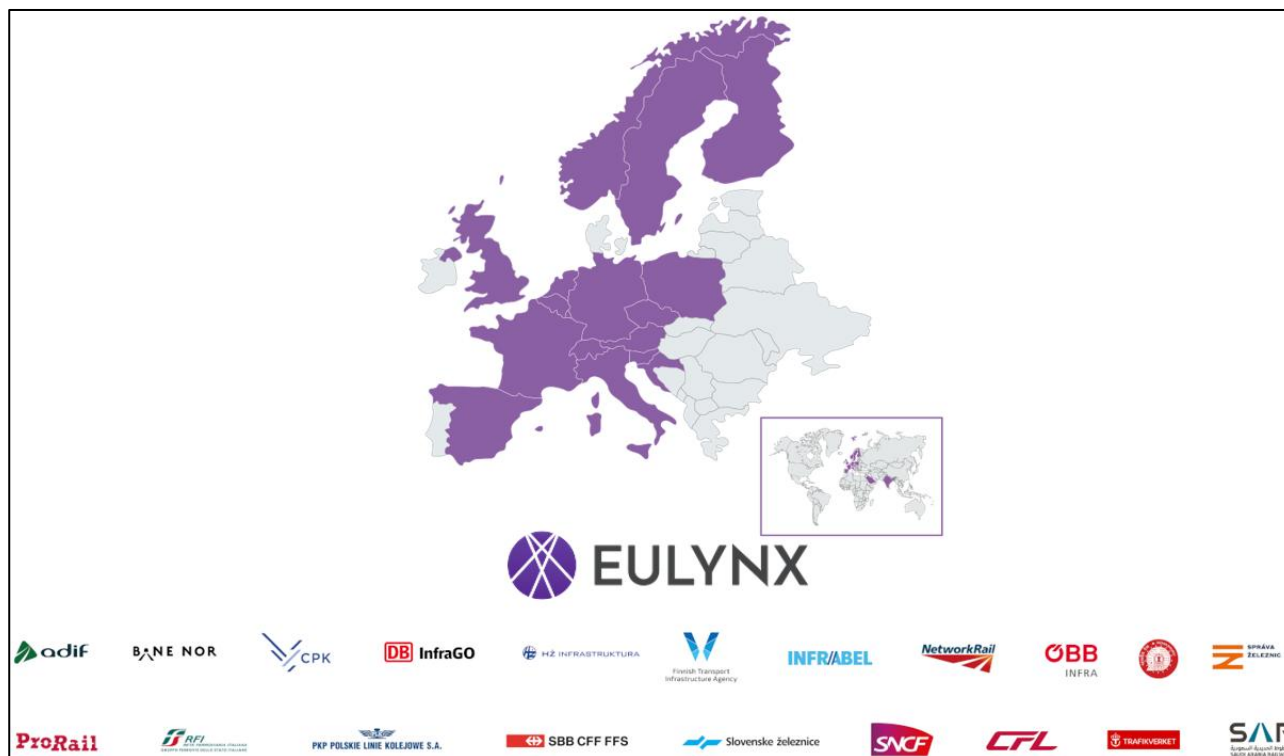


Figure 3: Countries and IMs involved in EULYNX [© EULYNX]

The objectives of EULYNX are:

- Develop a modular standardised signalling architecture.
- Define an international standardised signalling system.
- Provide precise and unambiguous specifications with a focus on interface-centric specifications.
- Reduce the lifecycle cost and the time to market for signalling systems.

2.1 EULYNX SPECIFICATIONS

EULYNX specifications enables modularity and interoperability of railway interlocking and field elements among various suppliers with the standardisation of interfaces, separation of the life cycle of the interlocking core and the field elements and the modular signalling architecture.

EULYNX through its reference architecture is a key driver in the evolution and innovation of the digital railway system. The technical specifications released by EULYNX are characterized by baseline sets. EULYNX Baseline 1 was published in March 2017, Baseline 2 in December 2017, Baseline 3 Release 1 in 2019 and Baseline 4 Release 2 in June 2023. Baseline 3 laid the foundation for standardised railway signalling interfaces and validation tools while Baseline 4 advanced these

efforts by aligning harmonisation and standardisation efforts with the Europe's Rail Joint Undertaking under the EU-Rail System Pillar.

2.1.1 Baseline 3

EULYNX Baseline Set 3 is a comprehensive set of technical specifications for standardising interfaces of railway signalling systems. The EULYNX Consortium, consisting of its international members, published several releases of Baseline Set 3, the last of which was Baseline Set 3 Release 8. Baseline Set 3 covered a wide range of topics including the definition of the EULYNX system, subsystems, adjacent systems, RAMS requirements and the glossary of terms and abbreviations.

As part of its reference SysML model accompanying the specifications, Baseline Set 3 introduced state machines in the EULYNX model for interface validation and simulation, enhancing specification clarity through increased granularity in the respective model-based requirements and validation capabilities. Baseline Set 3 was based on SysML v1.2 for the model-based requirements specification and model-based systems engineering approach. The requirements were documented and stored in DOORS.

2.1.2 Baseline 4

EULYNX Baseline Set 4 continued the development based on Baseline Set 3 and introduced functional packages to delimit required functionality scope in tenders and product development. With the publication of Baseline Set 4 Release 2 under the ERJU System Pillar, EULYNX specifications are aligned with the architecture of the Single European Railway Area (SERA).

As part of its reference SysML model accompanying the specifications, Baseline Set 4 is based on SysML v1.6 for the model-based requirements specification and model-based systems engineering approach. Baseline Set 4 introduced proxy ports and SysML signals to model the communication at the SCI interface. As part of the alignment with ERJU System Pillar, the requirements are documented in DOORS and Polarion.

2.1.3 EULYNX Input Documents

Key input documents are the following EULYNX documents [2]:

- Modelling Standard Eu.Doc.30
- EULYNX System Definition - Appendix A1 Eu.Doc.7_A1
- EULYNX System architecture specification Eu.Doc.16
- Generic interface and subsystem requirements Eu.Doc.20
- Generic interface and subsystem requirements for SCI Eu.Doc.119
- Requirements specification for subsystem Point Eu.Doc.36
- Interface definition SCI Eu.Doc.92
- Interface specification SCI-Generic Eu.Doc.93
- Interface specification SCI-P Eu.Doc.38

2.2 EULYNX MBSE

According to INCOSE, Model-Based Systems Engineering (MBSE) is a formalised utilisation of modelling to support SE activities such as system requirements, design, analysis, verification and validation starting with the conceptual design phase and continuing throughout the development and subsequent lifecycle phases [6].

The EULYNX initiative aims to specify EULYNX subsystems and standardise the interfaces (SCI, SMI, SDI) and the interfaces between adjacent systems. Command control and signalling (CCS) systems such as EULYNX subsystems are reactive control systems [7] and most of them safety critical [8]. They are distinguished by the continuous interaction and synchronisation between the system and its surroundings. The EULYNX Architecture Framework consists of an analysis model that describes the overall system architecture and a SysML specification model defining tender requirements of system elements and interfaces. The SysML Specification Model is based on the EULYNX MBSE Specification Framework (MBSE SF – Methodology) which contains the EULYNX AM MBSE process [2]. The AM MBSE, which concentrates on specification tasks to be carried out at the infrastructure manager (IM) side, enables the description of a SUS or a SIUS from various viewpoints capturing various stakeholder concerns and with varying degrees of granularity (different abstraction levels).

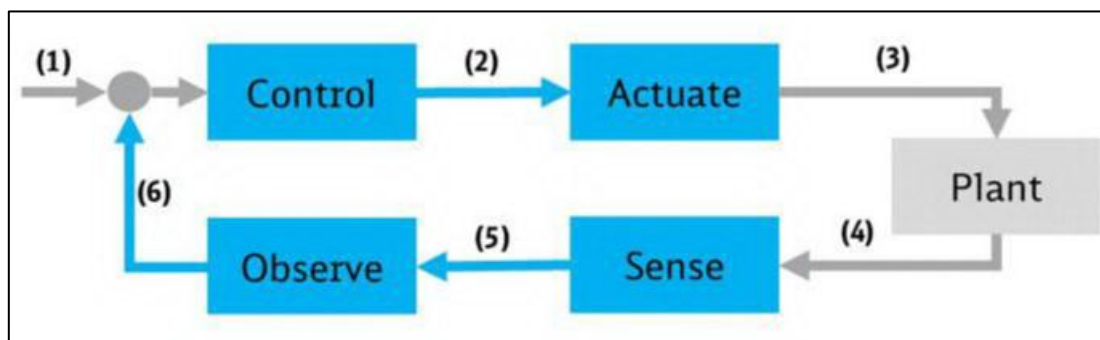


Figure 4: Typical control loop of a EULYNX subsystem [© EULYNX]

EULYNX applies MBSE to generate the model-based requirements of its respective railway signalling subsystems. EULYNX delivers validated specification models. The specification models are executable and therefore can be tested to ensure that the behaviour of the models meets the needs of the stakeholders. These specification models are transferred to the suppliers to manufacture the signalling equipment. By developing the respective specifications, standardising the interfaces of the interlocking core will be possible thereby separating the life cycle of the interlocking core from the field elements. With EULYNX, signalling equipment can be replaced without the need for a complete renewal [2].

2.2.1 Viewpoint

A viewpoint is a specification of a view that should be constructed and used. Viewpoints establish a view's objective, target audience and methods of development and analysis which serve as patterns or templates from which individual view can be developed.

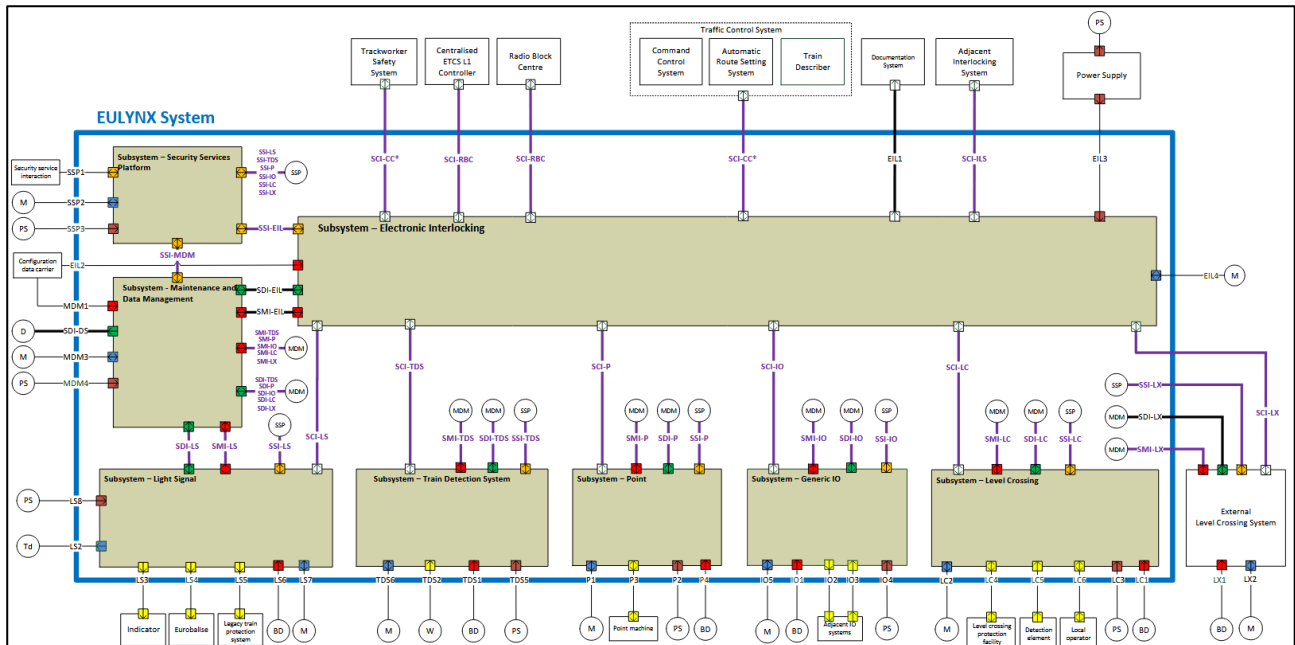


Figure 5: EULYNX System Definition [© EULYNX]

2.2.2 Abstraction Level

The SUS/SIUS is examined at a particular level of abstraction and granularity, which is defined as an abstraction level. An underlying structural trait from the layer above determines the level of detail i.e., the level of granularity of the respective abstraction level. The SUS/SIUS is initially considered as a whole. In other words, an abstraction level denotes the degree of complexity with which a SUS/SIUS is viewed and describes the whole SUS/SIUS under a certain level of abstraction. The higher the abstraction level, the less detail is present. Any abstraction level consists of various appropriate views. A low degree view is refined into a series of more specific SUS/SIUS views using the principle of divide and conquer in order to increase the level of granularity for a particular view. Basically, any viewpoint can be used to complete this step.

2.2.3 View

A view is a description of the SUS/SIUS from a specific viewpoint and with a specific degree of granularity. Within the scope of the Modelling Standard, a view is synonymously referred to a view, model view or system view.

2.2.4 Engineering Path

An engineering path summarises the development of views for a SUS or SIUS with a specific degree of granularity. The AM MBSE enables the seamless, model-based specification of digital CCS in the railway domain with three primary IM-related viewpoints:

- Functional Viewpoint
- Logical Viewpoint
- Technical Viewpoint

Following EN 50126 [9], the AM MBSE consists of three core IM-related abstraction levels (AL) namely:

- **AL1:** Subsystem/Interface Definition
- **AL2:** Subsystem/Interface Requirements
- **AL3:** Apportionment of Subsystem/Interface Requirements

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 given below:

- **AL1:** System Definition
- **AL2:** System Requirements
- **AL3:** Apportionment of System Requirements

Crosscutting system properties (CSP): Continuous engineering of crosscutting system properties is one of the principles of the EULYNX AM MBSE. This principle intends to establish the ability to consider crosscutting properties of the SUS/SIUS. RAMS, security and real-time properties of the SUS/SIUS are examples of typical crosscutting properties, they must be taken into account in any engineering activity and the respective artefacts. Nearly every process stage in the development lifecycle is impacted by safety, which is often defined as the absence of harm or unacceptable risk. Due to this, safety is not portrayed in a single viewpoint but rather as a qualitative component of the EULYNX AM MBSE that is integrated into several viewpoints and has a crosscutting influence. The complexity of safety-critical railway systems is growing, which makes safety analysis models more difficult. Therefore, it is not reasonable to develop functionality and consider safety in separate tasks. Safety aspects have to be tightly integrated into the development process and its models [2]. The EULYNX MBSE Methodology is shown in Figure 6.

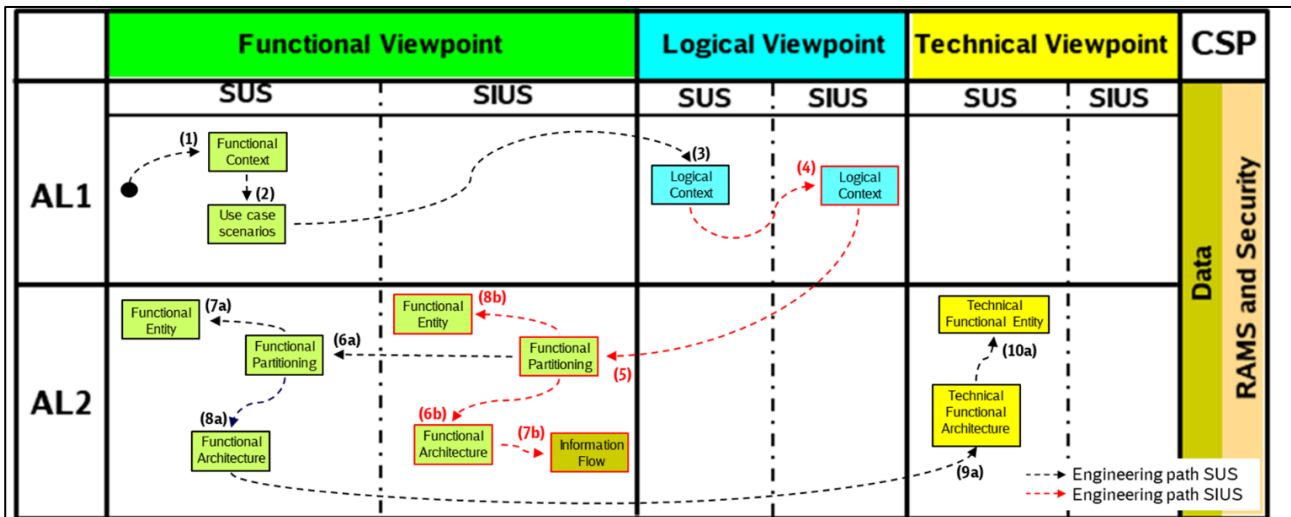


Figure 6 : EULYNX MBSE Methodology [© EULYNX]

2.2.5 SysML

The Systems Modelling Language (SysML) is a general-purpose modelling language for systems engineering applications. With SysML, a wide variety of systems and systems-of-system can be specified, analysed, designed, verified and validated. Hardware, software, information, processes, personnel or facilities could comprise these systems. SysML is a facilitator for Model-Based Systems Engineering (MBSE), and since 2006 has been managed by the Object Management Group (OMG) [10].

SysML is based on UML and involves modelling blocks instead of modelling classes, thus providing vocabulary that's more suitable for systems engineering [11]. SysML is an extension of UML and is defined as a UML profile. A UML profile is a UML dialect which in the case of SysML is called the SysML profile that uses the techniques of stereotypes, tagged values, blocks and constraints to tailor UML to SysML. The INCOSE and OMG were joined by industry, modelling tool vendors, universities and organizations to define systems modelling language SysML.

SysML is an expressive graphical modelling language that facilitates the visualisation and communication of the important aspects of a system's design, such as structure, behaviour, requirements [12]. SysML can be used as the first of the three pillars (Language Pillar) of MBSE. There are nine types of SysML diagrams as listed in Figure 8 which are divided into three categories that are structure, behaviour and requirements.

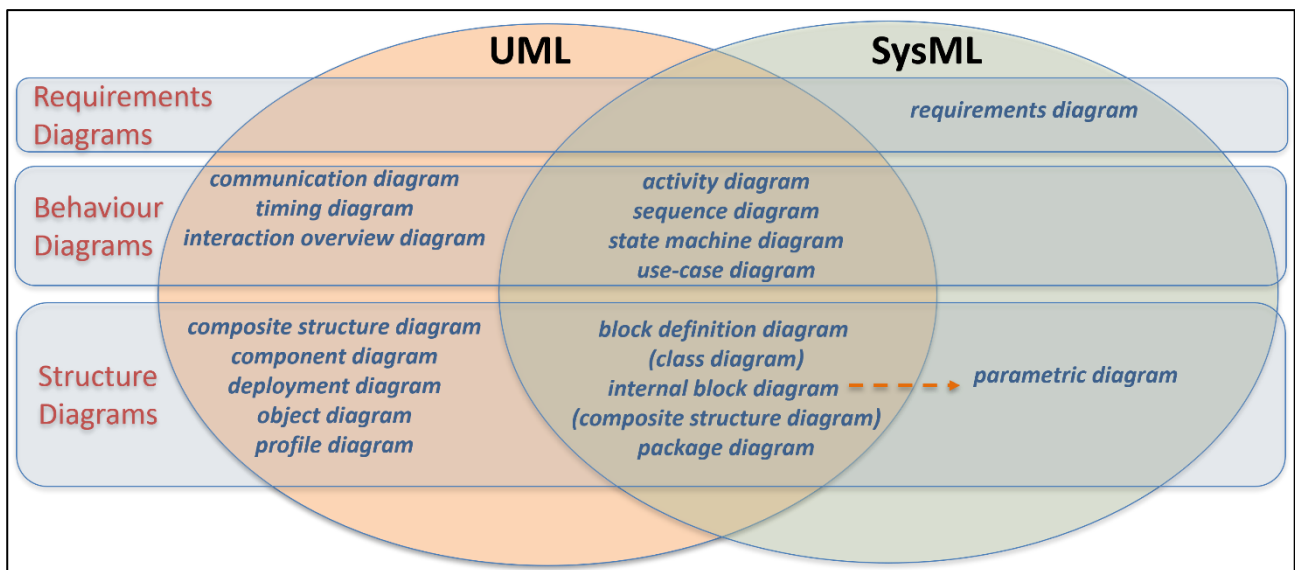


Figure 7: Relationship between SysML and UML and their respective diagrams

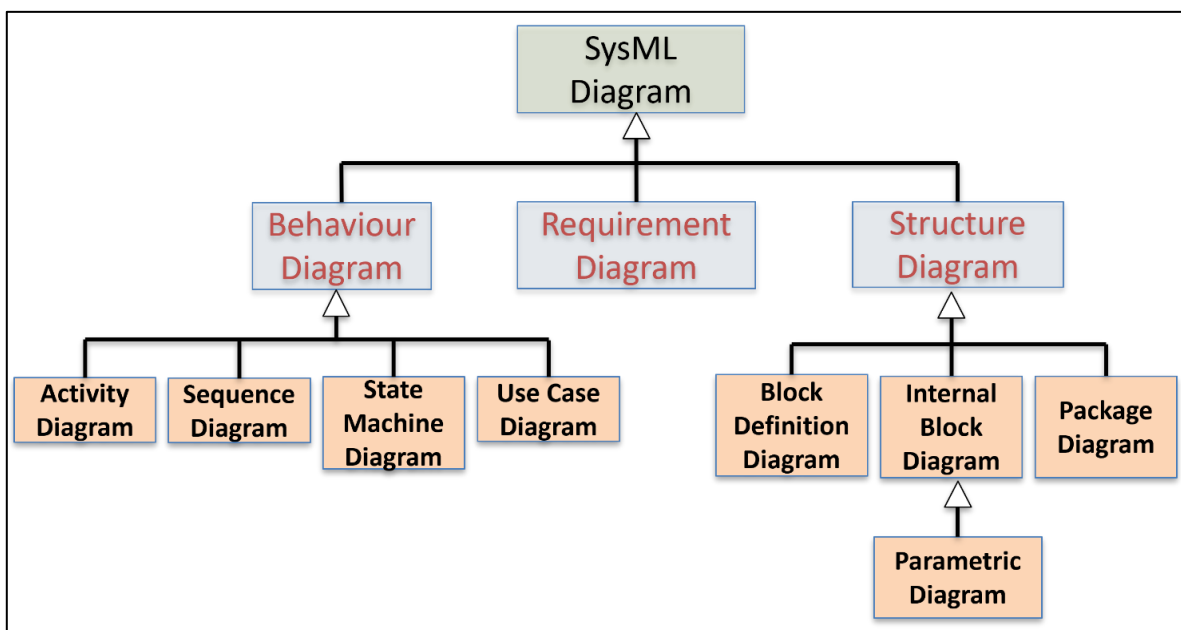


Figure 8: SysML Diagrams

3 REFERENCE MODEL(S)

The main input and reference for the work carried out to produce this deliverable are the reference models released by EULYNX accompanying their specifications.

According to *ISO/IEC 81346* [14], a reference model provides a common architecture that enables cross-functional teams to collaborate effectively within their workflows. A reference model allows various disciplines to maintain their objectives and processes while still working according to the model which describes the system. The validation approach utilises two reference models from EULYNX as the basis for X-in-the-Loop testing. These are:

- **EULYNX Baseline 3 Release 8 (BL3R8)** for the white box approach development of the hybrid testbed. EULYNX BL3R8 uses SysML v1.2 and flow ports to exchange information at the interfaces. EULYNX BL3R8 was utilised for the white box approach due to the availability of support documents for developing a model-driven solution. The state machines specified were incorporated and executed in the hybrid testbed.
- **EULYNX Baseline 4 Release 2 (BL4R2)** for the black box approach development of the hybrid testbed and for the development of the test component. EULYNX BL4R2 uses SysML v1.6 and proxy ports in combination with flow ports to exchange information at the interfaces. EULYNX BL4R2 was utilised for the black box approach due to the increase in detail and maturity of the specifications which enabled a comprehensive development of the hybrid testbed and test component according to the model-based requirements specified.

The focus for EULYNX BL3R8 was the completion and stabilisation of the EULYNX Baseline 3 specifications. The focus for EULYNX BL4R2 was increased maturity of the specifications and alignment with Europe's Rail System Pillar.

A reference model helps reduce engineering effort required to achieve modularity and interoperability of components and interfaces. The various parts of the validation approach have a common core of requirements, concepts and semantics with a reference model. The validation of EULYNX specification is enabled by using the EULYNX SysML model as a reference input for the development of the X-in-the-Loop testing validation approach based on the hybrid testbed.

3.1 SysML MODELLING IN EULYNX

SysML makes it possible to generate system specifications in a single language for heterogeneous teams and helps to visualise and simulate the system hardware and software blocks. Knowledge is thereby captured through models stored in an available repository, enhancing communication throughout all teams. In the long term, blocks can be reused as their specifications and models enable suitability assessment for subsequent projects [11]. SysML consists of graphical constructs to represent textual requirements and relate them to the other modelling elements. SysML only uses nine different diagrams, and thus, it is easier to understand, use and apply. Because of its simplicity and easy applicability, SysML has gained more popularity in different fields such as railway aerospace, defence and automotive industries. The following subset of SysML is used in EULYNX:

3.1.1 Block

A "Block" is a modular unit of structure of SysML [11] that is used to define the physical as well as conceptual entities or logical abstraction of a system. A block in SysML is the fundamental structural

element. A block can be modelled to represent any kind of entity, whether it exists inside the system of interest or in the external environment of the system [12].

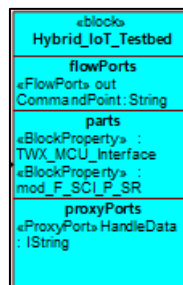


Figure 9: HTB represented by a SysML block

Interface blocks are specialisation blocks of SysML which has no behaviour and SysML provides a different concept for Interface Blocks:

- Ports: Ports are the points at which the external entities can connect to or interact with the block in different or limited ways instead of connecting with it directly.
- Flow Specification: Flow specification uses flow properties to define what can go through blocks, whether it is data, matter or energy.

3.1.2 Port

A port is a type of property that specifically designates a location at the boundary of a structure where external entities can interact with it, either to provide or request a service or to exchange data, matter or energy. Based on the version of SysML, there are different types of ports:

Flow Port (SysML v1.2): A flow port represents the different kinds of data, matter or energy that can flow in or out of a block at an interaction point on the boundary of the block.

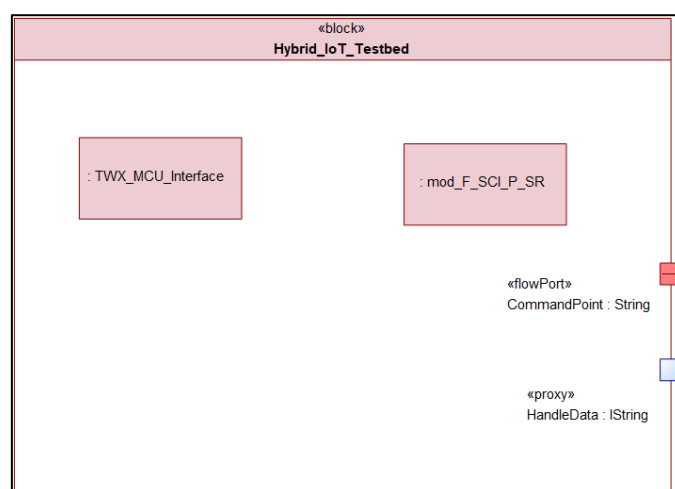


Figure 10: IBD of a block containing a proxy port and a flow port

Proxy Port (SysML v1.6): A proxy port represents the external interface of the block that owns it. It represents the section of behavioural and structural attributes of the owning block or its part properties that are accessible to external blocks. A *proxy* port is typed by a specific kind of block called an interface block as described above which is denoted by the <<interfaceBlock>> stereotype.

An interface block specifies the section of behaviour and structural attributes of a block that are accessible at a particular proxy port.

Figure 11 shows the compartments that show the sub elements of the block. These compartments enable the quick identification of the different properties a block holds. The compartments also assist to see what requirements a block satisfies or checks how different types are related to each other.

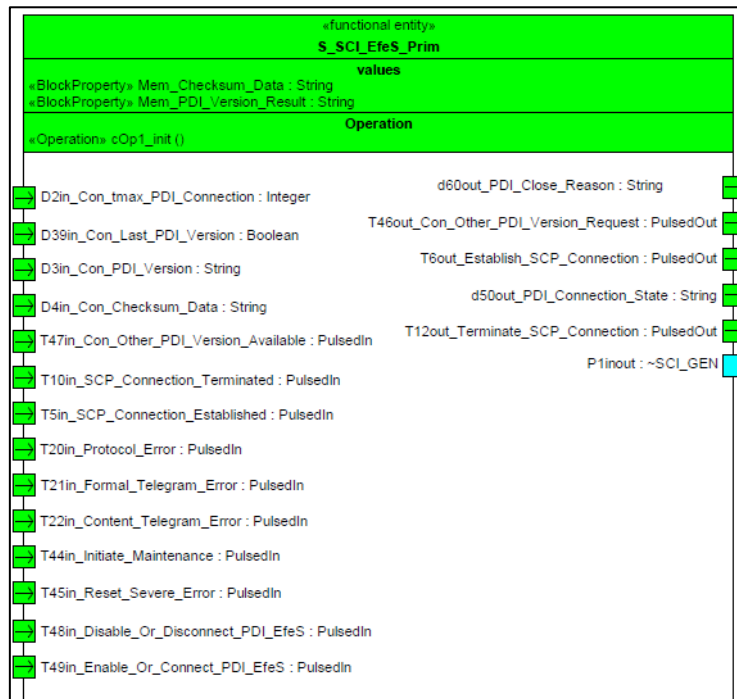


Figure 11: Example of an EULYNX SysML block (Baseline 4)

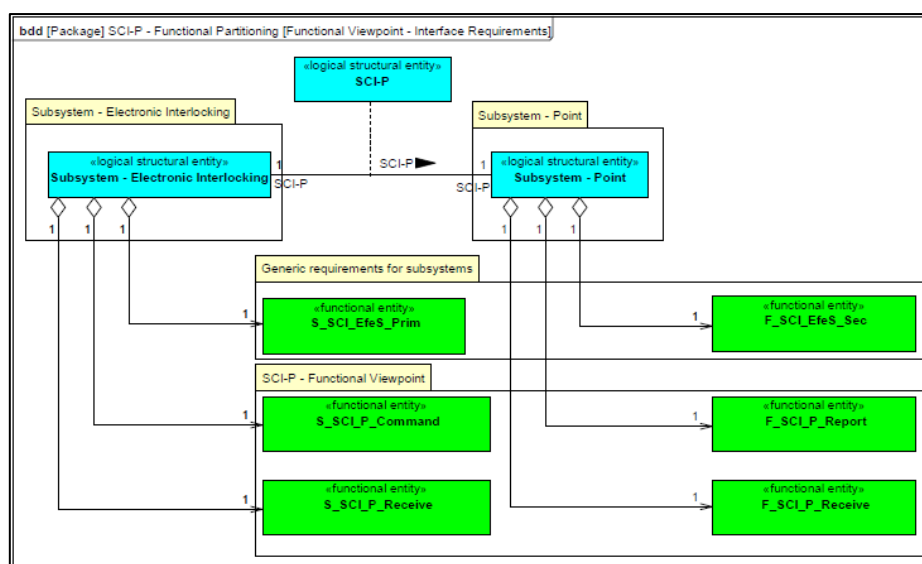


Figure 12: SysML Block Definition Diagram

3.1.3 Block Definition Diagram (BDD/bdd)

The block definition diagram is used to show the relationships between elements, such as blocks and value types which specify the types of items that can exist in an operational system and to show

those elements individually. Block definition diagram is a black box representation of the main system along with its composite blocks in a hierarchical way. BDD can include blocks of any type such as hardware or software (Figure 12).

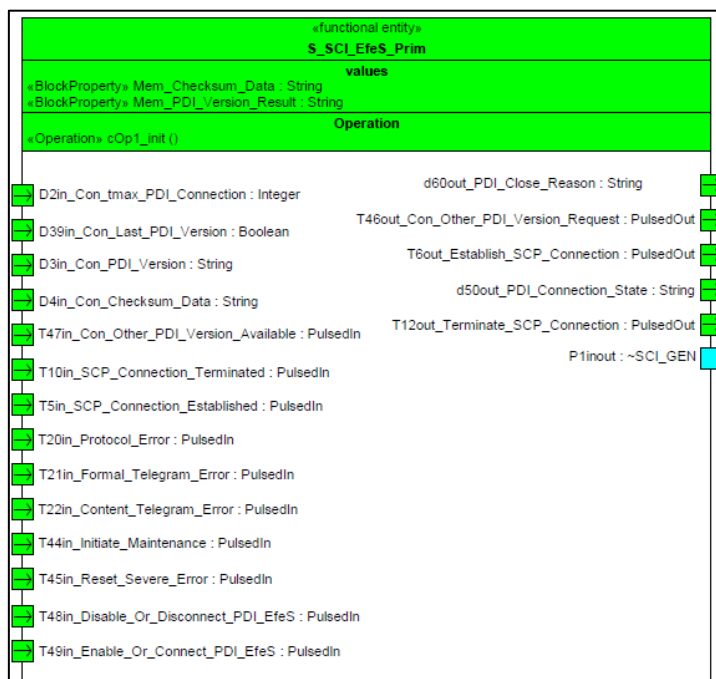


Figure 13: SysML internal block diagram

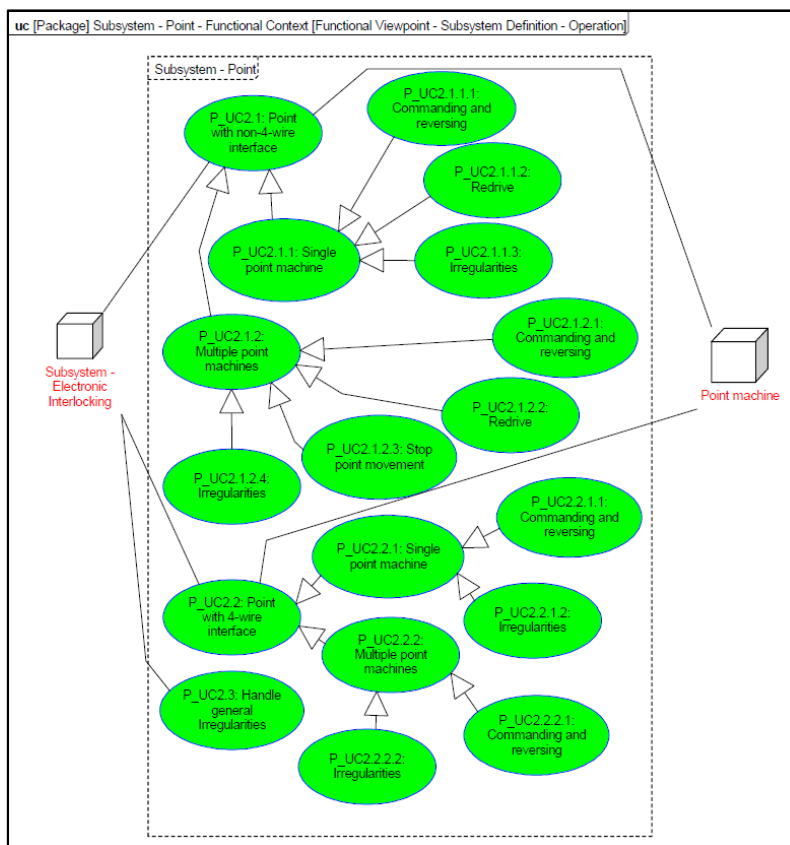


Figure 14: SysML use case diagram

3.1.4 Internal Block Definition (IBD/ibd)

The internal structure of a single block is specified using the internal block diagram. An *ibd* especially displays the connections and interfaces between the internal components of a block. IBD provides a white box or an internal view of the system block. It is instantiated from the Block Definition Diagram to represent the final assembly of all blocks within the main system block. Figure 13 illustrates the Internal Block Diagram.

3.1.5 Use Case Diagram (uc)

The use case executed by a system and the actors who invoke and part in them are represented in the use case diagram. A use case diagram is a black box representation of the services that a system conducts in conjunction with its actors. Use case diagrams are intended to identify the actors and use cases from a usability perspective i.e. actors/system interactions [13]. Figure 14 shows a SysML use case diagram.

3.1.6 Sequence Diagram (seq/sd)

The sequence diagram is used to specify behaviour, with an emphasis on how the operation calls and asynchronous signals are used by block components to interact with each other. Sequence diagrams are frequently used as a thorough design tool to accurately specify a behaviour as an input to the development stage of the system life cycle. Sequence diagrams are also a useful tool for specifying test cases. A sequence diagram provides the interaction between the objects and actors involved in the given scenario as well as the sequence of the messages between the objects and actors to carry out the functionality given in the scenario. Figure 15 shows a SysML sequence diagram.

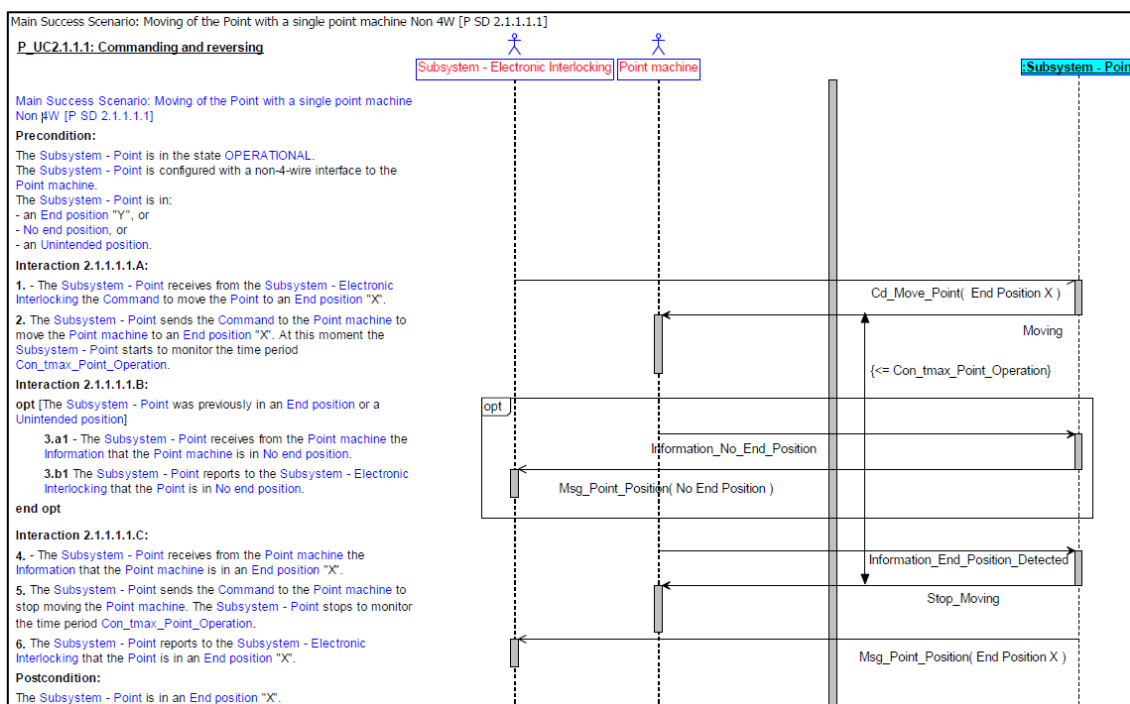


Figure 15: SysML sequence diagram

3.1.7 State Machine Diagram (stm)

With an emphasis on the set of states of a block and potential transitions between those states in response to event occurrences, the state machine diagram is used to specify behaviour. Similar to a sequence diagram, a state machine diagram rigorously specifies a block's behaviour and can be used as an input during the development stage of the system life cycle. A state machine diagram models the behaviour of systems, providing a way to specify a system lifecycle that its respective subsystems and system states comply with. A lifecycle of system states is a sequence of events that a system goes through during its lifetime in response to the events and the conditions that define the state transitions. Figure 16 shows a SysML state machine diagram.

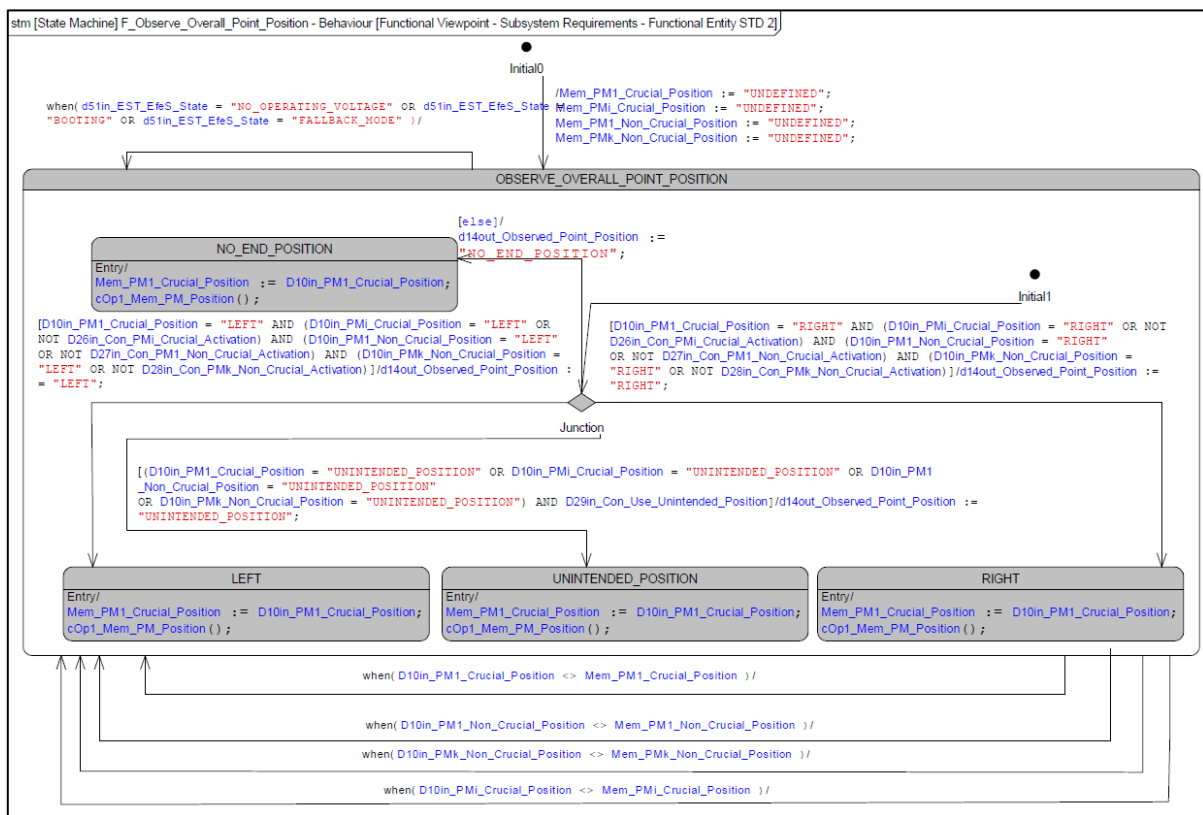


Figure 16: SysML state machine diagram

3.1.8 Connector

A connector between two properties on an IBD indicates that the two structures will be able to access one another inside a correctly assembled and operational system. The connector can be attached to compatible ports rather than the properties themselves if the two connected properties have ports. Doing this would indicate that the properties are connected at specific locations on their boundaries for interacting with each other.

3.1.9 State

During system operation, a system or a subsystem can exist in a defined set of states.

3.1.10 Transition

A transition represents the change of a state to another state. The change of a state to itself is called a self-transition.

3.1.11 Use Case

A use case is a service, action or behaviour that the system will perform.

3.1.12 Actor

An actor is any external system or person that interacts with the system of interest.

3.1.13 Interaction

An interaction is an emergent behaviour produced when components of a block interact with one another through operation calls and asynchronous signals.

3.1.14 Operation

An operation represents a behaviour that a block performs when called.

3.1.15 Signal

Signal is a model element used to represent data, matter or energy that one component of a system sends to another component typically with the intention of triggering a behaviour on the receiving end. Much like a block, a signal can have attributes. The attributes of a signal represent data that the signal transmits from a client to a target.

3.1.16 Stereotype

A stereotype defines a new type of model element by applying properties, semantics or constraints to an existing model element [12].

4 TESTBED – WHITE BOX APPROACH

The white box approach for the hybrid testbed, as visualised in Figure 17, focuses on the integration of a EULYNX SysML model with a hardware model. The Subsystem Point, a railway signalling subsystem which is a part of the EULYNX system, is selected as the case study for the white box approach. The field side of the Standard Communication Interface-Point F_SCI-P is the basis of the case study implementation and analysis. The hybrid testbed developed connects a EULYNX SysML model of F_SCI-P to the prototype hardware. Validation is done through the two components of the hybrid testbed by testing the functioning of the hardware against the executed state machines and simulation of the model. Checking for conformity is ensured by the successful validation of the corresponding model-based requirements in the EULYNX model according to the case study [15].

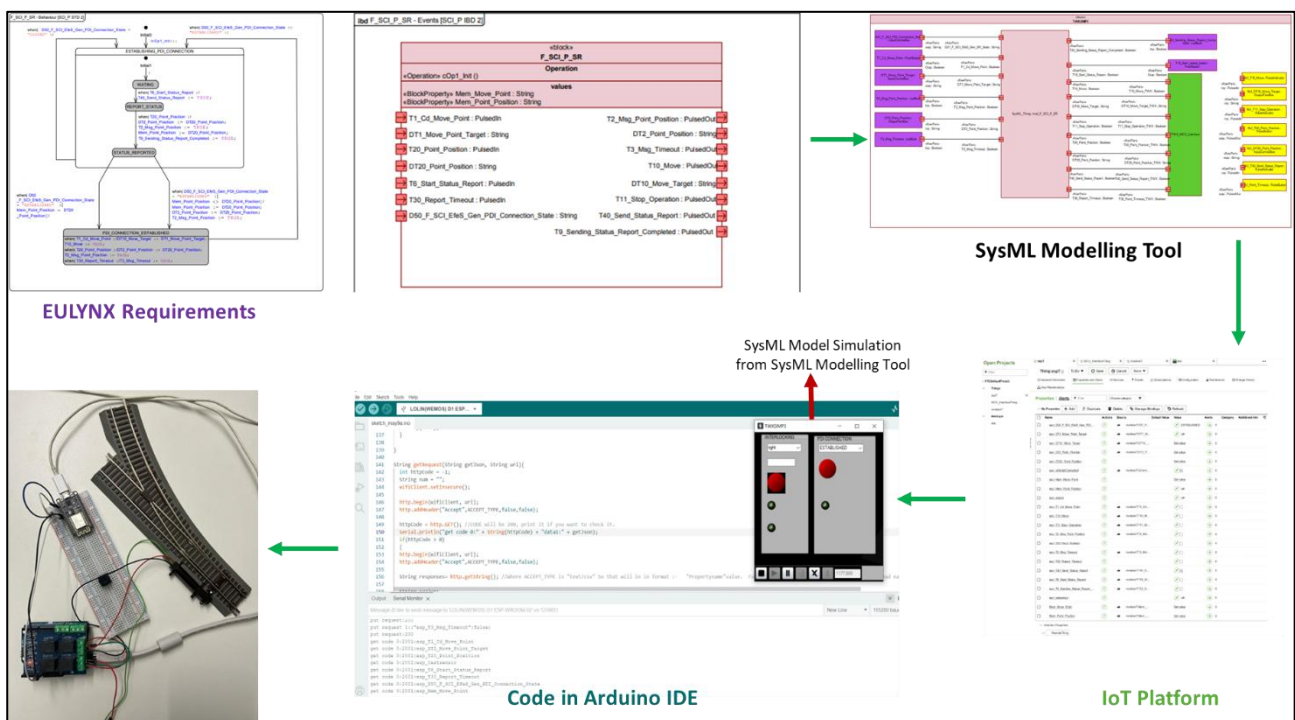


Figure 17: Hybrid Testbed – White Box Approach Setup

4.1 TESTBED DEVELOPMENT – SYSML HARDWARE INTEGRATION

4.1.1 Languages and Tools

The languages that are a part of the testbed are:

- SysML v1.2
- Visual Basic
- C Programming Language

The tools that form the toolchain used for the testbed are:

- PTC Windchill Modeler 9.6 [37]
- Visual Studio 2019

- PTC ThingWorx 9.3.8 [38]
- Arduino IDE 2.1

4.1.2 Model Simulation

The SysML model forms the basis of the virtual prototype (see Figure 18) and needs to be simulated to check the requirements and behaviour defined in the model are correct. Therefore, the SysML model i.e., the functional entity F_SCI-P is simulated using the PTC Windchill Modeler SySim model execution tool [36]. The virtual prototype is a simulator which is then simulated by SySim, which automatically generates Visual Basic code that is then executed to simulate the virtual prototype.

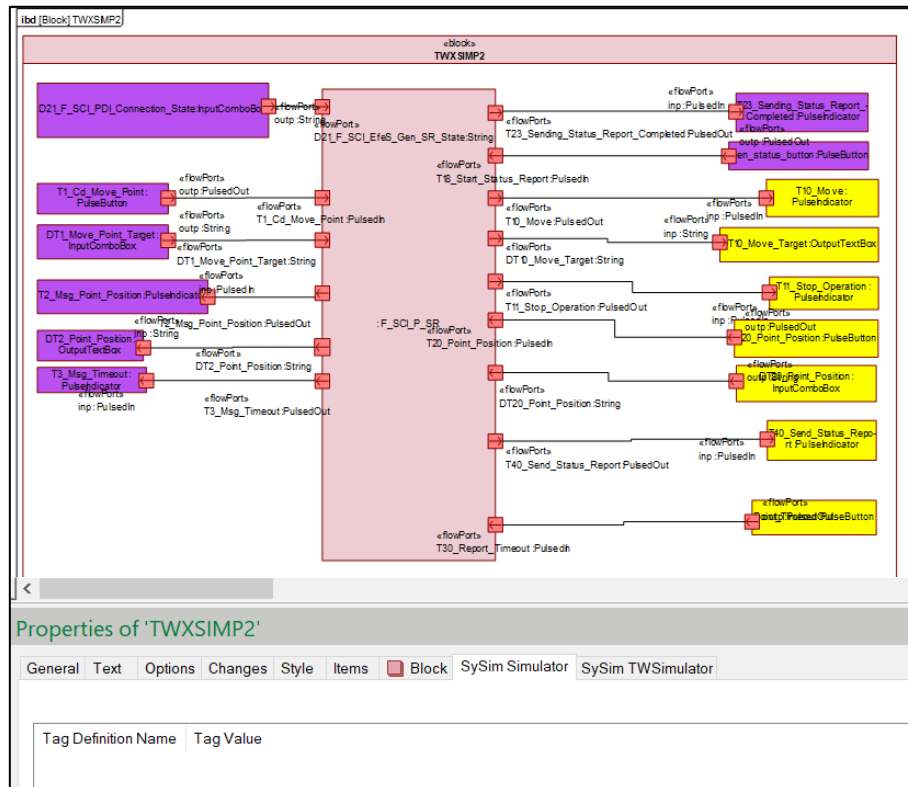


Figure 18: Virtual Prototype Simulator

The simulation of the SysML model consists of a SysML block, a SysML internal block diagram and application of SySim stereotypes. SySim prepares an executable from the stereotyped model by auto-generating Visual Basic code from the SysML model and the ASAL (Atego Structured Action Language) [16] code written in the transitions of the state machine, which is then built and run in Visual Studio to perform the simulation.

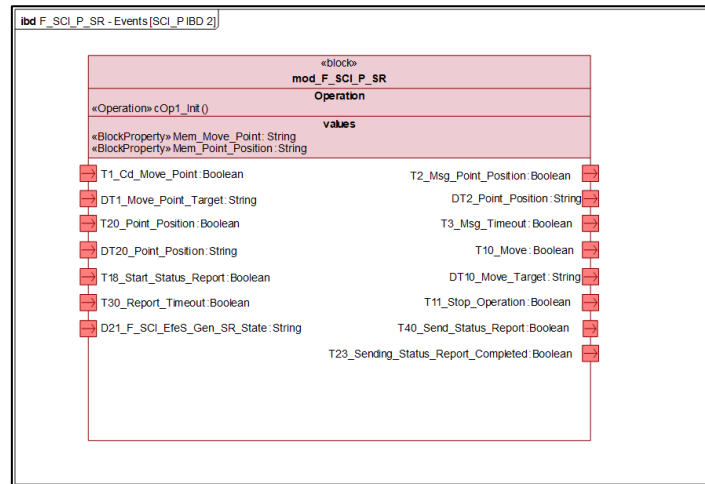


Figure 19: ibd of Functional Entity for simulation

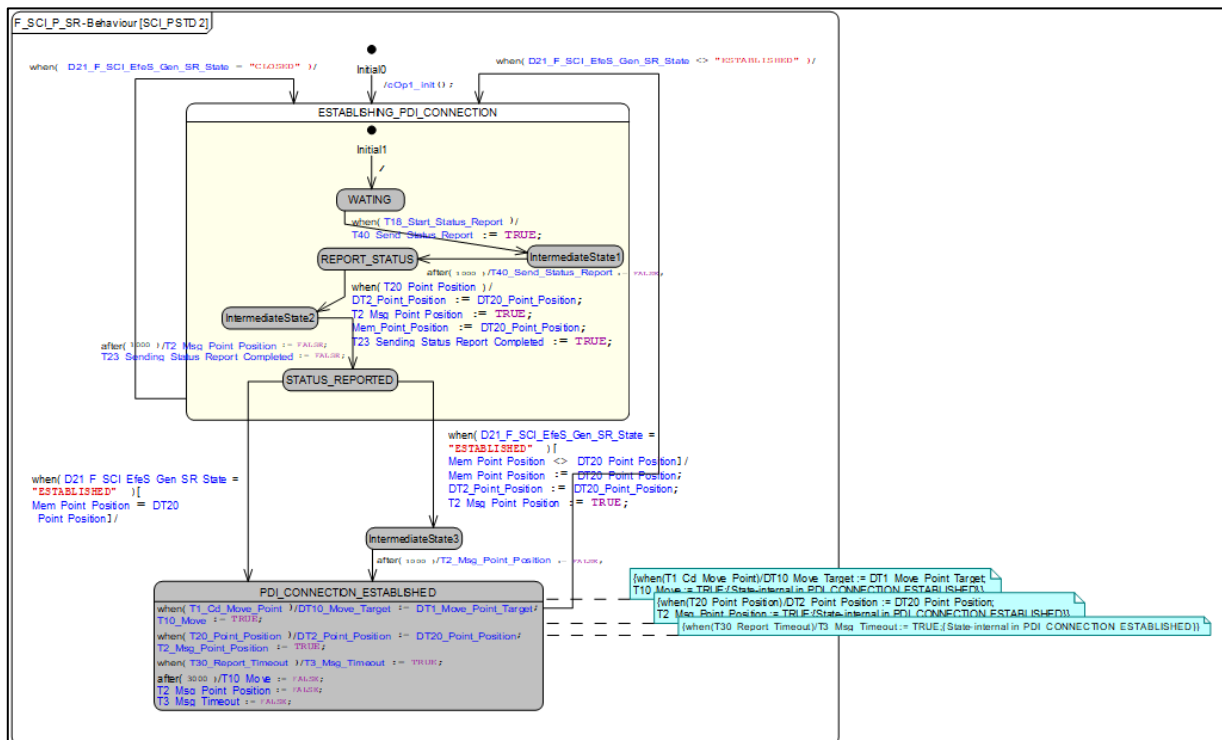


Figure 20: stm of Functional Entity for simulation

Before starting the simulation, the state machine of the SysML model is run in PTC Windchill Modeler which provides real-time state machine animation that checks the transitions and states according to the simulation. Most of the transitions which enable the change of states in the SysML model are EULYNX requirements. The simulation is concerned with the System Interface under Specification (SIUS).

4.1.3 Model – IoT Platform Connection

The SysML model is connected to the IoT platform PTC ThingWorx using the ThingWorx profile in PTC Windchill Modeler and by applying SySim ThingWorx stereotypes to the simulation. The SysML model and other relevant SysML elements for the model are integrated into the hybrid testbed. With

this, the values at the ports and the block properties of the functional entity will be shown as the properties of the SysML model on the IoT platform and will update in real-time according to the executed simulation. Figure 21 shows simulation of virtual prototype.

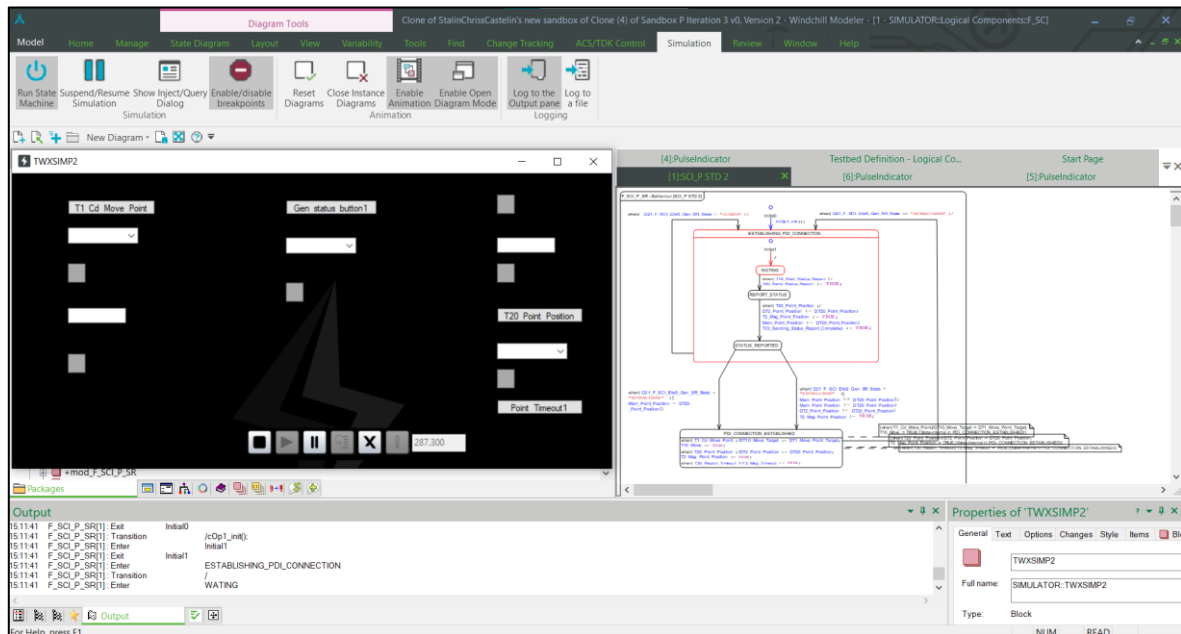
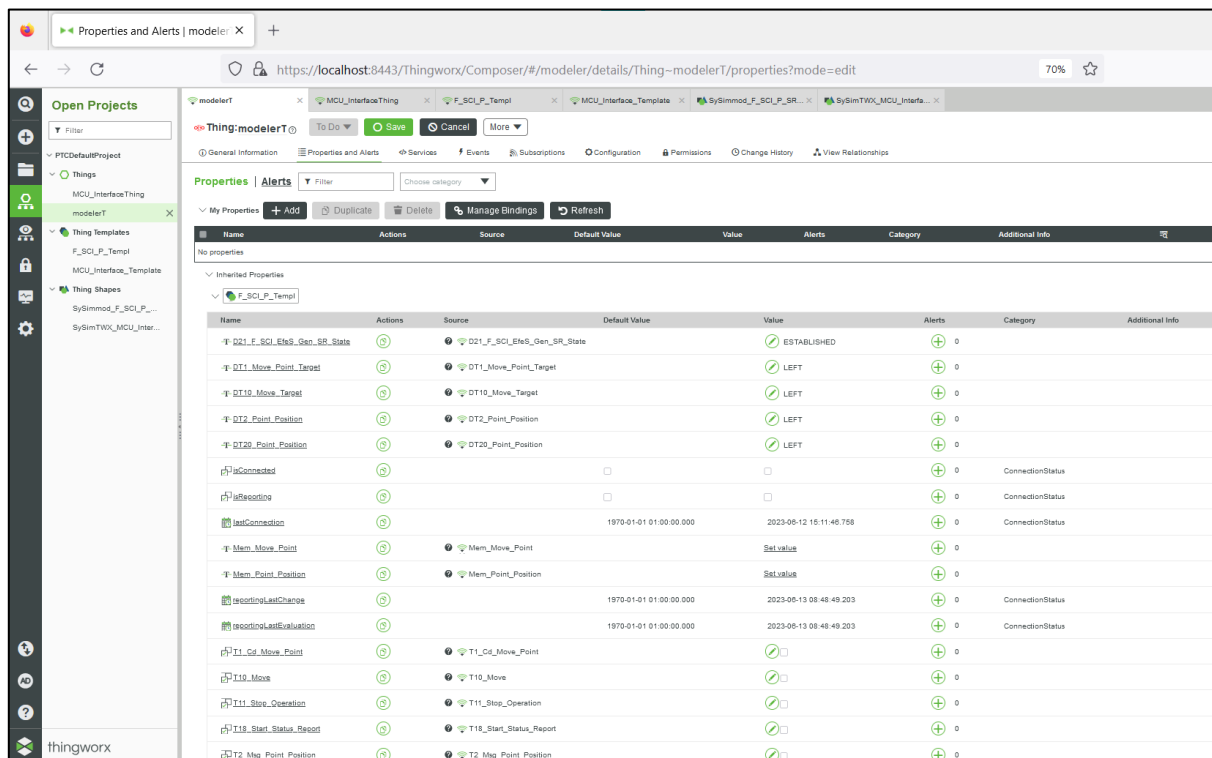


Figure 21: Simulation of virtual prototype



Name	Actions	Source	Default Value	Value	Alerts	Category	Additional Info
No properties							
Inherited Properties							
F_SCI_P_Templ							
F_SCI_Elts_Sr_State	⊞	⊞ D21_F_SCI_Elts_Sr_State		⊞ ESTABLISHED	⊞ 0		
DT1_Move_Point_Target	⊞	⊞ DT1_Move_Point_Target		⊞ LEFT	⊞ 0		
DT19_Move_Target	⊞	⊞ DT19_Move_Target		⊞ LEFT	⊞ 0		
DT2_Point_Position	⊞	⊞ DT2_Point_Position		⊞ LEFT	⊞ 0		
DT29_Point_Position	⊞	⊞ DT29_Point_Position		⊞ LEFT	⊞ 0		
isConnected	⊞		⊞	⊞	⊞ 0	ConnectionStatus	
isReceiving	⊞		⊞	⊞	⊞ 0	ConnectionStatus	
lastConnection	⊞		1970-01-01 01:00:00.000	2023-05-12 15:11:46.758	⊞ 0	ConnectionStatus	
Mem_Move_Point	⊞	⊞ Mem_Move_Point		⊞ Set value	⊞ 0		
Mem_Point_Position	⊞	⊞ Mem_Point_Position		⊞ Set value	⊞ 0		
receivingLastChange	⊞		1970-01-01 01:00:00.000	2023-05-13 08:48:49.203	⊞ 0	ConnectionStatus	
receivingLastEvaluation	⊞		1970-01-01 01:00:00.000	2023-05-13 08:48:49.203	⊞ 0	ConnectionStatus	
T1_Cd_Move_Point	⊞	⊞ T1_Cd_Move_Point		⊞	⊞ 0		
T19_Move	⊞	⊞ T19_Move		⊞	⊞ 0		
T11_Stop_Operation	⊞	⊞ T11_Stop_Operation		⊞	⊞ 0		
T18_Start_Status_Report	⊞	⊞ T18_Start_Status_Report		⊞	⊞ 0		
T2_Move_Point_Position	⊞	⊞ T2_Move_Point_Position		⊞	⊞ 0		

Figure 22: SysML model integrated into hybrid testbed

4.1.4 Hardware Development

The hardware components are connected using wires, breadboards, perfboards, a printed circuit board (PCB), jumper wires, relays, microcontrollers, resistors, buttons, LEDs, motors etc. for the considered railway signalling subsystem point of the EULYNX system. A program is written on the Arduino IDE to enable the working of the subsystem. The program is then flashed onto the microcontroller and the hardware setup is tested to check the functioning of the subsystem. Figure 22 shows SysML model integrated into hybrid testbed.

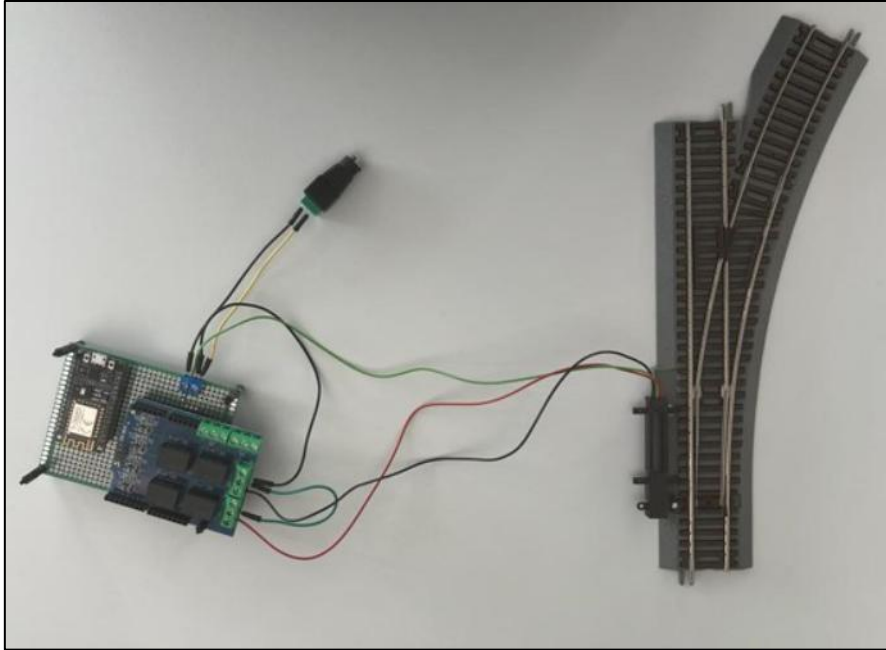


Figure 23: EULYNX-based hardware prototype for Subsystem Point

4.1.5 Hardware – IoT Platform Connection

The state machine of the functional entity is programmed in C and the properties are defined as variables in the Arduino code. The necessary settings for communication with the IoT platform over the REST API are programmed in the Arduino IDE. The hardware developed is setup on the IoT platform with properties that relate to the variables in the program that is to be uploaded to the microcontroller. The developed program is then flashed onto the microcontroller. With this, the hardware is integrated into the hybrid testbed. Figure 23 EULYNX-based hardware prototype for Subsystem Point.

4.1.6 Hybrid Testbed Integration

The SysML model and hardware are integrated into the IoT platform, the communication and control between the SysML model and hardware are checked and the properties of both components are checked if they are updating in real-time and providing output. A *mashup* (see Figure 24) is then developed to reflect the status and control of both components on the IoT platform and to make the testbed user-friendly. Upon completion of the mashup and integration of SysML model and hardware, the hybrid testbed is completed, developed and ready to be deployed to test and validate EULYNX requirements of the concerned railway signalling subsystem point.



Most of the contents of the functional entity are EULYNX requirements. These are specified model-based requirements necessary to realise the EULYNX subsystem point. These requirements are documented in the relevant EULYNX interface specification document. With the hybrid testbed, a

test case is derived for each requirement of the function entity through model traversal which is then tested. Each successful test case means that the corresponding EULYNX requirement is validated. Figure 25 shows Integrated hybrid testbed.

The test cases and testing of the model requirements for subsystem point using the white box approach of the hybrid testbed is covered in Chapter 8 titled, “Test Cases”.

4.2 TESTBED DEVELOPMENT – SYSML VIRTUAL SAMPLER INTEGRATION

The standard communication interface (SCI) established when a subcomponent from the Electronic Interlocking Subsystem and a subcomponent from Subsystem Point are connected via their respective ports and the flow of information between them is determined by the triggers of the state machines belonging to each of the subsystem components respectively.

This working description helps narrow the area of the models that are essential for the development of the HTB. However, the description does not sufficiently define the area of focus. Within the model, there are several pairs of subsystem components that can be chosen that fit the description above [17].

A system under test needs to be defined which is narrow enough for an effective investigation while simultaneously wide enough to accommodate critical aspects of the model. The first step in narrowing down the scope is identifying the correct mode of operation. The Subsystem Point operates in either of the three modes, Initialisation, Operation and Maintenance. Each mode determines which standard interface is used.

The mode that is considered is the operation mode which uses the SCI-P. In this mode the Electronic Interlocking Subsystem and the Subsystem Point have established a connection on application layer, and the safety protocols have already been completed. The information exchanged is for the specific purpose of achieving a CCS directive that influences the physical state of the point blades. The entities involved in this mode like the SysML blocks representing Subsystem Points, state machine diagrams, information flows, use cases and sequence diagrams are considered to be the part of the SCI-P to be validated by the HTB.

The execution of several use cases is possible in Operation mode. The use case Command Point and its respective use case scenario (sequence diagram) is considered because it addresses the fundamental objective of the Subsystem Point which is to achieve change of position of the point blades. If a validation test for this use scenario is possible, it stands to reason that validation of the other use case scenarios is also possible since the other 6 scenarios are all derivatives of this main scenario and the same information flows utilised in the main scenario are the same ones utilised in every other scenario. All the subsystem components associated with the SCI-P are represented in Figure 26.

The system boundary to be tested by the HTB is shown by the red dashed box in the above figure. A comprehensive list of the SysML entities in this system boundary is:

- Interlocking block (S_SCI_P_SR) in Figure 26.
- Subsystem Point block (F_SCI_P_SR) in Figure 26.
- Interlocking State Machine (SCI_P STD 1) in S_SCI_P_SR.
- Subsystem Point State Machine (SCI_P STD 2) in F_SCI_P_SR.

- The 5 pairs of ports:
 - T1_Cd_Move_Point
 - DT_Move_Point_Target
 - T2_Msg_Point_Position
 - DT2_Point_Position
 - T3_Msg_Timeout

Some blocks added in Figure 26 represent input and output controls that would set data to the SCI-P in order to establish the required preconditions for test cases used in the HTB.

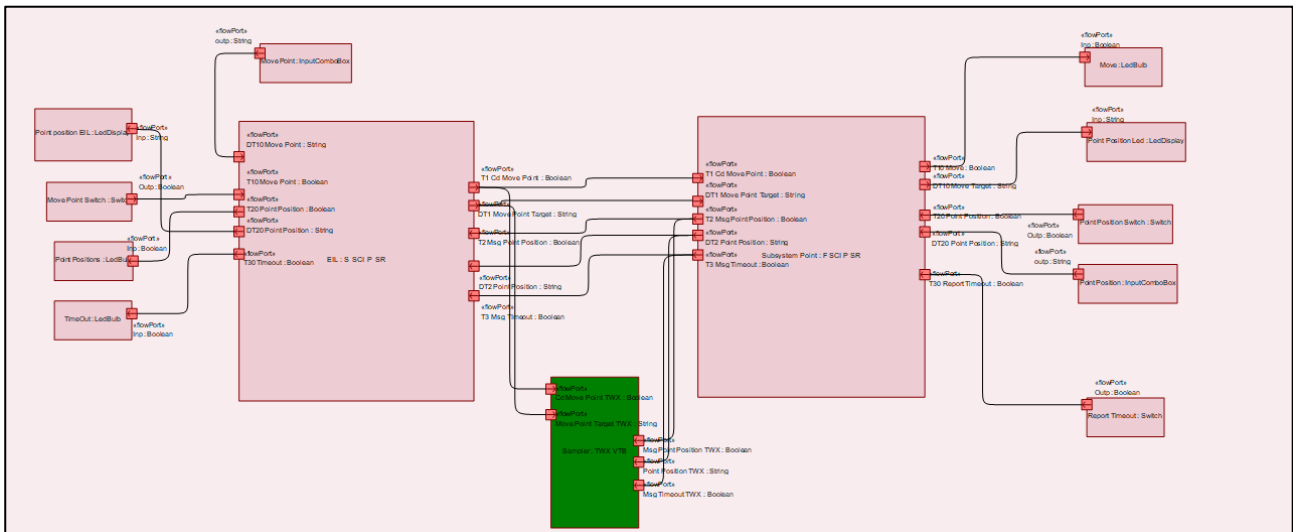


Figure 26: IBD showing relevant parts of the SCI-P and the Sampler Block

The sample block considers the principal logic needed for the Operation Mode. The interlocking and subsystem point state machines embedded in their respective are the state machines whose behaviour is executed when the test cases are executed. As a result, a system of interest was defined that retains the parts of the model that are relevant for testing the SCI-P; modifying it just enough to allow it to be manageable during validation test and retaining its likeness to a full EULYNX SCI-P model.

4.2.1 System Under Specification (Hybrid Testbed)

Before building the Virtual Test bed, it is necessary to understand the requirements and draft a map of the work that needs to be done. The specification of the HTB was carried out in part with the aid of familiar SysML diagrams.

4.2.1.1 HTB responsibilities Use case diagram.

With the help of the refined requirements, it's possible to define the behavioural responsibilities the HTB would complete in services to the stakeholders' needs. A set of use cases are defined. Each use case starts with a verb to guarantee that it covers a behavioural aspect. Furthermore, the creation of this use case diagram helps identify actors outside the system that would either invoke a use case or be acted upon by a use case. [12] In Figure 27 the primary actors that invoke use cases are on the left while the Secondary whom the HTB acts on are in the right.

The behaviour of the system captured by the use case sometimes relies on other behaviours. The significance of behaviour is captured in the use case diagram. For instance, when the use case 'Perform validation RIGHT command' executes, it is mandatory for use case 'Sample Test variables' to be executed too. This relationship is shown by assigning the arrow between them the label <include>. On the other hand, when the first use case 'Perform validation RIGHT command' executes, the use case 'Perform Validation LEFT command' can execute depending on the conditions but it is not mandatory. The label <extend> reflects that.

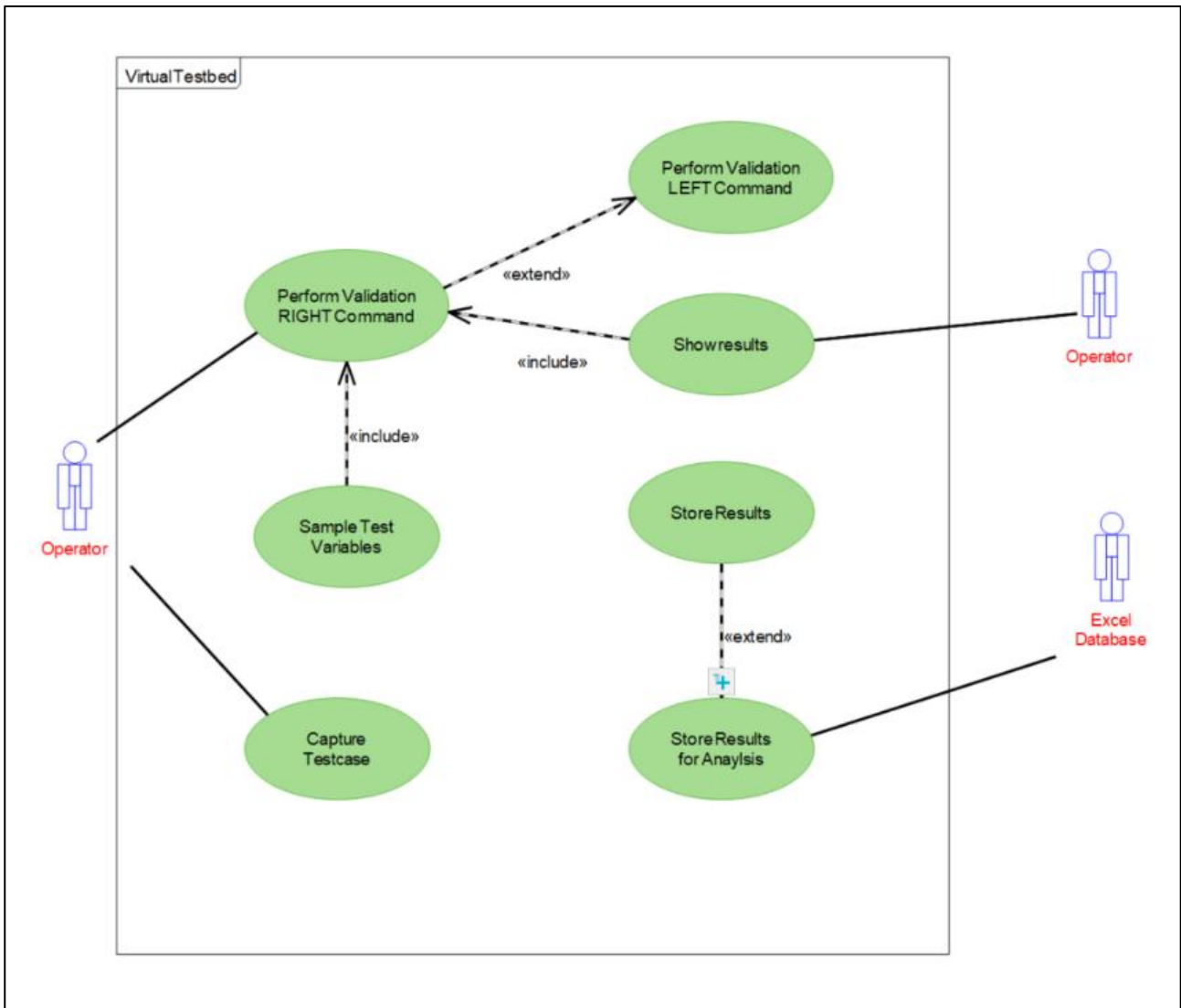


Figure 27: Use Case Diagram for HTB

4.2.1.2 Internal HTB Components communication

The next step in the specification of the HTB is to establish the internal communication of HTB components. Using the sequence diagram, the generic components that receive, process and send messages are highlighted. The descending arrangement of the arrows shown in the sequence diagram represents the chronological order of communication between HTB components. Whether the message exchange needs to be synchronised or not, is determined and represented by a type of sequence diagram arrow. In the HTB, none of the internal communication needed to be

synchronised to any other. Maintaining the chronology was sufficient for a functioning test bed. Hence all the same arrows are used in Figure 28.

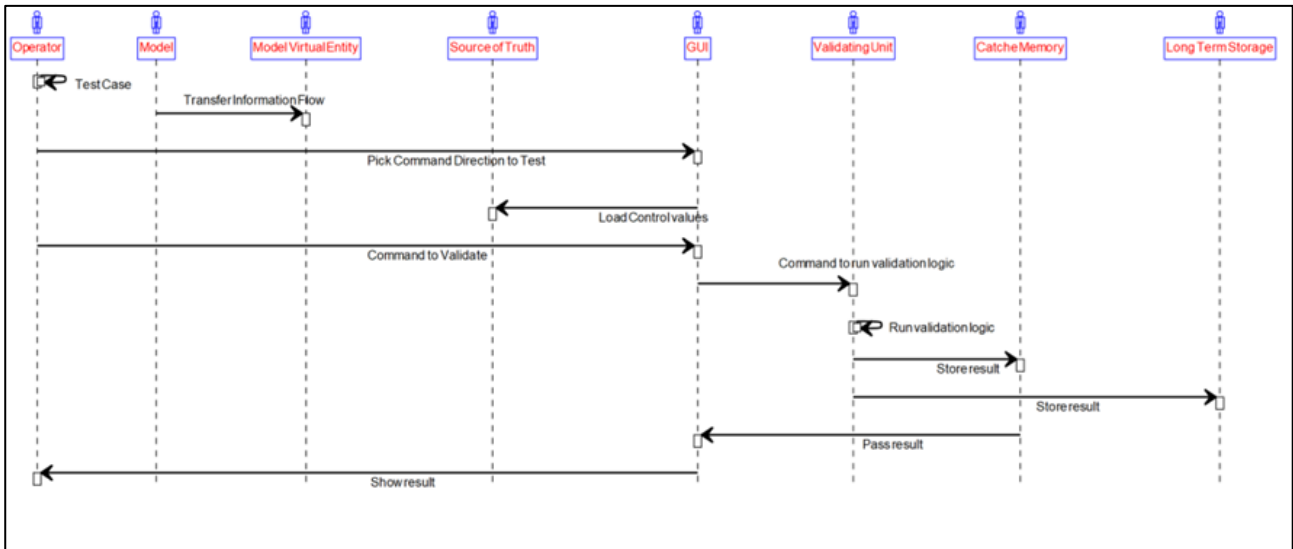


Figure 28: Sequence Diagram of HTB system components executing validation

4.2.1.3 HTB architecture

The HTB architecture comprises of essential concepts or characteristics of a system within its surroundings, manifested in its elements, connections, and design and developmental principles. It serves as the framework upon which the exterior system design is built. The HTB architecture is described by Figure 29.

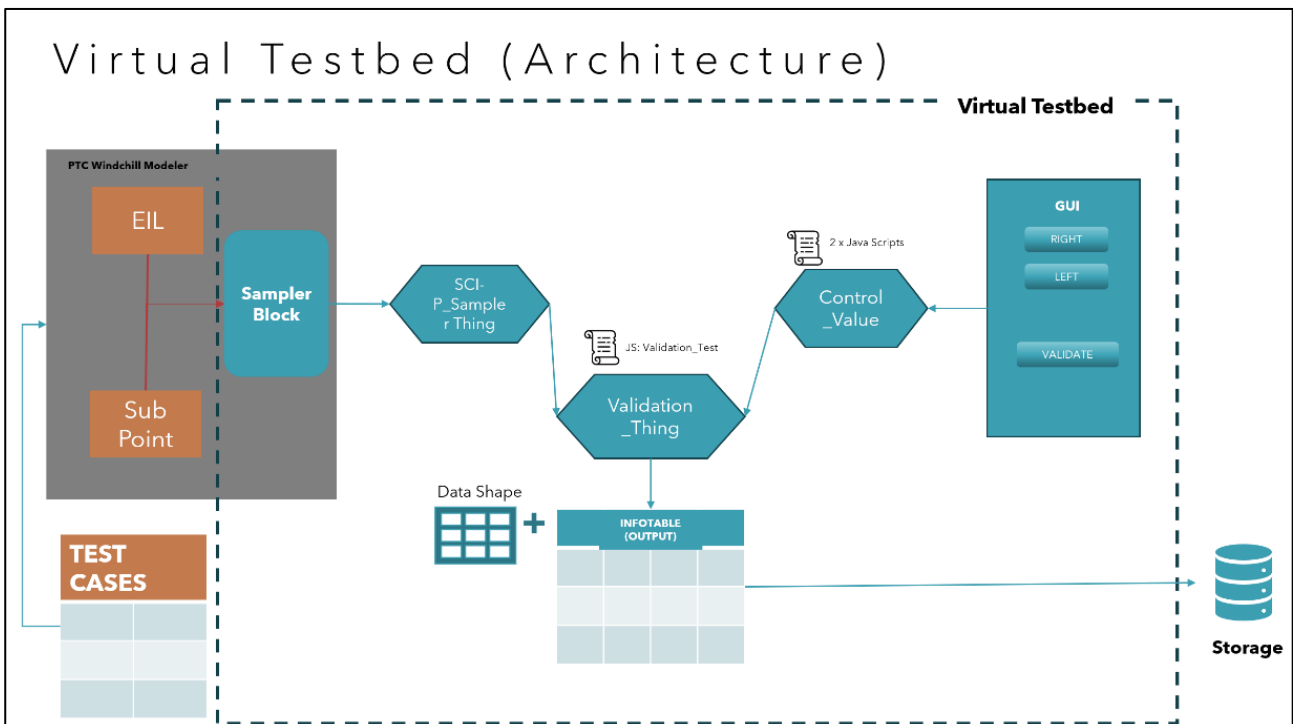


Figure 29: HTB Architecture

The architecture in Figure 29 has a sampler block to listen in on the information that is being exchanged over the SCI-P. A duplicate of just this sampler block is what is then seen in the IoT platform. The data passed to the Sampler Block's virtual duplicate is then used in the HTB to perform validation.

With the HTB architecture, it possible to perform validation with limited alteration to the SCI-P in the source environment. The architecture which uses the sampler block makes it possible to get the appropriate data without potentially introducing errors into the SCI-P.

4.2.2 Design definition process

The design characteristics of each system element introduced in the HTB architecture analysing the applicable technology and its contribution to the HTB solution are considered. The design characteristics of the architectural elements are a factor in developing the HTB.

4.2.2.1 Sampler block

The sampler block is a key element which was introduced in the SysML model where the SCI-P is. It has a total of five ports and is the green block in Figure 26. It has two ports that capture String variables and three that capture boolean variables. Figure 30 is an internal block diagram of the same sampler block.

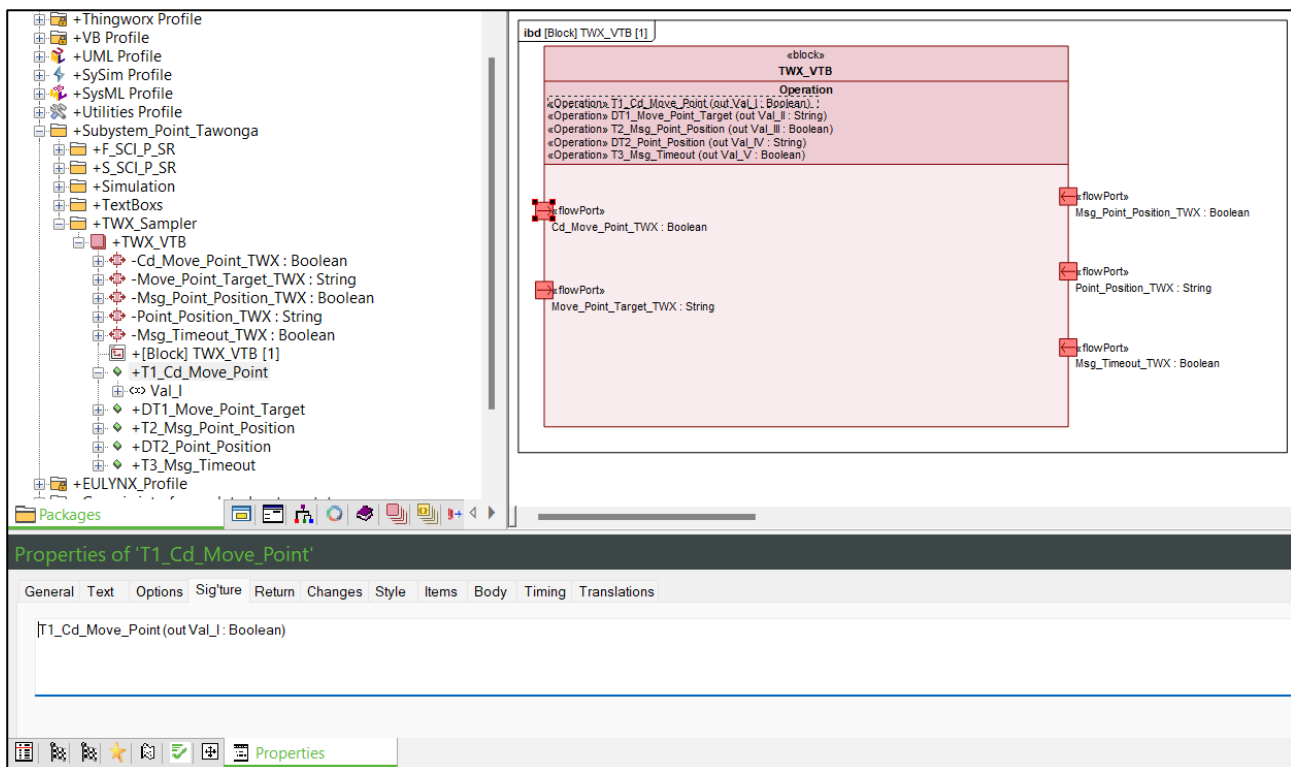


Figure 30: IBD of sampler block

When the Electronic Interlocking Subsystem commands to move the position of the Point, EIL establishes the direction by setting a String (LEFT/RIGHT) in the and follows it up by confirming the command by setting (True) through its command channel. Two of the sampler block's ports are connected to the Electronic Interlocking Subsystem and this message exchange is transferred to the IoT platform in real time.

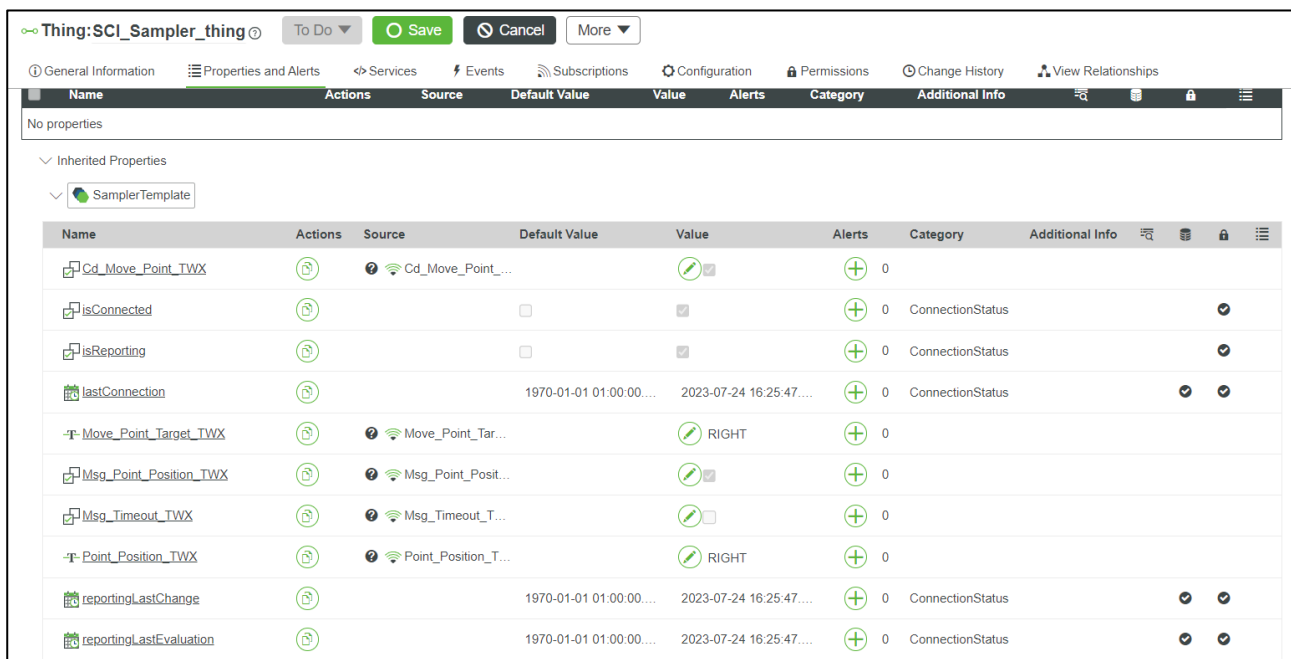
Upon receiving and executing this command, the subsystem Point performs a status update. It receives confirmation from one of its sensors about the position of the point and sends back a string variable (LEFT/RIGHT) and a confirmation Boolean (True) of the current position of the point. The fifth port, the timeout port, in the event that the Electronic Interlocking subsystem has sent a command to the subsystem Point, and a report is not sent back in a preset time, a boolean (True) will be sent to the EIL from the subsystem Point. Three of the Sampler Block's ports are connected to the Subsystem Point block and this message exchange is transferred to the IoT platform in real time.

The sampler block as seen in Figure 30 as an ibd is the gateway for the data into the HTB. Without it the operation of validation on a different platform would become very difficult. The block is also generated during run time in the IoT platform by establishing a connection between the SysML modelling tool and the IoT platform.

4.2.3 HTB components

With further reference to the architecture in Figure 29, the HTB components were set up to meet the system requirements that in turn satisfy stakeholder requirements.

The sampler block when it is generated in the HTB after runtime as a virtual entity. In this state it only has the properties of the sampler block; 2 string properties and 3 boolean properties. In order to be able to manipulate this virtual entity, the sampler block can be called in various circumstances for instance calling it using a GUI triggered event or calling it in scripting service associated with other virtual entities.



Name	Actions	Source	Default Value	Value	Alerts	Category	Additional Info
No properties							
Inherited Properties							
SamplerTemplate							
Cd_Move_Point_TWXX	[Icon]	Cd_Move_Point_...		[Icon]	[Icon]	0	
isConnected	[Icon]		☐	☒	[Icon]	0	ConnectionStatus
isReporting	[Icon]		☐	☒	[Icon]	0	ConnectionStatus
lastConnection	[Icon]		1970-01-01 01:00:00...	2023-07-24 16:25:47...	[Icon]	0	ConnectionStatus
Move_Point_Target_TWXX	[Icon]	Move_Point_Tar...		RIGHT	[Icon]	0	
Msg_Point_Position_TWXX	[Icon]	Msg_Point_Posit...		[Icon]	[Icon]	0	
Msg_Timeout_TWXX	[Icon]	Msg_Timeout_T...		[Icon]	[Icon]	0	
Point_Position_TWXX	[Icon]	Point_Position_T...		RIGHT	[Icon]	0	
reportingLastChange	[Icon]		1970-01-01 01:00:00...	2023-07-24 16:25:47...	[Icon]	0	ConnectionStatus
reportingLastEvaluation	[Icon]		1970-01-01 01:00:00...	2023-07-24 16:25:47...	[Icon]	0	ConnectionStatus

Figure 31: SCI-P_Sampler virtual entity properties

To store control variables that are used as the source of truth a virtual entity is developed called *Control_Values*. It will house predetermined sets of control variables. Due to the IoT platform, there is a possibility to connect real physical electronic interlockings and point machines to it, then use the values from the hardware as the source of truth.

The services *Right_Set_Control_Variables* and *Left_Set_Control_Variables* each have a Java Script that can be run to set the values of the properties on the virtual entity *Control_Value*. The scripts as shown in Figure 32.

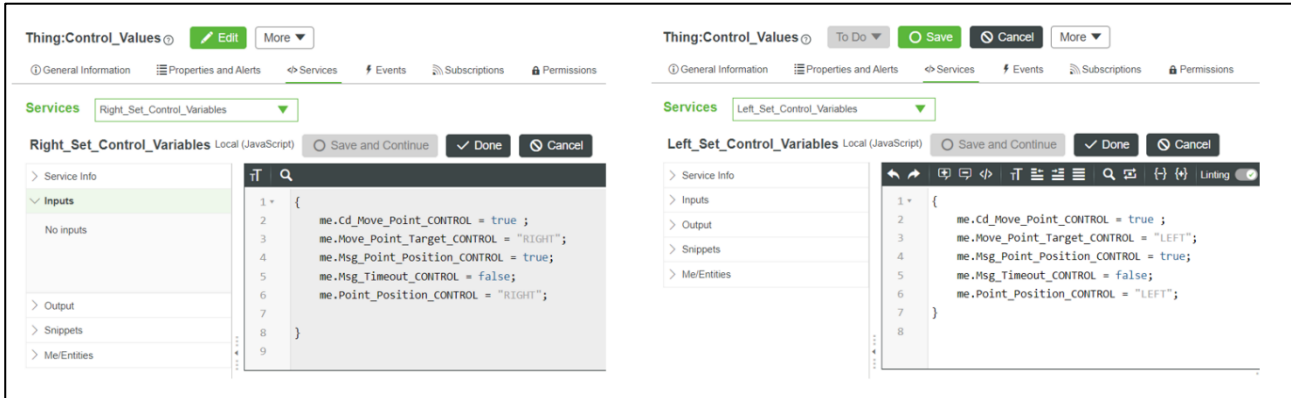


Figure 32: Services belonging to Control_Value virtual entity

For these values to be set, they need to be accessible to the tester. For that, binding is performed. Binding is where a service belonging to a virtual entity is connected to a GUI element. In the case of this HTB the buttons Left and Right are connected in a mashup environment. The click properties of the buttons are bound to the run-script property of the service. An example of the binding is shown in Figure 33.

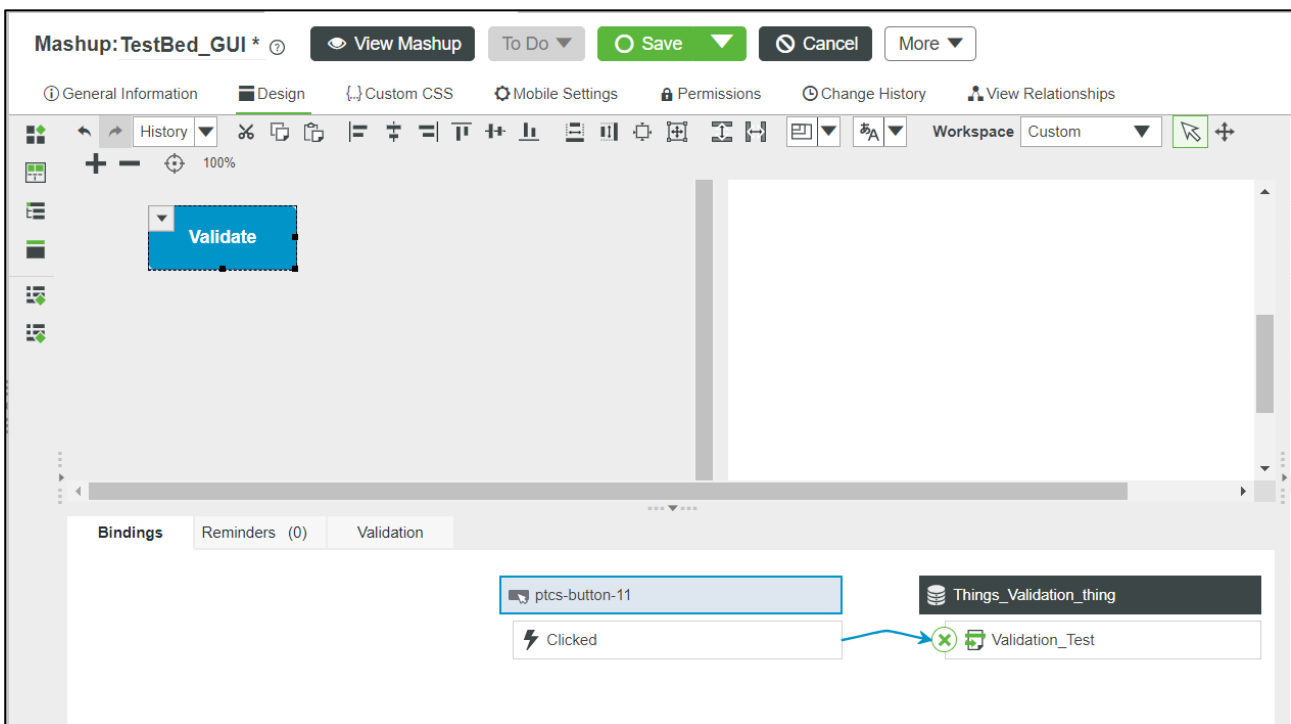


Figure 33: Binding the Validation_Test service to the Validate button

The validation entity houses the logic for performing validation of the SCI-P. Its service, *Validation_Test*, written in Java Script, pulls the values assigned to the properties of *Control_Value*. Then it pulls the values assigned to the properties of the *SCI-P_Sampler* virtual entity. The service *Validation_Test* compares a set of values from the SCI-P against the control variables. A snippet of

the code that is run during this process is seen in Figure 34. Depending on the outcome of the comparison, a validation result is returned. When a validation failure occurs, the script identifies the source of the error and outputs it for the purpose of debugging. The trigger for the running of the script is the validate button that is on the GUI and is bound in the mashup environment.

```

10 // Check each condition and collect failed ones
11 if (Things["Control_Values"].Cd_Move_Point_CONTROL !== Things["SCI_Sampler_thing"].Cd_Move_Point_TWIX) {
12     failedConditions.push("Cd_Move_Point in EIL");
13 }
14 if (Things["Control_Values"].Move_Point_Target_CONTROL !== Things["SCI_Sampler_thing"].Move_Point_Target_TWIX) {
15     failedConditions.push("Move_Point_Target in EIL");
16 }
17 if (Things["Control_Values"].Msg_Point_Position_CONTROL !== Things["SCI_Sampler_thing"].Msg_Point_Position_TWIX) {
18     failedConditions.push("Msg_Point_Position in Sub Point");
19 }
20 if (Things["Control_Values"].Msg_Timeout_CONTROL !== Things["SCI_Sampler_thing"].Msg_Timeout_TWIX) {
21     failedConditions.push("Msg_Timeout in Sub Point");
22 }
23 if (Things["Control_Values"].Point_Position_CONTROL !== Things["SCI_Sampler_thing"].Point_Position_TWIX) {
24     failedConditions.push("Point_Position in Sub Point ");
25 }

```

Figure 34: Javascript snippet from the service Validation_Test

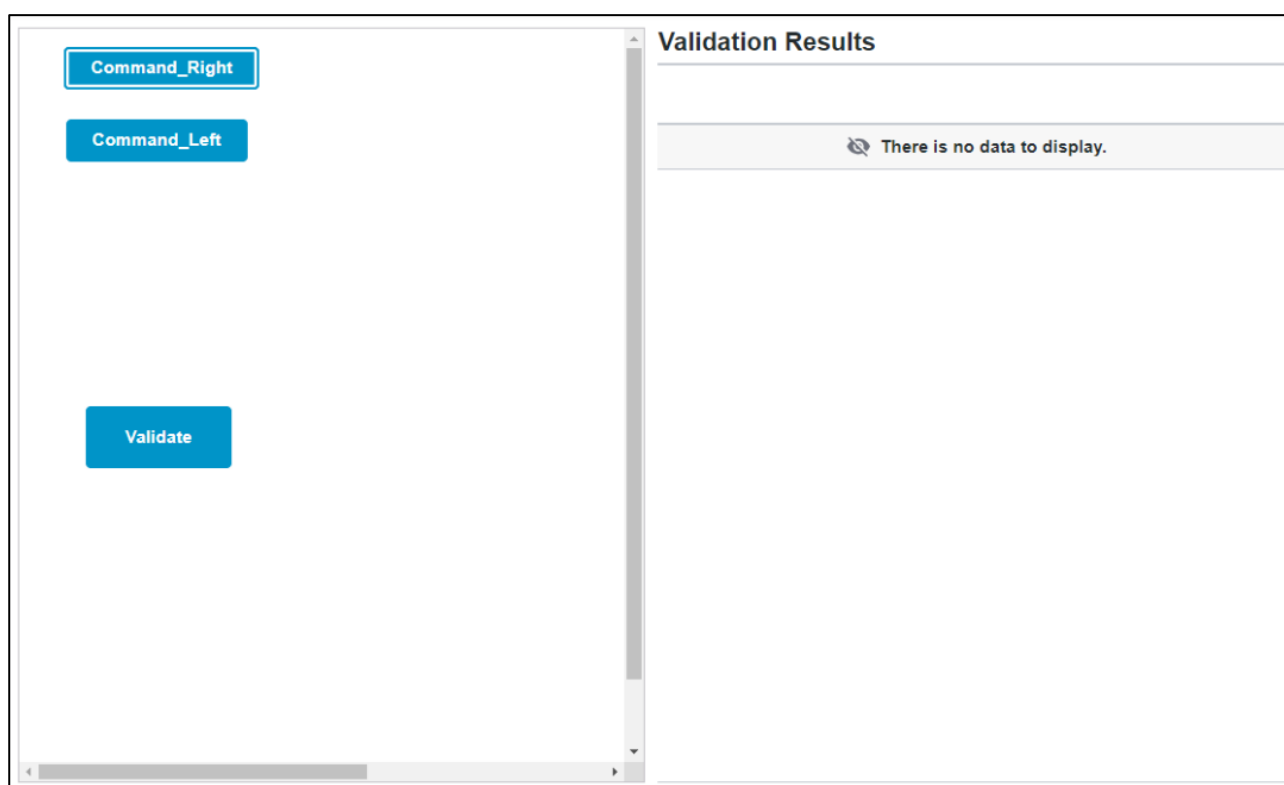


Figure 35: HTB Mashup GUI

The validation entity also has the responsibility for temporarily storing the results of the validation for the purposes of displaying them to the tester via the GUI. For this, the *validation entity* has an information table. The field definitions that are defined in the Data shape form the columns of the information table are *Command_Being_Testing*, *Error_Type* and *Validation_Status*. They are all a string data type. When the service *Validation_Test* executes, it performs validation and simultaneously generates the information table. The information is bound to a grid and is shown on

the GUI. In the end of it the tester has the ability to input data and get the output of the process through the GUI that looks like in Figure 35.

4.2.4 Validation Procedure

This section elaborates on how to perform a validation test of the SCI-P model using the HTB described in the above section. The steps and order are highlighted using the Sequence diagram that is in Figure 37.

In order to test the interfaced SCI-P there has to be information exchanged between the Electronic Interlocking Subsystem and the Subsystem Point. The state machines belonging to each of the subsystems are simulated to facilitate interaction. Simulation happens with the help of the SysML model and state machine that comes with the SysML modelling tool. When the SysML models are run, the GUI in Figure 36 is produced. The left side of the GUI represents the Subsystem EIL controls, and the right side of the GUI represents the Subsystem Point controls. From here, the tester enters one of four predefined test cases.

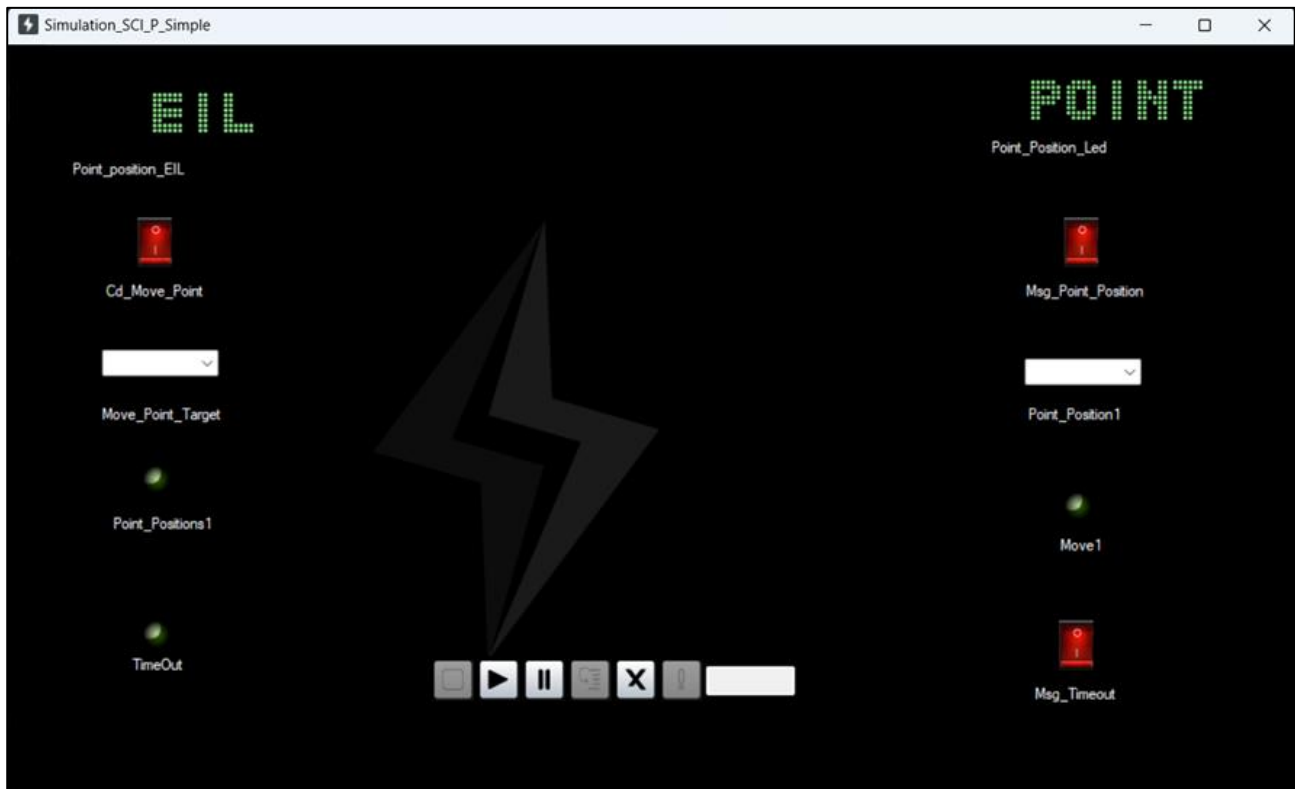


Figure 36: GUI for simulating the SCI-P

The Validation Test starts from the following initial assumptions:

- Power is supplied to both electronic interlocking subsystem and subsystem point.
- Initialisation phase has occurred, and the systems are in operation mode.
- The Point Position is in the Left Position.

The tester from here interacts with the SysML models via the GUI in Figure 36 and inputs the test cases, then interacts with the HTB via its GUI to select the test being carried out and the result is

shown to the operator as well as stored in relevant storage. The test cases and the respective testing done is described in chapter 8. The sequence diagram below shows the validation interaction.

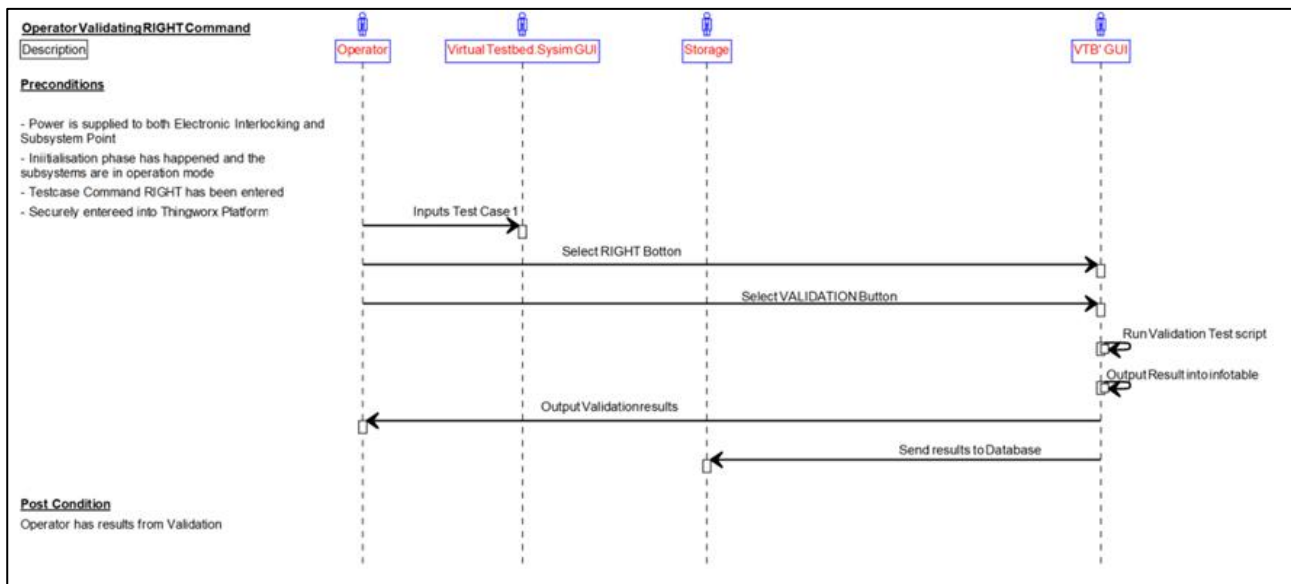


Figure 37: Sequence diagram of the validation process

4.3 LIMITATIONS

During the development the hybrid testbed using the white box approach, limitations were encountered due to various factors, this caused a pivot into developing the hybrid testbed using the black box approach as described in Chapter 5. The limitations were:

- Licensing and installation of tools and software.
- Unavailability of relevant support and documentation.
- Procuring of hardware.

5 TESTBED – BLACK BOX APPROACH

The black box approach for the hybrid testbed combines the subsystem – electronic interlocking side of the EULYNX SysML model along with the necessary telegrams to communicate with the respective EULYNX field element subsystem (EfeS).

The reference model used for developing the black box HTB is EULYNX Baseline Set 4 Release 2 (BL4R2) which is comprehensive and specifies the interfaces between the subsystems of the electronic interlocking and the field elements in detail. This reference model is also accompanied with respective test cases derived from the specifications.

The HTB described here also represents a shift towards open-source technologies due to difficulties in procuring proprietary tools, respective documentation and needing support from tool vendors. An overview of the HTB setup for the black box approach is shown in Figure 38.

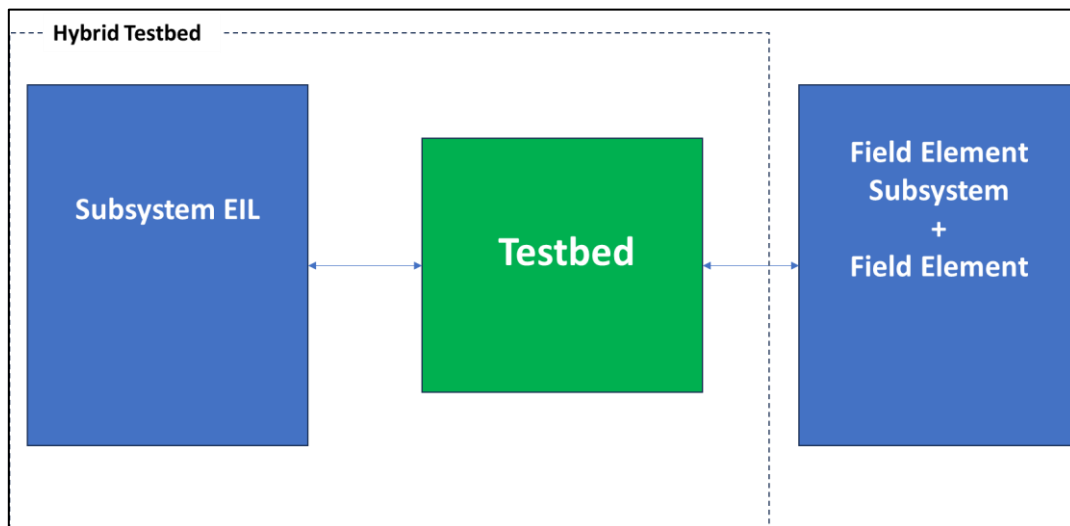


Figure 38: Hybrid Testbed – Black Box Approach Setup

5.1 HYBRID TESTBED ARCHITECTURE

The subsystem considered for the system architecture of the HTB is the Subsystem Point. The system architecture of the HTB is shown in Figure 39.

The HTB contains the EIL generic SCI and specific SCI-P functionality according to the EULYNX BLR2 model-based requirements specified based on the phase 4 of the EN50126 standard as its behaviour reference. A GUI is embedded in the HTB to enable visualisation and useful interaction by a user such as a signaller to observe the working of the field element subsystem. The HTB consists of a test suite that is capable of automatically running input test cases, model-based tests based on the state machines and model-based requirements from the EULYNX BL4R2 SysML model. The embedded test suite is also capable of running faulting injection test cases and manually running EULYNX BL4R2 test cases. The HTB also performs telegram encoding and decoding according to the SCI telegrams and ISO/IEC 8859-1:1998 [39] encoding standard specified in the EULYNX interface requirements based on the phase 5 of the EN50126 standard. The HTB containing a UDP server is able to remotely communicate with a point test component as the client. The remote connection over UDP is enabled by *Tailscale* VPN. The HTB composed of its system architecture establishes the EULYNX BL4R2 SCI-P IF over UDP between the HTB and a point test

component and enables the validation and testing of EULYNX model-based requirements and interface requirements specifications.

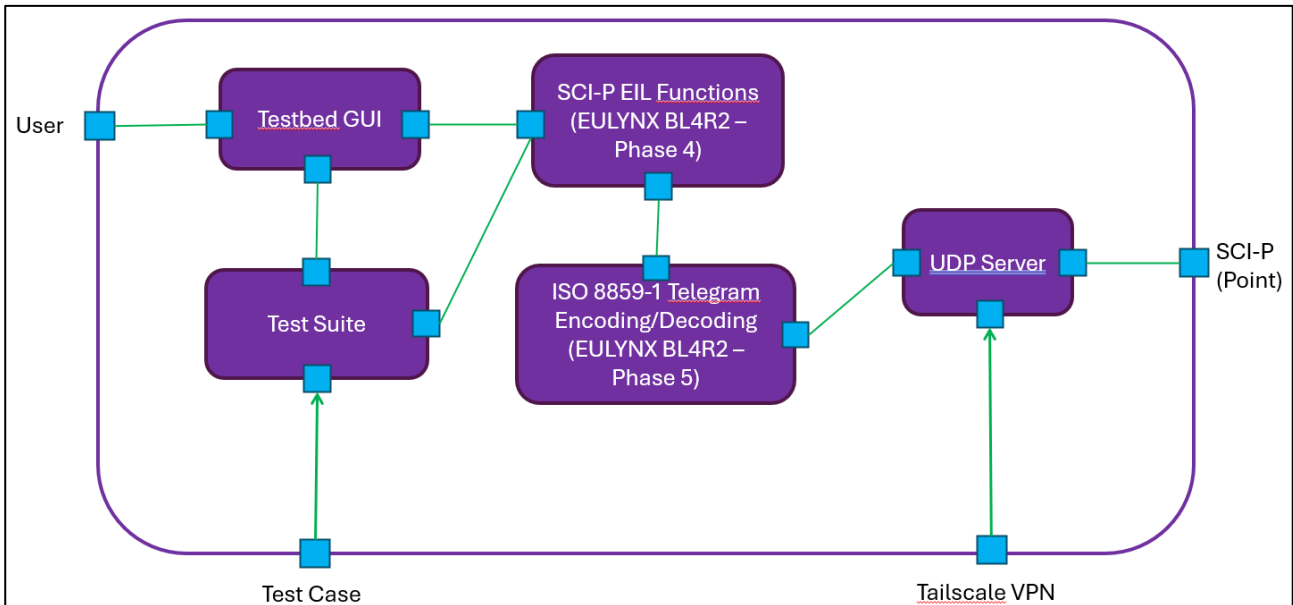


Figure 39: Hybrid Testbed System Architecture

The hybrid testbed defined for the validation approach of EULYNX components is a testbed because it is a controlled experimentation platform solutions can be integrated and test in an environment that is similar to real-world conditions [18] and it is hybrid because it is capable of integrating, interface, connecting and testing a combination of models, software and/or hardware.

The HTB through its deployment enables the control loop defined by the Traffic CS domain in the ERJU System Pillar as described in [29]. One such control loop from the System Pillar architecture is shown in Figure 40. This control loop is also in line with the control loop of EULYNX as shown in Figure 4 and explained in section 2.2 and defined by the Trackside Asset CS (TACS) domain in the System Pillar [2].

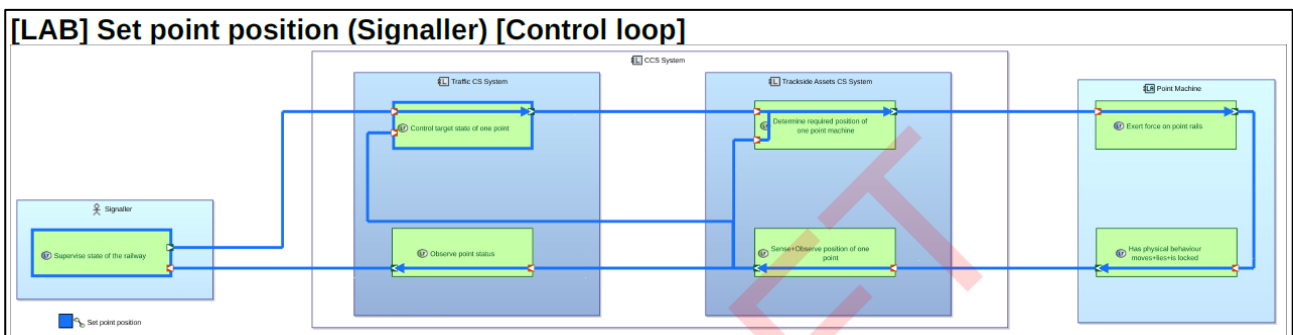


Figure 40: [LAB] Set point position (Signaller) [Control loop] [© ERJU System Pillar]

The control loop taken into consideration in Figure 40 is aligned with the EULYNX control loop in Figure 4.

This control loop is traced by the EULYNX use case P_UC2.1.1.1 Commanding and reversing as shown in the use case diagram Figure 14 and further refined by its respective use case scenario P_UC2.1.1.1.1 as shown in the sequence diagram in Figure 15. This use case scenario is used as

the basis for the EULYNX BL4R2 test case Eu.P.6714 which is a test case for the HTB and described in chapter 8.3.

From a functional view, the use case, use case scenario and test case traced from the control loop is traced to the model-based requirements specified by EULYNX as described by the internal block diagram and state machine diagram shown in Figure 44 and Figure 45 respectively.

This brings the HTB in alignment with the other tasks in Task 30.1 of WP30 of the FP2 R2DATO project.

5.2 HYBRID TESTBED IMPLEMENTATION

5.2.1 Languages and Tools

The languages that are a part of the testbed are:

- SysML v1.6
- Python 3

The tools that form the toolchain used for the testbed are:

- PyQt6 [34]
- Visual Studio Code
- Tailscale VPN [35].

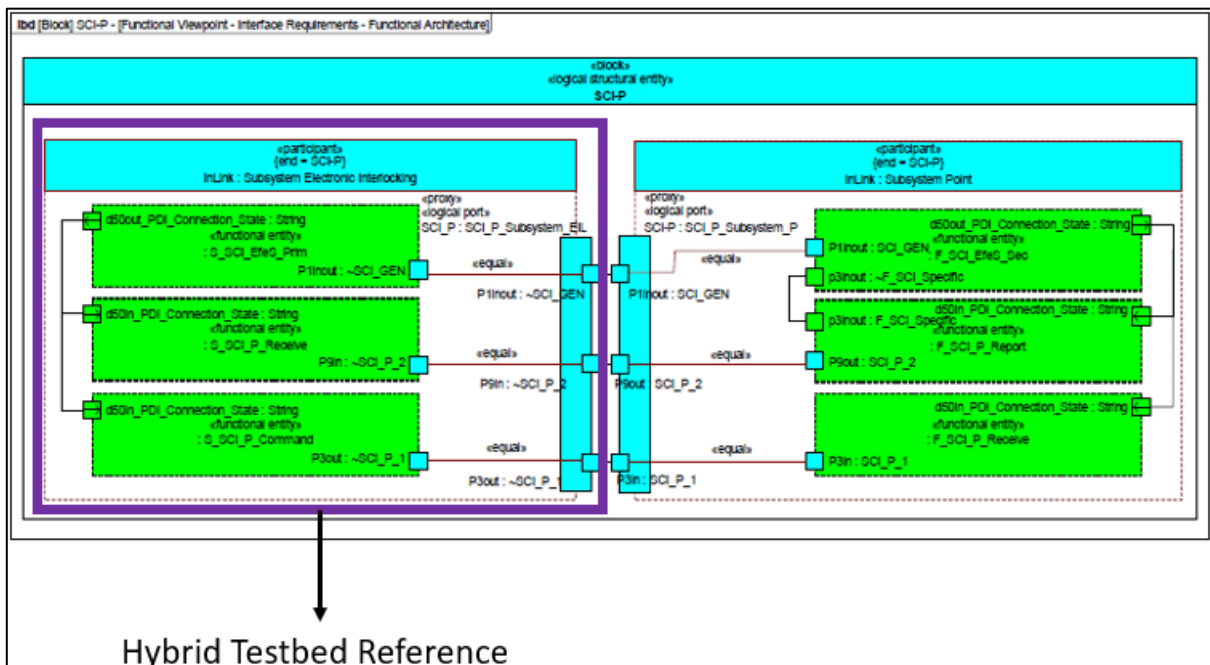


Figure 41: EULYNX Functional Entities considered for HTB implementation

5.2.2 Testbed Behaviour Reference

The hybrid testbed contains the functional entities of the Subsystem – Electronic Interlocking for the SCI-P interface as the case study for this deliverable is the EULYNX Subsystem Point. The behaviour reference from the SCI-P model also enables the integration of the HTB with the TC as

described in chapter 7. The respective functional entities (Figure 42, Figure 43, Figure 44, Figure 45, Figure 46 and Figure 47) from the EULYNX SCI-P functional architecture (see Figure 41) are derived from the reference model EULYNX BL4R2 developed in Python to utilise the requirements to be tested against a test component and to establish communication over the SCI interface.

5.2.2.1 SysML model extract

Three blocks and their respective state machines shown below were utilised for developing the behaviour program.

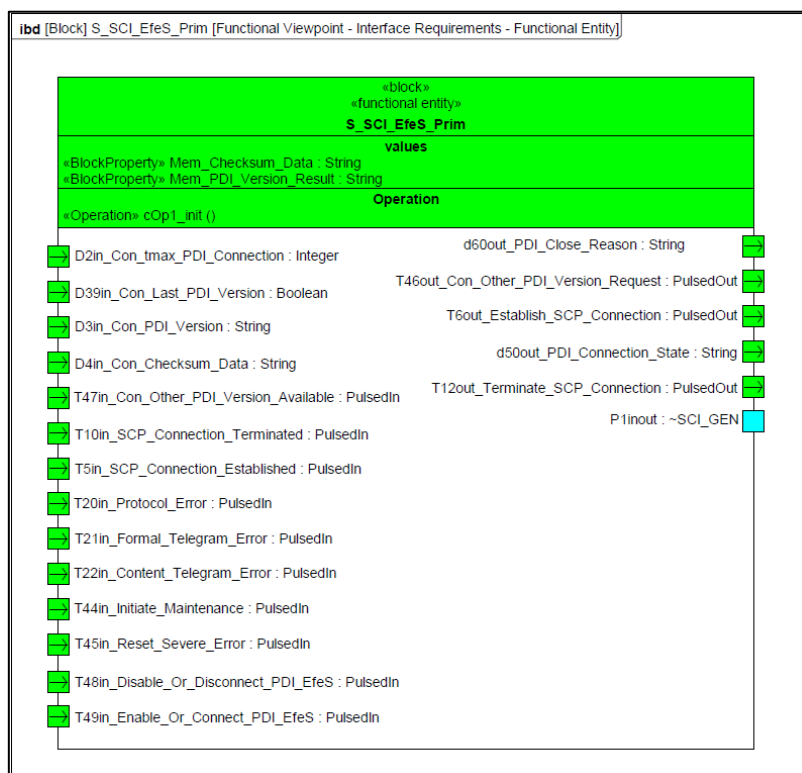


Figure 42: IBD of S_SCI_EfeS_Prim functional entity

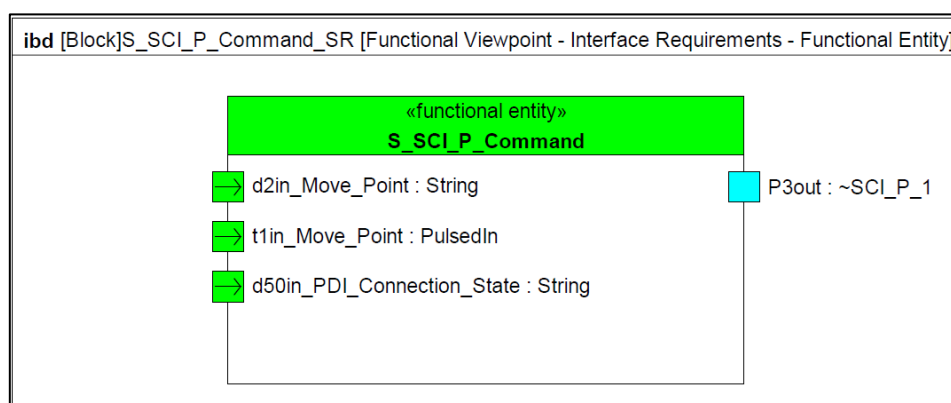


Figure 43: IBD of EULYNX Point S_SCI_P_Command FE

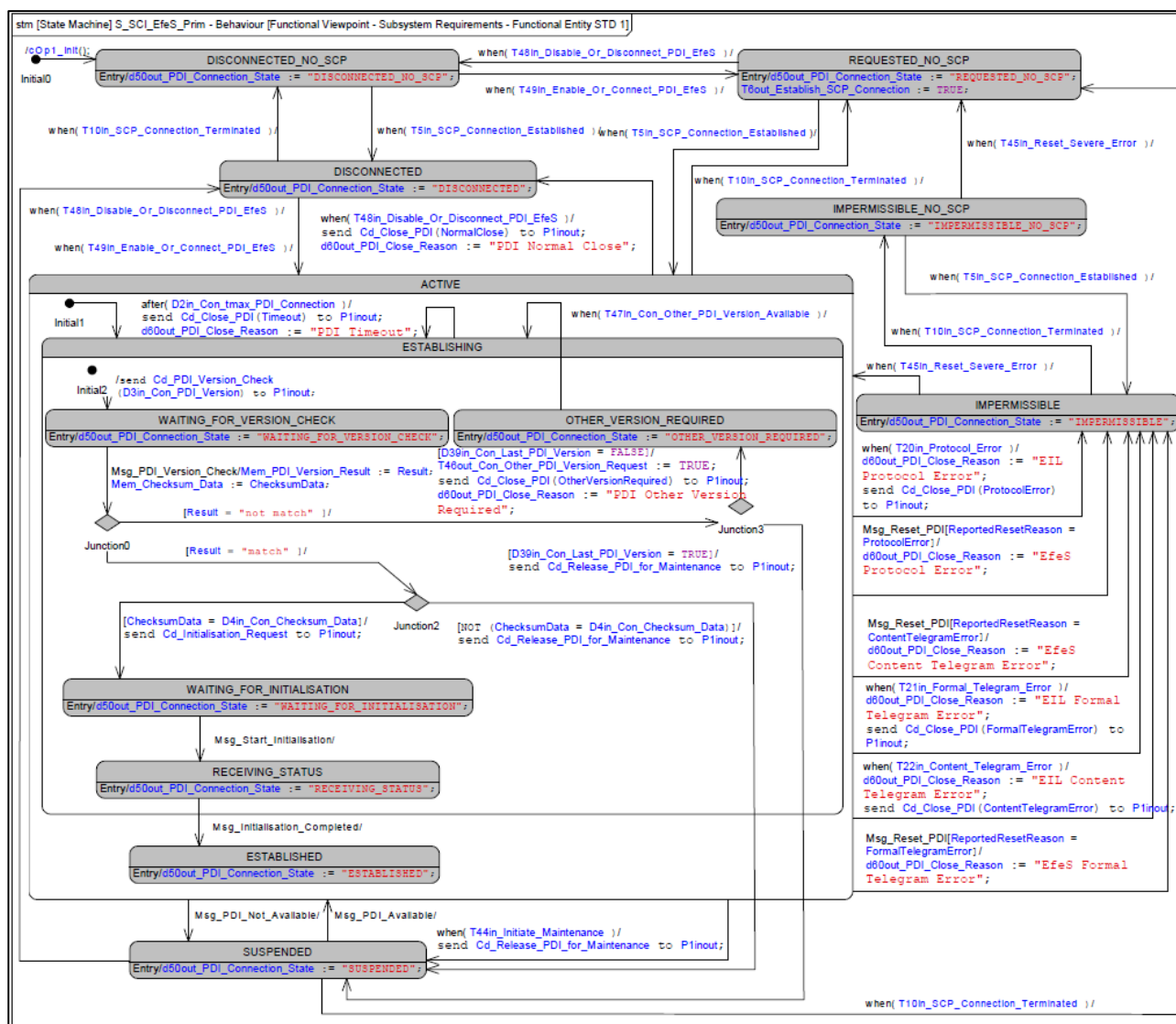


Figure 44: stm of EULYNX Generic EIL S_SCI_EfeS_Prim functional entity

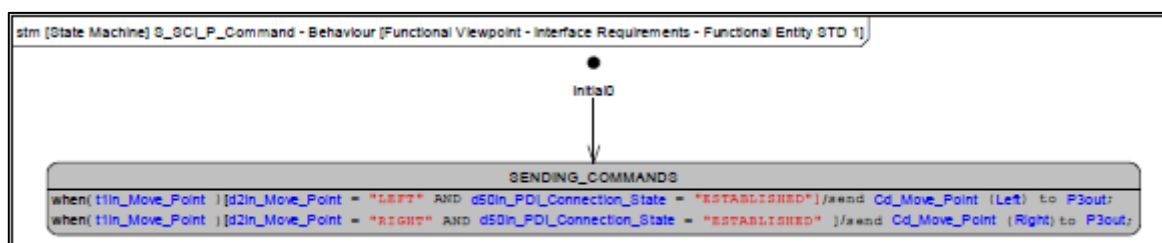


Figure 45: stm of EULYNX Point S_SCI_P_Command FE

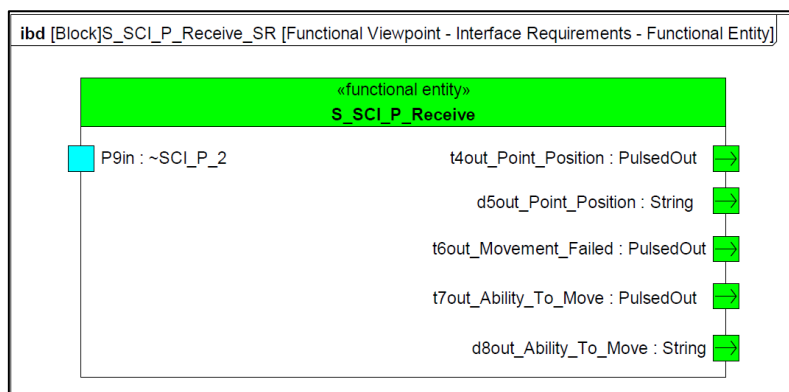


Figure 46: IBD of EULYNX Point S_SCI_P_Receive FE



Figure 47: stm of EULYNX Point S_SCI_P_Receive FE

5.2.2.2 Python code implementation excerpt

Excerpts of the Python code implementation of the three functional entities within the hybrid testbed is shown below:

```
def DISCONNECTED(self):
    self.state = "DISCONNECTED"
    self.d50out_PDI_Connection_State = "DISCONNECTED"
    self.setPDI()
    print("state: DISCONNECTED")
    while self.state == "DISCONNECTED":
        self.collectInputs()
        if self.T49in_Enable_Or_Connect_PDI_EfeS == True: #Transition (EULYNX
Requirement Eu.Gen-SCI.331: when(T49in_Enable_Or_Connect_PDI_EfeS)/{DISCONNECTED -
ACTIVE})
            self.T49in_Enable_Or_Connect_PDI_EfeS = False
            self.ACTIVE() #Moving to ACTIVE state
        if self.T10in_SCP_Connection_Terminated == True: #Transition (EULYNX
Requirement Eu.Gen-SCI.330: when(T10in_SCP_Connection_Terminated)/{DISCONNECTED -
DISCONNECTED_NO_SCP})
            self.T10in_SCP_Connection_Terminated = False
            self.DISCONNECTED_NO_SCP() #Moving to DISCONNECTED_NO_SCP state
```

Program Code 1: Python code of DISCONNECTED state and transitions to ACTIVE and DISCONNECTED_NO_SCP states in S_SCI_EfeS stm

```
self.state = "SENDING_COMMANDS" #Current state of S_SCI_P_Command stm
while self.state == "SENDING_COMMANDS":
    self.d50in_PDI_Connection_State = self.readPDI()
    #Internal Transition (EULYNX Requirement Eu.P.6133:
when(tlin_Move_Point)[d2in_Move_Point = "LEFT" AND d50in_PDI_Connection_State =
"ESTABLISHED"]/send Cd_Move_Point (Left) to P3out;{State-internal in SENDING_COMMANDS})
    if self.tlin_Move_Point == True:
        if self.d2in_Move_Point == "LEFT" and self.d50in_PDI_Connection_State ==
"ESTABLISHED":
            self.P3out = "Left"
```

Program Code 2: Python code of SENDING_COMMANDS state, internal transition in SENDING_COMMANDS and sending command to Subsystem Point to move the Point to LEFT position in S_SCI_P_Command stm

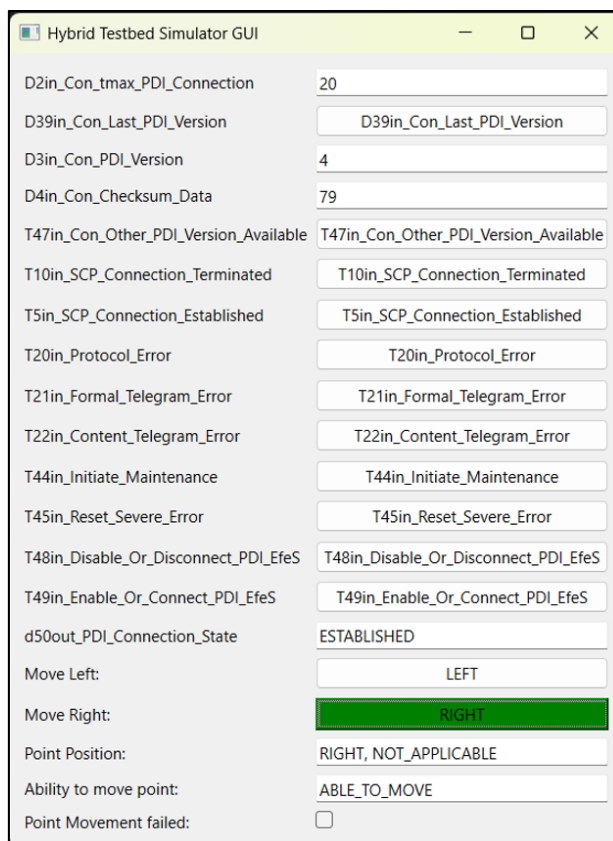
```
if self.state == "RECEIVE_OVERALL_POINT_POSITION_AND_DEGRADED_POINT_POSITION_REPORT":
    #Current state of S_SCI_P_Receive stm
    if self.P9in.split(sep = ",", ") != [] and len(self.P9in.split(sep = ",", ") ==
2:
        #Internal Transition (EULYNX Requirement Eu.P.7060:
Msg_Point_Position[ReportedPointPositionState = Left AND ReportedDegradedPointPosition =
NotApplicable]/d5out_Point_Position := "LEFT, NOT_APPLICABLE";
                                                                    #t4out_Point_Position
:= TRUE;{State-internal in
RECEIVE_OVERALL_POINT_POSITION_AND_DEGRADED_POINT_POSITION_REPORT})
        if self.P9in.split(sep = ",", ") [0] == "Left" and self.P9in.split(sep = ",",
") [1] == "NotApplicable":
            self.d5out_Point_Position = "LEFT, NOT_APPLICABLE"
```

Program Code 3: Python code of RECEIVE_OVERALL_POINT_POSITION_AND_DEGRADED_POINT_POSITION_REPORT state, internal transition in the state and receiving message from Subsystem Point to that

the Point is in LEFT position with no applicable degraded point position in S_SCI_P_Receive stm

5.2.3 Testbed GUI

The testbed provides a graphical user interface (shown in Figure 48) for a user such as a signaller to command and control the field element subsystem or a tester to run test cases and test the working of the SCI interface and the external behaviour of the field element subsystem. The Python library PyQt6 for used to develop the GUI of the hybrid testbed.



Hybrid Testbed Simulator GUI	
D2in_Con_tmax_PDI_Connection	20
D39in_Con_Last_PDI_Version	D39in_Con_Last_PDI_Version
D3in_Con_PDI_Version	4
D4in_Con_Checksum_Data	79
T47in_Con_Other_PDI_Version_Available	T47in_Con_Other_PDI_Version_Available
T10in_SCP_Connection_Terminated	T10in_SCP_Connection_Terminated
T5in_SCP_Connection_Established	T5in_SCP_Connection_Established
T20in_Protocol_Error	T20in_Protocol_Error
T21in_Formal_Telegram_Error	T21in_Formal_Telegram_Error
T22in_Content_Telegram_Error	T22in_Content_Telegram_Error
T44in_Initiate_Maintenance	T44in_Initiate_Maintenance
T45in_Reset_Severe_Error	T45in_Reset_Severe_Error
T48in_Disable_Or_Disconnect_PDI_EfeS	T48in_Disable_Or_Disconnect_PDI_EfeS
T49in_Enable_Or_Connect_PDI_EfeS	T49in_Enable_Or_Connect_PDI_EfeS
d50out_PDI_Connection_State	ESTABLISHED
Move Left:	LEFT
Move Right:	RIGHT
Point Position:	RIGHT, NOT_APPLICABLE
Ability to move point:	ABLE_TO_MOVE
Point Movement failed:	☐

Figure 48: HTB User GUI

5.2.4 Testbed Remote Connection

The HTB has the capability of remotely connecting with UDP to a test component using Tailscale VPN. The hybrid testbed and test component are configured on a network to enable the remote connection (see Figure 49). The relevant IP addresses are included in the server of the testbed and client of the test component which then connects the test component to the hybrid testbed.

5.2.5 Testbed Telegram Communication

The HTB encodes and decodes the command and message information flows exchanged as telegrams for the communication between the interlocking and the subsystem point over the SCI-P interface according to the ISO:8861-1:1998 standard specified by EULYNX in their phase 5 interface specifications.

htb302.github

Machines Apps Services Users Access controls Logs DNS Settings Resource hub

Machines

Manage the devices connected to your tailnet. [Learn more](#) Add device

Search by name, owner, tag, version... Filters

3 machines

MACHINE	ADDRESSES	VERSION	LAST SEEN
hybridtestbed HTB302@github	100....	1.90.6 Windows 11 24H2	Connected
hybridtestbedcomponent HTB302@github	100....	1.86.2 Linux 6.12.25+rpt-rpi-v8	Oct 28
pointtestcomponent HTB302@github	100....	1.88.3 Linux 6.1.21-v8+	Nov 5, 10:09 AM GMT+1

Figure 49: HTB and components in Tailscale network for remote connection and validation

The information flows from the EULYNX model-based requirements in their phase 4 requirements specifications shown in the Figure 50 and Figure 51 below were considered by the HTB for the SCI-P interface.

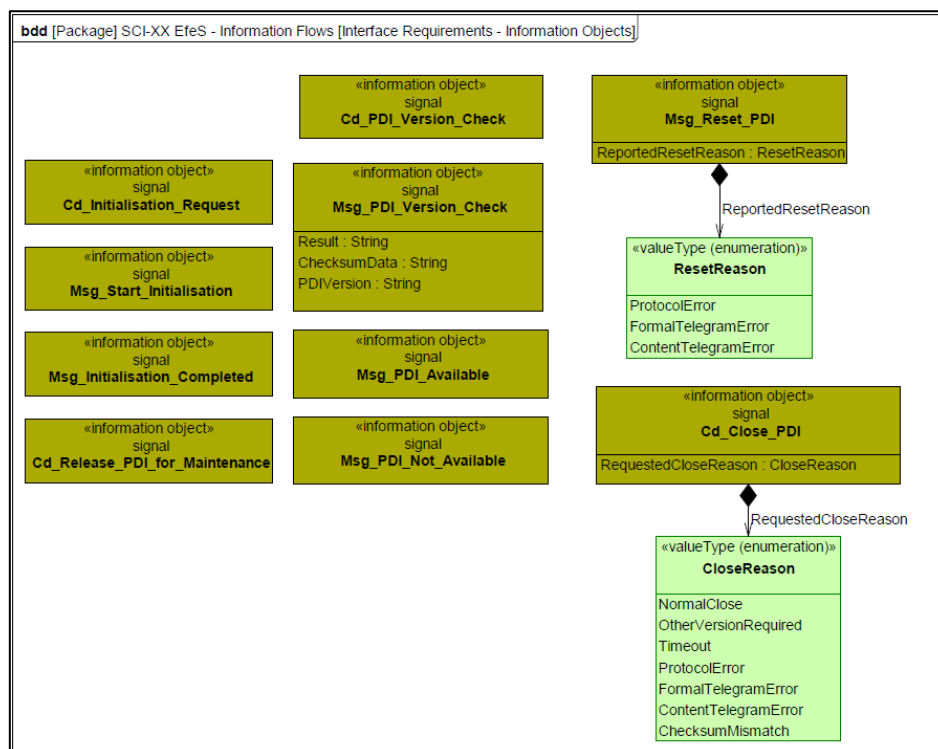


Figure 50: Generic SCI information flows communicated as telegram with ISO/IEC 8859-1:1998

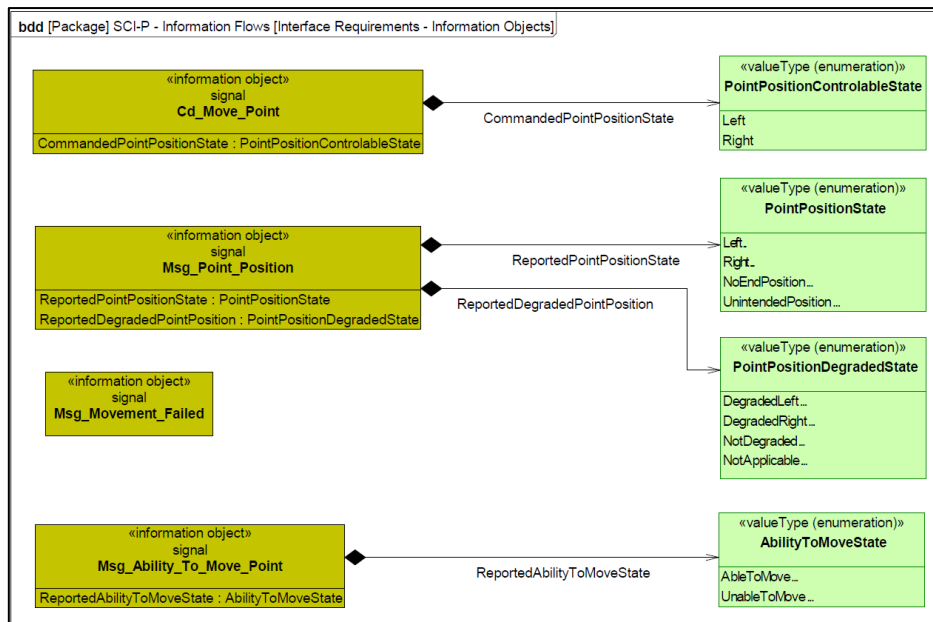


Figure 51: SCI-P information flows communicated as telegram with ISO/IEC 8859-1:1998

		Subsystem point messages		Protocol		Message Type		Sender Identification														Receiver Identification															
								Country Code		Area Designation		System type		Code		Tag		Sequence no		Country Code		Area Designation		System type		Code		Tag		Sequence no							
Point Specific	cd	Move Point	(Right)	040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F					
			(Left hand position)	040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F				
				defined end position																																	
				degraded																																	
				(Left hand position)	040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F			
				defined end position																																	
				degraded																																	
				point position is not applicable																																	
				(No end position)	040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F			
				defined																																	
			(Unintended position)	040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F				
			point position is not applicable																																		
msg	Point Position			040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F				
msg	Movement Failed			040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F				
msg	Ability To Move Point			040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F				
				able to move																																	
				unable to move																																	
Generic	cd	PDI Version check		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F				
			do not match		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F			
	msg	PDI Version check		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F			
			do match		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F		
	cd	Initialization Request		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F			
	msg	Start Initialization		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F		
	msg	Status Report Completed		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F		
	msg	Initialization Completed		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F		
	cd	Close PDI		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F		
			protocol error		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F		
			Format telegram error		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F		
			Content telegram error		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F		
	cd	Release PDI for Maintenance		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F		
			Timeout		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F		
		msg	PDI Available		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F		
		msg	PDI Not Available		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F		
msg	Reset PDI		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F			
		protocol error		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F			
		Format telegram error		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F			
		Content telegram error		040	040	040	044	045	048	042	045	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F	04F			

Figure 52: Overview of EULYNX Telegram Encoding/Decoding according to ISO:8859-1:1998

```

elif self.btgm == bytes([0x40, 0x00, 0x23,
                        0x44, 0x45, 0x48, 0x42, 0x53, 0x5F, 0x50, 0x5F,
                        0x5F, 0x5F, 0x5F, 0x39, 0x39, 0x23, 0x23, 0x39, 0x39, 0x39, 0x39,
                        0x44, 0x45, 0x48, 0x42, 0x53, 0x5F, 0x5F, 0x45, 0x49,
                        0x4C, 0x5F, 0x5F, 0x31, 0x31, 0x23, 0x23, 0x31, 0x31, 0x31, 0x31, 0x31, 0x31,
                        ]): # EULYNX Telegram covering EULYNX SCI interface
requirements: Eu.SCI-XX.PDI.112, Eu.SCI-XX.PDI.113, Eu.SCI-XX.PDI.128, Eu.SCI-
XX.PDI.114, Eu.SCI-XX.PDI.115
self.Plinout = "Initialisation Completed"

```

Program Code 4: Python code excerpt of Msg_Initialisation_Completed Telegram decoded into its respective information flow at the P1inout proxy port that transitions the PDI connection into the ESTABLISHED state

5.2.6 Test Suite

The HTB contains a test suite that supports automatic and manual testing. Automatic testing by executing the state machine and its respective model-based requirements in combination with the interface requirements is possible. Automatically execution of developed test cases is supported by the embedded test runner. The test suite also supports EULYNX BL4R2 test cases for testing. Fault-injection testing is enabled by the test suite. The various forms of testing described by the test suite of the hybrid testbed is demonstrated in chapter 8.

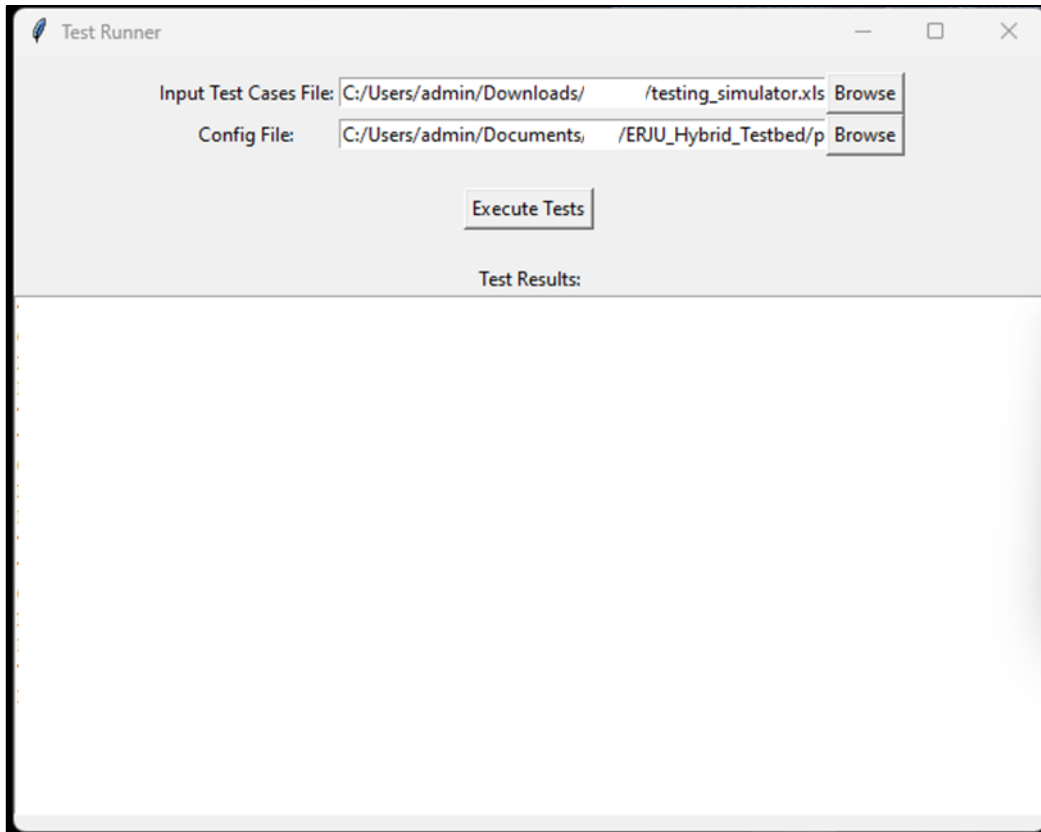


Figure 53: HTB test suite with test driver integrated as test runner

5.3 HYBRID TESTBED VALIDATION

Validation performed by the hybrid testbed is done by the correct fulfilment of the intended functions, performance and behavioural expectations of the EULYNX Subsystem Point derived from its respective model-based and interface requirements as defined in the EULYNX specifications.

Validation is conducted by integrating the SysML-based EULYNX model, its respective behaviour code implementation, the remote connection infrastructure and a connected test component into a combined testing environment. The validation procedure assesses whether the HTB behaves in accordance with stakeholder expectations. Some of the collected stakeholder expectations are:

- Transmission of information flows between the interlocking and the subsystem point.
- Accurate execution of interlocking commands to the subsystem point.
- Correct response of the subsystem to interlocking messages.
- Reliable data and telegram exchange over the SCI-P via UDP interface communication.

The validation approach ensures that the hybrid testbed not only implements the EULYNX SysML model but also reproduces realistic operational scenarios in context of the railway signalling subsystem. By running automatic, manual and fault-injection tests, the validation confirms that the testbed can effectively replicate field-level interactions, visualise subsystem behaviour through the testbed GUI and behavioural SysML state machines and maintain correct and consistent communication according to the *ISO/IEC 8859-1:1998* telegram encoding and EULYNX model-based and interface requirements. The HTB validation approach proves that the integrated combination of model, software and hardware meets the intended use cases and scenarios of EULYNX and its respective test case for components. The HTB provides a reliable platform for model-based system validation in near real-world-conditions. The validation performed by the HTB is further described in chapters 7 and 8.

6 A VERSATILE POINT TEST COMPONENT

The content of this chapter has also been published in [19], with slight differences in choice and arrangement of the texts.

6.1 BASIC IDEA: USE CASES AND USER REQUIREMENTS

The most basic requirement for the TC is to (ideally completely) implement the nominal behaviour of the subsystem(s). For standardised behaviour this means “as specified”; for non-standardised behaviour this means “adding a simulation of realistic input/output (I/O) behaviour”. It may include behaviour in degraded situations; and it includes to make any external and possibly open internal IF, as well as important status information, accessible from the outside. This already allows to realize a number of use cases for the TC:

- Use as SUT, e.g. during building up a test lab, or as part of the SUT, e.g. for early integration testing. More generally, this use case applies when no subsystem implementation is available yet or the available ones do not provide sufficient information/insight for debugging purposes.
- Use as reference. This means that SUT and reference both have the same scope and receive the same inputs, and SUT outputs are compared to reference outputs.
- Use – possibly multiple instances – as replacement (“dummy”) in larger test set-ups, e.g. during (software) system testing. This use case applies when real products are not yet ready or considered too expensive / too laborious to deploy for testing.

As the identity and functionality of signalling systems is usually configurable, the TC is required to be fully configurable (according to its specification or to the simulation options available). This ensures that all configurable functions can actually be tested or are available as part of the dummy, and that the TC can be adapted for use in given environments.

In case more than one subsystem is covered by the TC, an option to switch off subsystems – making internal IF of them external – should be considered. This allows to alternately or gradually replace subsystems of the TC by real subsystems, supporting testing of all kinds of integration stages.

Furthermore, impossible behaviour of the subsystems should be possible to provoke by the tester, going beyond nominal I/O behaviour. This may include manipulation of functions (failure injection) or of non-functional properties (e.g. insertion of delays). This enables additional use cases like testing the SUT reaction to a faulty environment, or to check that test cases actually fail in case of a faulty SUT.

6.2 CONCEPT: REALISATION OF THE EULYNX SUBSYSTEM POINT

6.2.1 Scope and Architectural Overview

The scope of the TC has been chosen to comprise two subsystems (see darker blue boxes in Figure 54):

1. A point controller, with IF and functionality as specified by EULYNX (see Section 6.2.2 below), except that the special “4-wire” variant of the P3 IF, the option to use an IF to the EULYNX “Maintenance and Data Management” subsystem and the “Standard Diagnostic IF” (not affecting the controller functions) have not been realised. The main controller functions are to relay a point position commanded by the interlocking to the point machine (PM) and to

report an aggregated point position status and failed point movements back to the interlocking.

2. A PM simulation, with the IF to the point controller inherited from the EULYNX specification, but otherwise with self-specified functionality. The main PM simulation functions are to simulate the movement of the PM and to report its status – position and ability to move – to the point controller.

This scope has the advantage of being self-contained in the sense that there is only one mandatory external IF (the standard communication interface, SCI, of the point controller) to another subsystem (interlocking) that is important for functional testing, which moreover is completely standardised. Note that to achieve the latter, a (pre-existing) DLR implementation of the necessary RaSTA (Rail Safe Transport Application) protocol [20] has been integrated into the TC software (see leftmost lighter blue box in Figure 54). A general advantage of choosing a EULYNX subsystem is the existing detailed specification not only of IF, but also of the subsystem behaviour, and the existing (first version of) certification test cases.

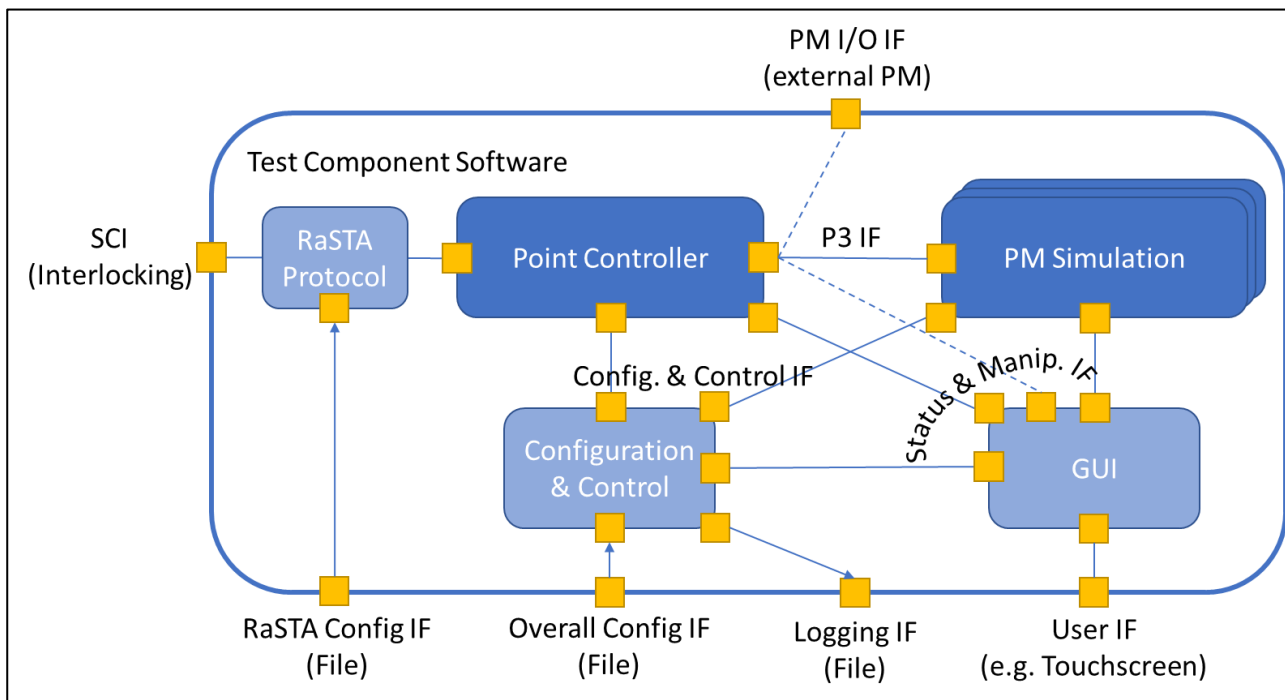


Figure 54: Overview of the TC's structure and IF

For setting up and running the TC, configuration and control functionality has been added, as well as a graphical user IF (GUI) for interaction with a tester (see lower lighter blue boxes). The former e.g. cares for a fair processing of the point controller, all PM simulations and the in- and outputs of the SCI and PM I/O IF and stores all log entries to a file upon finishing a TC run. The latter has four tabs corresponding to the four internal status and manipulation IF shown in Figure 54. (for a picture of the implemented GUI see Figure 57 below).

Configurability of the TC already starts at the overall TC level: induced by a configuration parameter of the EULYNX point controller that allows an arbitrary¹ number (≥ 1) of controlled PM, it is possible

¹ [17] mentions a limit of 52 PM, but without further justification or reference.

to use several instances of the PM simulation (indicated by multiple boxes for PM Simulation in Figure 54). Also, for each PM it is possible to switch off the built-in simulation, leaving the PM behaviour to an external actor (e.g. a connected physical PM, or the user of the TC interacting with its GUI – see the dashed lines in Figure 54).

6.2.2 Existing and Made-Up Specifications

The point controller subsystem is implemented according to the latest EULYNX specifications from Baseline 4 Release 2; this includes in particular the requirement specifications [21][22][23], the IF specifications [24][25] of the SCI, and the architecture specification [26]. Generally, the tabular (SCI telegrams) and model-based (subsystem logical structure and behaviour) specifications define the subsystem very precisely and consistently; also, a list of configuration options is already included. Overall, the specifications comprise 14 telegrams, 10 interconnected SysML (Systems Modelling Language [10]) blocks containing one statechart each, and 15 configuration options relevant for the point controller. However, there remain specification gaps, in particular since SysML (version 1.6) leaves the execution order of events in statecharts, as well as few other details, up to implementation, and EULYNX seems not to close these gaps either. In addition, during analysis of the specifications (by review), besides some formal issues, behavioural ambiguities in the concrete model were detected, reported to and discussed with the EULYNX cluster in charge. To proceed, some working assumptions were made to resolve these issues.

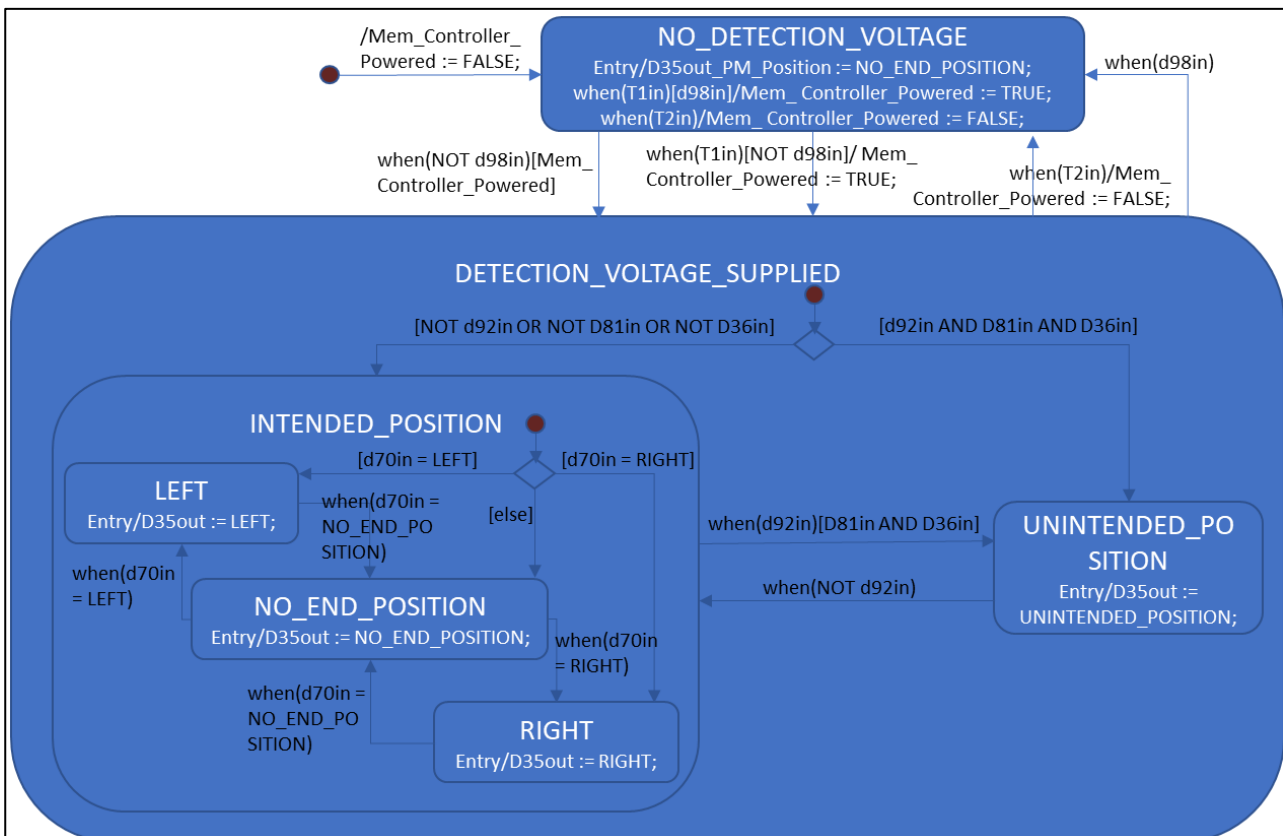


Figure 55: Statechart for observing the PM position as part of the PM simulation specification

For the PM simulation subsystem, no standardised specification was known. Building on the already defined P3 IF (see Figure 54) and on some imagination of the physical processes, a subsystem has been specified in the same style as the EULYNX specifications. This amounted to 3 interconnected

SysML blocks containing one statechart each (one of which is shown in Figure 55), and 2 configuration options.

6.2.3 Hardware and Software

Important requirements on the TC hardware are

- support for connecting the SCI, which is normally done via ethernet cable;
- some IF for displaying the GUI and operating the TC through it;
- some IF for connecting external PM;
- easy transportability (as a versatile TC); and
- low cost (to keep testing cost moderate even if many such TC are used).

It was decided to use a Raspberry Pi microcomputer (model 4B) running Raspberry Pi OS (version “Bookworm”, based on Debian 12), inside a solid case and together with a transportable 10.1” touchscreen connected via HDMI. This clearly fulfils the above requirements (using its GPIO IF for external PM) and provides even more possibilities (e.g. using WiFi as an alternative to Ethernet cable).

For the TC software, it was decided to mainly write C++ code by hand. Using C++ had the advantage of enabling object-oriented programming towards a standard realisation of SysML blocks as classes, while making it easy to incorporate the pre-existing RaSTA implementation in C. For the creation of the GUI, the mature and free-to-use gtkmm framework [28] in version 4.0 was chosen, which is an adaption of the original gtk framework to C++. gcc was available on the Raspberry Pi to compile the software.

6.3 DESIGN AND IMPLEMENTATION: FROM CONFIGURABLE AND MANIPULABLE FUNCTIONS TO STATECHART CODE

6.3.1 Design

6.3.1.1 Point Controller

Having the structured (interconnected SysML blocks) and detailed EULYNX Subsystem Point specification at hand and considering that performance optimization is not urgent for controllers which have timing requirements in the order of 100 milliseconds, it was decided that it is easiest to stick with the design given by the specification. The latter divides the point controller into the following ten functions:

- Manage overall controller state
- Manage SCI application level connection
- Receive telegrams via SCI
- Send telegrams via SCI
- Control point
- Control PM (to be instantiated for each PM)
- Observe point position

- Observe degraded point position
- Observe failed movement
- Observe ability to move

However, a few parts are left open intentionally by the specification, modelled as open ports of SysML blocks. Mostly, simply corresponding GUI controls were designed for direct input by the tester (e.g. buttons for power on/off, detection of a safety critical fault, controller-internal ability to move point). For some others, explicit connections to controller events were designed (if the tester activates “auto-finish booting”, booting is considered finished if a RaSTA connection has been established successfully; inputs for RaSTA disconnect and for telegram errors at the SCI are triggered by corresponding detection mechanisms).

6.3.1.2 PM Simulation

For the PM simulation design, besides the P3 IF defined by EULYNX, further external IF for power supply and to the physical point were defined which can be controlled by the tester via the GUI (e.g. buttons for local point operation to left/right, and for forcing unintended position representing an event like trailing of the point). Furthermore, the time it takes the PM to move from one end position to the other can be changed via the GUI.

The PM simulation subsystem has been structured into three logical components, each of which is responsible for a certain function of the PM simulation, defined by a statechart:

- The main function is the simulation of the PM movement, based on inputs from the P3 IF, the physical point IF, on drive power availability and the PM movement time. Internally, it maintains a precise position between the end positions, but its output is just one of the three states left, right or no end position. For its design, it was necessary to define what happens in special situations like conflicting inputs (e.g. move left AND move right) or change of the movement time during an ongoing movement. In case a PM is configured as “no drive capability” (i.e. being a point detector only), the logical component responsible for the PM movement will not be instantiated.
- The second function is the detection of the PM position (see the corresponding state machine in Figure 55), based on the output of the PM movement simulation, forced unintended position and detection power availability. Its output is left, right, no end position or unintended position. For the special case of detector only PM a design was chosen which derives position information (d70in input) from the neighbouring PM with drive capability.
- The third function is the detection of the PM's ability to move, based on drive power availability and on whether the PM is configured to detect inability to move at all. Its output is able/unable to move or not used. Its design was straight forward.

6.3.1.3 Connecting External PM

For connection of external PM (alternatively to using the PM simulation), it is foreseen by the design that the P3 IF can be exposed as an external IF of the TC. More precisely, it is exposed in parallel both for connecting real PM and on the GUI – see the two dashed lines in Figure 54. The GUI serves for displaying the status of the IF to the tester, but at the same time allows the tester to change inputs (all input changes from the two sources will be processed, i.e. relayed to point controller and GUI). The latter has been allowed to (a) enable simulation of literally any PM scenario directly by a tester,

without the need to connect a real device, and (b) enable to force or resolve special situations through the tester (e.g. force unintended position without damaging a connected real point). Currently, it is the tester's responsibility to use the GUI input feature with care and not to disturb/overwrite outputs of a real PM.

6.3.1.4 Configuration and Manipulation

A simple text-based parameter/value configuration file format has been designed to allow for a flexible use of different configurations. Firstly, it defines fixed-value data which cannot be changed during a test run: 15+2 parameters for point controller and PM configuration, such as for enabling redrive functionality of the controller or for the initial position of a PM. In addition, there are configuration parameters to choose between internal simulation or external use of a PM and between auto-powering the subsystems at test-component start-up or not. Secondly, it defines initial values for simulation parameters that can be changed during a test run via the GUI: 8+12 parameters for point controller and PM such as the initial internal ability to move for the controller, or whether left movement is initially blocked for a PM. Thirdly, it defines 9+9 initially displayed failure injection options for the GUI (see paragraph below). All parameters related to PM need to be specified for each configured PM.

A default configuration has been designed which allows ad-hoc usage of the TC. Without shutting down the TC, it is possible to reset (restart without changing the configuration) or reconfigure (restart choosing the default or a user-defined configuration) it. To enable successful RaSTA connections with any devices taking the role of an EULYNX interlocking, a separate RaSTA configuration XML file is used (see Figure 54).

Manifold manipulation capabilities have been planned (not yet fully implemented) for both the behaviour of the point controller and the PM simulation. For the controller, those functions with timing requirements specified by EULYNX (e.g. starting redriving) can be artificially delayed. And for both subsystems, outputs, event processing and statechart transitions can be manipulated by means of suppressing, inserting, exchanging, duplicating, swapping or delaying them. The desired fault can be selected and injected via the GUI.

6.3.2 Implementation

The logical components of the subsystems and their statecharts have been transformed into C++ classes in a structured way. In particular, member functions were created for the evaluation of changed inputs or timers, which may generate events; further member functions were created for the processing of these change or timeout events (see Figure 56 for an example transition which translates to the C++ code shown in Program Code 1). The event scheduling is managed by a central control class, based on one queue for each subsystem; also relaying of information at internal and external IF is done by this control class.

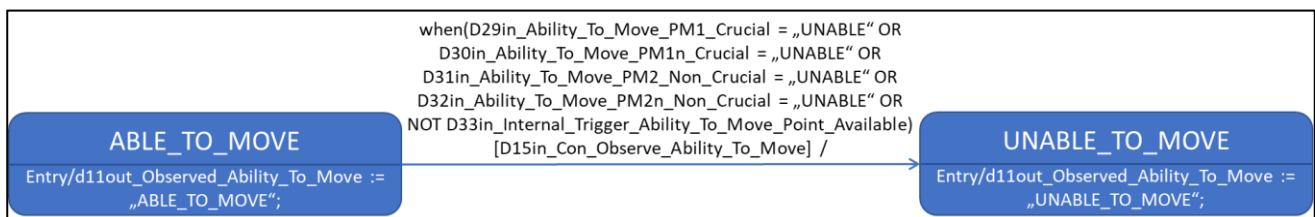


Figure 56: Example statechart transition, redrawn from [20]

```
// Process change event involving D29in to D33in ports
void F_Observe_Ability_To_Move::processChangeEvent_D29inToD33in(int event)
{
    if (currentState == ABLE_TO_MOVE) {
        // Transition ABLE_TO_MOVE -> UNABLE_TO_MOVE
        if (event == 0) {
            if (Con_Observe_Ability_To_Move) {
                [...]
                enterUnableToMove([...]);
                currentState = UNABLE_TO_MOVE;
            }
        }
    }
    [...]
}
```

Program Code 4: C++ code for the statechart transition from Figure 56, where event 0 stands for the change event “when(D29in_Ability...Point_Available)” from the transition

Finally, the GUI concept has been realised through a C++ class for the main window (see Figure 57). Besides the GUI elements for user in- and outputs, it manages the whole software application by calling functions of the control class in the correct order or in appropriate time intervals; this includes functions for initialization, main loop step, finalization, and RaSTA connection establishment.

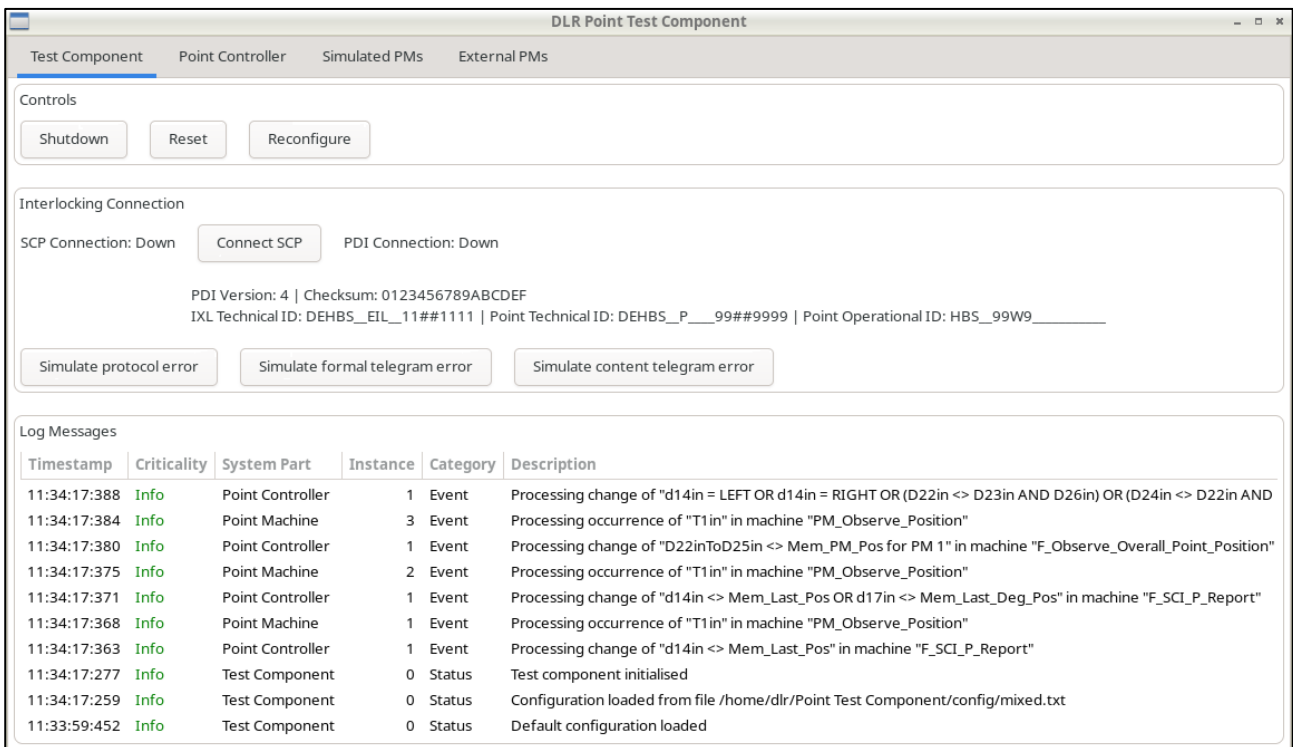


Figure 57: Screenshot of the first tab of the TC's GUI for overall monitoring and control

6.4 VALIDATION: TESTING THE TEST COMPONENT

6.4.1 Validation Set-Up at RailSiTe® Lab

For a basic validation of the TC, it has been tested using the DLR RailSiTe® lab. The core of the latter is a modular lab software for simulation and testing of different signalling subsystems. In the current context, the lab executed test cases and acted as the interlocking side of the SCI. The detailed definition of SCI telegrams (SCI-P version 4) to be generated by the lab and those expected back from the point controller has been implemented in the lab (telegram structure) and has been given as part of the test cases (telegram values). The RailSiTe® includes a RaSTA protocol implementation and runs on a computer with ethernet port, so that a connection with the TC can be easily established after defining some connection parameters (IP address, UDP ports, RaSTA IDs) on both sides.

6.4.2 Component Testing

With EULYNX Baseline 4 Release 1, a set of certification test cases for EULYNX point controllers has been delivered. However, this is neither complete (see [5]) nor fully compatible with Baseline 4 Release 2 point controllers (e.g. using telegrams in SCI-P version 3), so existing test cases have been adapted and completed for the purpose of TC validation. The test cases and testing are described in chapter 8.

7 TESTBED – TEST COMPONENT INTEGRATION

The Testbed – Test Component Integration describes the connection and interaction between the Hybrid Testbed (HTB) and the Test Component (TC) described in chapters 0 and 0 respectively. The integration of the HTB and the TC enable the validation and testing of the EULYNX Subsystem Point according to its SysML model, requirements and interface specifications. The integration established a communication link over the SCI-P interface using the UDP protocol, where the HTB operates as the server and the TC operates as the client. Secure and remote connection is achieved through a *Tailscale* VPN network, allowing both the HTB and the TC to exchange telegrams encoded and decoded in compliance with the *ISO/IEC 8859-1:1998* standard as described in the EULYNX system architecture specifications and the interface specifications. Figure 58 shows an overview of validation approach with HTB integration.

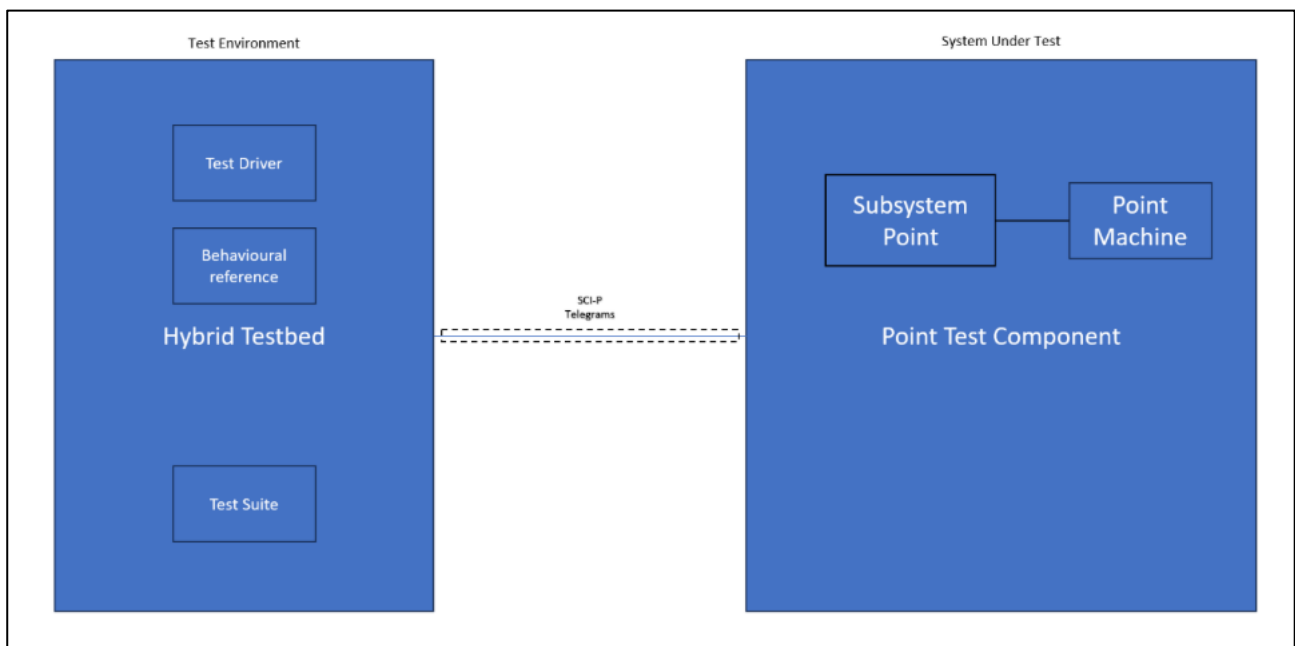


Figure 58: Overview of validation approach with HTB integration

Through this integration, the HTB executes and transmits commands to the TC and receives the corresponding status messages from the TC, enabling real-time interaction and testing of the test component's behaviour as the subsystem point. The embedded test suite within the HTB enables testing according to the EULYNX SysML state machines and integrates EULYNX test cases as part of its testing process, thereby ensuring that the SCI-P interface implementation conforms to both the model-based requirements and interface requirements. This integration setup provides a controlled yet realistic environment where the HTB and the TC collaboratively simulate field-level interlocking communication. The integration setup supports the end-to-end validation of a EULYNX implementation.

7.1 TESTBED INTEGRATION ARCHITECTURE

The Hybrid Testbed integration architecture is modelled in Capella [40] as part of the logical architecture of the hybrid testbed. The HTB integration architecture is described by the logical architecture blank shown in Figure 59 describing the integration of the TC into the HTB.

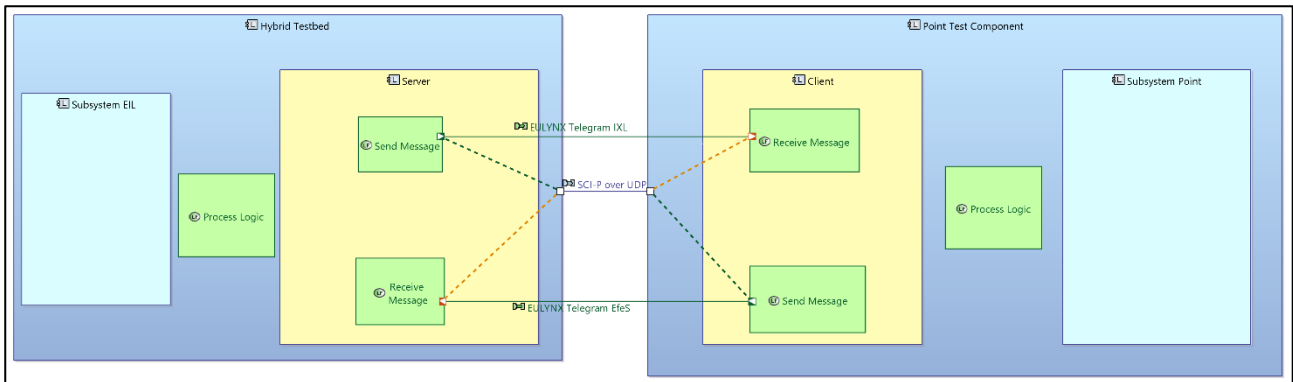


Figure 59: [LAB] Hybrid Testbed Integration Architecture

This integration architecture is further refined (see Figure 60) showing the connection and integration of the HTB and TC over the SCI-P interface described by the architecture and developments present in chapters 5 and 6 respectively.

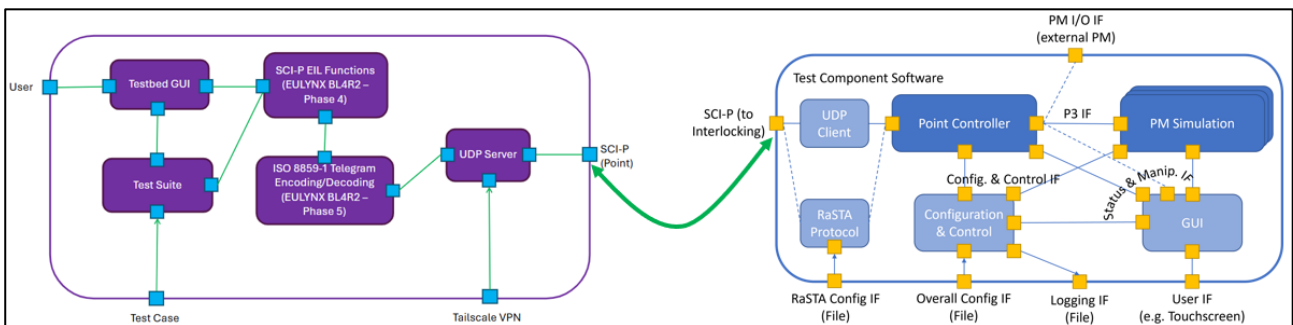


Figure 60: Refined Hybrid Testbed Integration Architecture

7.2 INTEGRATION TESTING

Integration testing between the HTB and the TC aims to validate the correct interaction between the HTB and the TC as defined in the integration architecture. It focuses on ensuring that the SCI-P interface as the functional and communication interface operates as intended when the model-based system behaviour is connected to real or simulated hardware components. The integration process validates that data exchange, telegram communication and behavioural responses conform to the EULYNX BL4R2 specifications. The following steps outline the systematic procedure adopted to conduct integration testing, from setup and execution to validation and reporting.

7.2.1 Define Scope and Objectives

- Identify the boundary between the HTB and the TC as shown in the integration architecture (see Figure).
- Specify the interface to be tested, which in the case of this deliverable is the SCI-P interface between the HTB and the TC.
- Define test objectives, such as communication establishment, checking correct telegram exchange and encoding/decoding and functional behaviour conformance.

7.2.2 Prepare the Testing Environment

- Configure both systems according to the integration architecture:
- HTB (Server): deploy behaviour program and application based on Python and GUI developed with *PyQt6*.
- TC (Client): deploy test component based on C++ and application based on *gtkmm* [28] representing the EULYNX subsystem point.
- Verify correct IP configuration and port assignment on both HTB and TC as defined in the VPN network.
- Ensure all dependencies (SysML model, code, test suite, configurations, SCI telegram definitions and logs) are loaded and synchronised.

7.2.3 Initialise Communication and Interface Validation

- Launch both the HTB and the TC and confirm initial handshake between them over UDP.
- Validate that the SCI telegrams exchanged over the SCI-P interface follows ISO:8859-1:1998 encoding as specified by EULYNX interface requirements.
- Monitor the communication to ensure that connections establishment, message exchange and checksum validation are successful.

7.2.4 Execute Test Cases

Execute considered EULYNX BL4R2 test cases for subsystem point input into the test suite of the HTB:

- Using the HTB GUI, simulate interlocking actions such as establishing PDI connection, commanding the point to move point position and receiving point position.
- Confirm correct feedback loop between HTB commands and TC responses, representing a realistic interlocking scenario.
- **Interface compliance tests:** Validate that all SCI commands and messages conform to EULYNX interface specification. Compare transmitted and received telegrams over the SCI-P interface with the reference model information flows.
- **Fault-injection tests:** Introduce controlled communication faults such as incorrect telegrams, disconnections and inducing state responses based on commands to examine HTB and TC resilience. Ensure induced errors are handled.

7.2.5 Log, Monitor and Analyse Results

- Collect communication logs, execution traces and test results.
- Analyse pass/fail outcomes of test cases.

7.2.6 Validate and Document Outcomes

- Confirm that the integrated HTB and TC achieve the behaviour as expected from the EULYNX BL4R2 Subsystem Point SysML model.

- Validate functional behaviour of subsystem point and SCI-P communication interface.
- Review test case outcomes and validation performed.
- Document test cases, test case outcomes, mapping to requirements and overview of the integration setup based on the integration architecture.

8 TEST CASES

Test cases for the validation approach have been developed from multiple sources. Each development carried out has its own sources for deriving test cases. These are mentioned in the sub-sections below.

8.1 TEST CASES FOR WHITE BOX APPROACH HYBRID TESTBED

• TC Val – Validation Test

Source: Here, the test cases are derived directly from the model-based requirements of Subsystem Point for BL3R8 described by transitions in the state machine. This ensures model-in-the-loop testing and enables model-based testing.

Test Case ID: TC.EuB.P.Val.2

EULYNX Requirement IDs: Eu.P.4674, Eu.P.4693, Eu.P.4687, Eu.P.4677, Eu.P.4684, Eu.P.4679, Eu.P.4681, Eu.P.4682, Eu.P.4685, Eu.P.4692, Eu.P.4925, Eu.P.4926 and Eu.P.4927

Test Case Description:

The EULYNX model-based requirements relevant for the test case is embedded into the hybrid testbed as a result of the SysML model integration with the IoT platform. The requirements to be tested by the hybrid testbed have been directly derived from the SysML model.

Test Data:

```
void StateMachine_FSCIP(){

    switch (currentState) {
        case INITIAL:
            Serial.println("||State: INITIAL||");
            cOp1_Init();
            Serial.print("@\n|\n|\n|/cOp1_Init();\n|\nv\n");
            currentState = ESTABLISHING_PDI_CONNECTION;
        case ESTABLISHING_PDI_CONNECTION:
            Serial.println("||State: ESTABLISHING_PDI_CONNECTION||");
            Serial.print("@\n|\n|\n|/\n|\nv\n");
            currentState = WAITING;
        case WAITING:
            Serial.println("||State: WAITING||");
            printThingProperties();
            .....
            .....
    }
}
```

Program Code 5: Excerpt of F_SCI_P stm implemented in C code

```
void state_PDI_CONNECTION_ESTABLISHED(){
    Serial.println("State: PDI_CONNECTION_ESTABLISHED");
    while (D50_F_SCI_EfeS_Gen_PDI_Connection_State == "ESTABLISHED"){
```

```

iotScan();

if (T1_Cd_Move_Point == "true"){

    DT10_Move_Target = DT1_Move_Point_Target;
    pointMove(DT10_Move_Target);
    putRequest("{\"esp_sensor\":\"\" + DT10_Move_Target + "\"}", fullRequestURL);

    T10_Move = "true";
    putRequest("{\"esp_T10_Move\":true}", fullRequestURL16);
    delay(100);
    T2_Msg_Point_Position = "false";
    putRequest("{\"esp_T10_Move\":false}", fullRequestURL16);
}

if (T20_Point_Position == "true"){
    T1_Cd_Move_Point = "false";
    delay(100);
    putRequest("{\"esp_T1_Cd_Move_Point\":false}", fullRequestURL5);

    DT2_Point_Position = DT20_Point_Position;
    putRequest("{\"esp_DT2_Point_Position\":\"\" + DT2_Point_Position + "\"}",
fullRequestURL14);

    T2_Msg_Point_Position = "true";
    putRequest("{\"esp_T2_Msg_Point_Position\":true}", fullRequestURL13);
    delay(100);
    T2_Msg_Point_Position = "false";
    putRequest("{\"esp_T2_Msg_Point_Position\":false}", fullRequestURL13);
}

...
}
if (D50_F_SCI_EfeS_Gen_PDI_Connection_State != "ESTABLISHED"){
    currentState = ESTABLISHING_PDI_CONNECTION;
}
}

```

Program Code 6: Excerpt of PDI_CONNECTION_ESTABLISHED state implemented in C code

Simulation is executed with controls from the SysML model and the IoT platform, including the state machine animation to check which transition is executed and which state the model and hardware is in.

Expected Outcome: According to the EULYNX Requirement IDs, the transitions and states of the SysML model of F_SCI-P should be executed in the C code for the prototype hardware during

program execution after the code is uploaded on the microcontroller while simultaneously the identical state machine animation of the model should be shown.

Actual Outcome: The transitions and states of the SysML model of F SCI-P should be executed in the C code for the prototype hardware during program execution after the code is uploaded on the microcontroller while simultaneously the identical state machine animation of the model is shown, according to the EULYNX Requirement IDs.

Result: Pass

• **TC 1 – Command RIGHT**

The test case **TC1 - Command RIGHT** replicates an ideal scenario when a point's position is changed successfully. The position of the point prior to the test case execution is preset to be 'LEFT' and the electronic Interlocking Subsystem is aware of this position. The ports *Move Point Target* and *Point Position* in the precondition row highlight this in Table 1: TC 1 – Command RIGHT. Figure 61 shows GUI of HTB displaying results of a successful validation test.

Test case			TC 1 - Command RIGHT		
Channel	Move_Point_Target (EIL)	Cd_Move_Point (EIL)	Msg_Point_Target (Sub.P)	Point_Position (Sub.P)	Msg_Timeout (Sub.P)
Precondition	LEFT	-	-	LEFT	-
Input	RIGHT	True	True	RIGHT	-
Post condition	RIGHT	True	True	RIGHT	-
Expectation	Success	Success	Success	Success	Success

Table 1: TC 1 – Command RIGHT

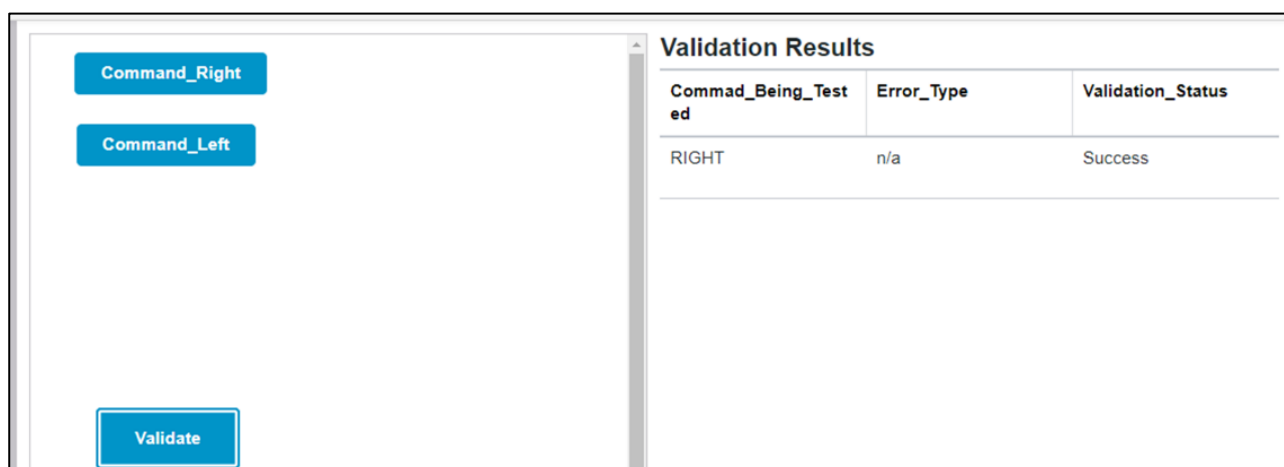


Figure 61: GUI of HTB displaying results of a successful validation test

The Electronic Interlocking Subsystem sends a command to the Subsystem Point to move the point blades to the 'RIGHT' through the SCI-P. First, the electronic interlocking subsystem transmits the intended direction through port *Move Point Target* using a string variable and then confirms with a

Boolean variable through port *Cd_Move_Point*. The point is moved to the right and the subsystem Point sends back an update of the changes to the electronic interlocking subsystem. The subsystem Point first sends the new direction of the point as a string variable through port *Point Position* and confirms the report transmission using a Boolean variable through port *Msg_Point_Target*. There is no expected data transfer in the port *Timeout* since all the messages are sent within the expected time. The input and post-condition rows reflect the activities detailed above.

The test bed checks the post conditions that are on the ports against its control values for a Command Right scenario. The expectation row shows the results the HTB is supposed to have after performing a validation check. When the HTB's GUI shows the status of the test after running TC1 as 'Success' it means that the HTB is performing the validation test as expected.

• **TC 2 – Command LEFT**

The test case **TC 2 – Command LEFT** is similar to the TC 1 but replicates an ideal scenario when a point's position is being changed successfully from 'RIGHT' to 'LEFT'. The ports *Move_Point_Target* and *Point_Position* highlight the preconditions. While the input row and post condition row reflect the activities and transfer of data over the SCI-P. This is shown in Table 2: TC 2 – Command LEFT.

The test bed checks the post conditions that are on the ports against its control values for a command LEFT scenario and it will declare the SCI-P information exchanges as 'success'. This expected result shows that the HTB is performing its validation test properly.

Test Case	TC 2 - Command LEFT				
Channel	Move Point Target (EIL)	Cd_Move_Point (EIL)	Msg_Point_Target (Sub.P)	Point_Position (Sub.P)	Msg_Timeout (Sub.P)
Precondition	RIGHT	-	-	RIGHT	-
Input	LEFT	True	True	LEFT	-
Post condition	LEFT	True	True	LEFT	-
Expectation	Success	Success	Success	Success	Success

Table 2: TC 2 – Command LEFT

• **TC 3 – Command RIGHT fault**

The test case **TC 3 - Command RIGHT fault** is derived from TC 1. It replicates a scenario when a point's position is being changed successfully but the change is not communicated back to the Subsystem Electronic Interlocking due to a malfunction in the SCI-P. The position of the point prior to the test case execution is preset to be 'LEFT' and the Electronic Interlocking subsystem is aware of this position. The ports *Move_Point_Target* and *Point_Position* in the precondition row highlight this. Table 3: TC 3 – Command RIGHT fault shows the full test case description.

The Electronic Interlocking Subsystem sends a command to the Subsystem Point through the SCI-P to move the point blades to the 'RIGHT'. First the Electronic Interlocking Subsystem transmits the intended direction through port *Move_Point_Target* as a string variable and then confirms with a boolean variable through port *Cd_Move_Point* as expected. The point is moved to the right. However, the Subsystem Point does not successfully send back an update of the changes to the Electronic Interlocking Subsystem. A timeout is triggered because the status update of the point's position is not sent in the expected time. The input row and post condition row reflect the activities this test case evokes.

Test Case			TC 3 - Command RIGHT fault		
Channel	Move_Point_Target (EIL)	Cd_Move_Point (EIL)	Msg_Point_Target (Sub.P)	Point_Position (Sub.P)	Msg_Timeout (Sub.P)
Precondition	LEFT	-	-	LEFT	-
Input	RIGHT	True	-	-	-
Post condition	RIGHT	True	-	LEFT	True
Expectation	Success	Success	Failure	Failure	Failure

Table 3: TC 3 – Command RIGHT fault

The test bed checks the post conditions that are on the ports against its control variables for what it expects to be a successful command RIGHT scenario. The expectation row shows the results of the test bed's validation check which is communicated to the operator via the HTB's GUI. The test bed expects the communication over the SCI-P to be that of a success scenario and therefore it will flag the errors and declare that there are inconsistencies in the information flows over the SCI-P. The HTB lets the operator know which ports are the sources of the errors (see Figure 62 below).

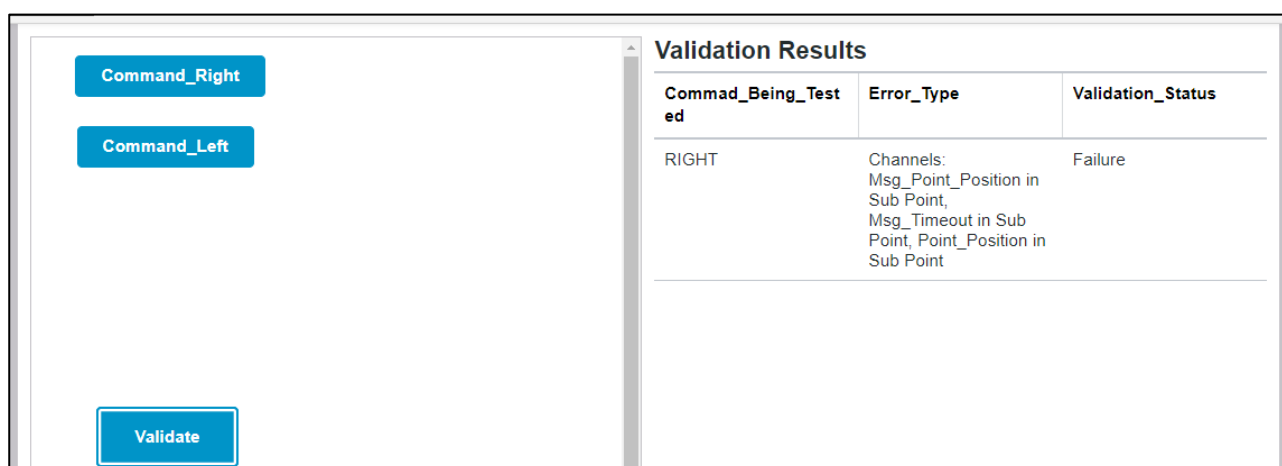


Figure 62: GUI of HTB displaying results of a failed validation test

- **TC 4 – Command LEFT fault.**

The fourth test case TC 4 – Command LEFT fault in Table 4: TC 4 – Command LEFT fault is similar to TC 3. It replicates a scenario when a point's position is being changed successfully from 'RIGHT

to 'LEFT' but the change is not communicated back to the Electronic Interlocking Subsystem due to a malfunction in the SCI-P.

The ports Move_Point_Target and Point_Position highlight the preconditions. While the input row and post condition row reflect the activities during the transfer of data over the SCI-P. The test bed checks the post conditions that are on the ports against its control values for command LEFT scenario. The test bed expects the communication over the SCI-P to be that of a success scenario and declares that there are inconsistencies in the information flows over the SCI-P. The HTB lets the operator know which ports are the sources of the errors.

Test Case			TC 4 – Command LEFT fault		
Channel	Move_Poin _Target (EIL)	Cd_Move_ Point (EIL)	Msg_Point _Target (Sub.P)	Point_Posit ion (Sub.P)	Msg_Timeo ut (Sub.P)
Preconditio n	RIGHT	-	-	RIGHT	-
Input	LEFT	true	-	-	-
Post condition	LEFT	true	-	RIGHT	true
Expectation	Success	Success	Success	Failure	Failure

Table 4: TC 4 – Command LEFT fault

In all four cases, the HTB successfully compares the Post conditions with its control values. With TC 1 and TC 2, the testbed is able to affirm that the SCI-P is performing as expected. With TC 3 and TC 4, the test bed proves that it can catch when there is something wrong in the SCI-P's performance. The HTB's outputs align with the expectation rows of the test case.

8.2 TEST CASES FOR BLACK BOX APPROACH HYBRID TESTBED

The test suite with its respective test cases and test driver that enables automatic testing of the SCI-P interface validates the correct behaviour of the subsystem point by controlling, monitoring and testing the position of the point within the HTB environment.

The test suite contains three core test cases as shown in the Table 5 below, two validating normal functional behaviour and one performing fault injection to confirm error detection and handling robustness.

The objectives of these test cases are:

- To validate that commands from the HTB to move the point (LEFT, RIGHT) produce the expected responses and messages from the subsystem point.
- To confirm that the SUT correctly identifies and handles invalid or fault-injection commands.
- To ensure executed commands and result reports are in accordance with the EULYNX model-based requirements.

These test cases form part of the functional validation in the test suite in accordance with the *ISO/IEC/IEEE 29119-3:2013 – Software Testing* standard [32] and are aligned with the *ISTQB* scheme [33].

TestCase_ID	Test Description	PreCondition	Steps	Expected Result	PostCondition	Actual Result	Pass/Fail
TC-001	Testing Point Movement - Right	Behaviour initialised and GUI ready	RIGHT	RIGHT, NOT_APPLICABLE	Subsystem Point responded correctly to RIGHT command	RIGHT, NOT_APPLICABLE	Passed
TC-002	Testing Point Movement - Left	Behaviour initialised and GUI ready	LEFT	LEFT, NOT_APPLICABLE	Subsystem Point responded correctly to LEFT command	LEFT, NOT_APPLICABLE	Passed
TC-003	Testing Point Movement - Fault	Behaviour initialised and GUI ready	ERROR	Failed Result	Subsystem Point handled fault correctly	LEFT, NOT_APPLICABLE	Failed (expected failure)

Table 5: Black Box HTB automatically executed test cases

Test Run:

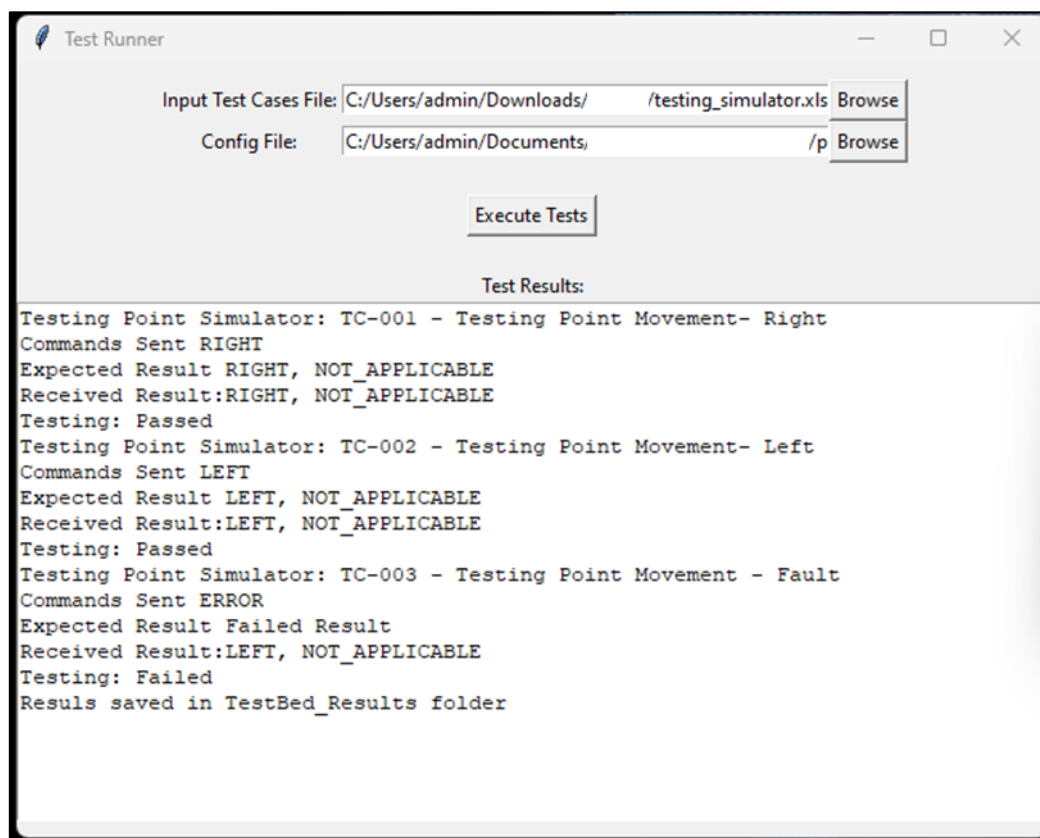


Figure 63: Automatic test case execution in test suite of HTB with test driver

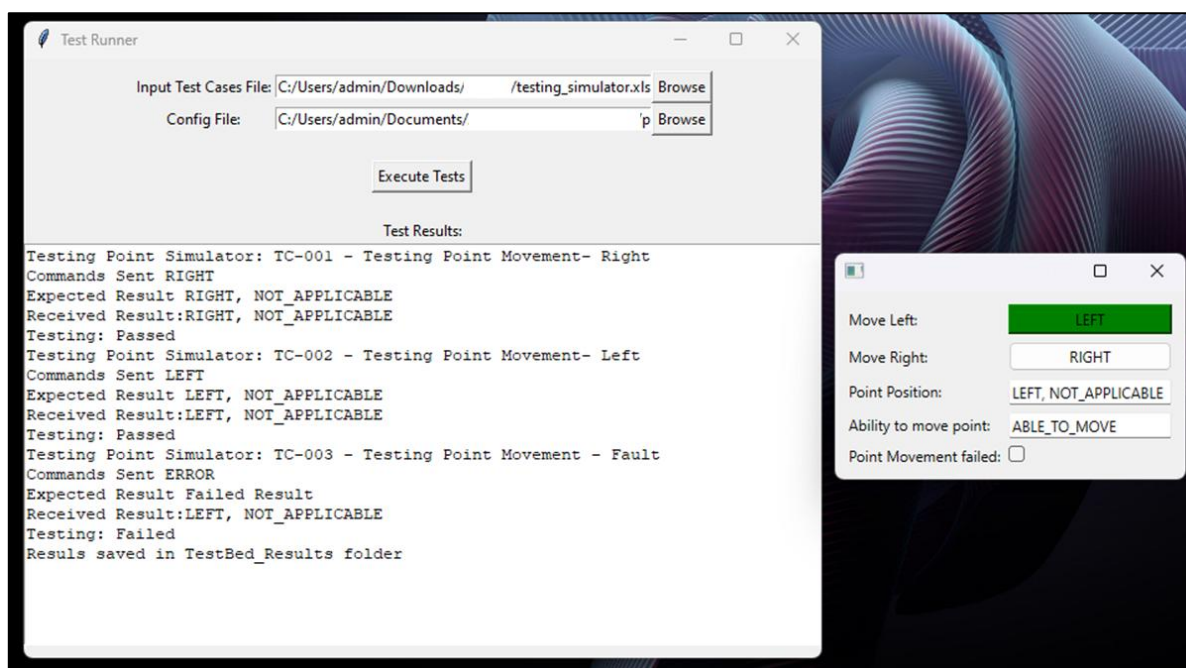


Figure 64: Automatic test case execution in test suite of HTB with HTB GUI

TestCase_ID	Test Description	PreCondition	Steps	Expected Result	PostCondition	Actual Result	Pass/Fail
TC-001	Testing Point Movement- Right		RIGHT	RIGHT, NOT_APPLICABLE		RIGHT, NOT_APPLICABLE	Passed
TC-002	Testing Point Movement- Left		LEFT	LEFT, NOT_APPLICABLE		LEFT, NOT_APPLICABLE	Passed
TC-003	Testing Point Movement - Fault		ERROR	Failed Result		LEFT, NOT_APPLICABLE	Failed

Figure 65: Results sheet saved after automatic test case execution by HTB

Key observations from the test run are:

- The functional and fault-injection test cases executed as planned.
- TC-001 and TC-002 validated correct subsystem point response to commands to move the point (LEFT and RIGHT).
- NOT_APPLICABLE means that a degraded point position is not applicable for the point position.
- TC-003 injected a fault (erroneous command) which the subsystem Point correctly handled and flagged as a failure.
- Test results matched the expected outcome. Normal operation scenarios passed as expected, the fault-injection scenario [Section 7.2.4] failed as expected, demonstrating error handling robustness.
- The test driver successfully logged results and saved the output without crashing the HTB or unexpected errors.

8.3 TEST CASES FOR POINT TEST COMPONENT AT RAILSiTe® LAB

Before the TC was connected to the HTB and tested using the HTB as reported above, it has undergone a basic validation. To this end, it was connected to DLR's *RailSiTe®* lab – this time including the RaSTA protocol at the SCI-P – and a number of test cases was executed. Five main test cases have been defined for use with the default configuration of the TC:

1. Successfully commanding the point from left to right end position
2. Commanding the point from right to left end position, taking longer than allowed
3. Reporting inability to move due to PM output “unable to move”
4. Reporting ability to move (again) due to PM output „ability to move not used“
5. Reporting degraded point position due to non-crucial PM output „no end position“

These test cases have been arranged into two test sequences which are executable by the lab, including additional prefix test cases for establishing a connection at the SCI. During test execution, the TC GUI has been used by the tester to manipulate PM simulations so that the PM outputs were delayed (test case 2) or changed (test cases 3-5) as required. The log messages recorded by the TC for both test sequences have been saved (automatically) into a text file (see Table 6) for detailed evaluation of the test runs.

Table 6 shows a log excerpt from a test run. It is to be read bottom-up. The central block of five log entries shows a test sequence where the second PM's drive voltage becomes insufficient (encoded as Boolean input d99in, see bottom step of block). The point controller is notified that the PM is unable to move (the following step upwards, compare the transition from Figure 56), processes that change and finally sends a telegram informing the interlocking that the point is considered unable to move (topmost step of block).

Timestamp	Criticality	System Part	Instance	Category	Description
03:05:49:126	Info	Test Component	0	Status	Test component finished
03:05:44:114	Info	IXL - Controller	0	Status	SCP connection closed
[...]					
03:04:01:805	Info	IXL - Controller	0	Data <-	Sent "Ability To Move Point(ability=UNABLE)" telegram
03:04:01:790	Info	Point Controller	1	Event	Processing change of "d11in = UNABLE_TO_MOVE" in machine "F_Control_Point"
03:04:01:776	Info	Point Controller	1	Event	Processing change of "d11in = UNABLE_TO_MOVE" in machine "F_SCI_P_Report"
03:04:01:761	Info	Point Controller	1	Event	Processing change of "D29in = UNABLE OR D30in = UNABLE OR D31in = UNABLE OR D32in = UNABLE OR NOT D33in" in machine "F_Observe_Ability_To_Move"
03:04:01:747	Info	Point Machine	2	Event	Processing change of "NOT d99in" in machine "PM_Observe_Ability_To_Move"
[...]					
03:03:49:202	Info	IXL - Controller	0	Status	SCP connection established successfully
[...]					

03:03:48:821	Info	Test Component	0	Status	Test component initialised
03:03:03:895	Info	Test Component	0	Status	Default configuration loaded

Table 6: Log excerpt from a test run

8.4 TEST CASES FOR HYBRID TESTBED – TEST COMPONENT INTEGRATION

The integration testing of the HTB with the TC at the SCI-P interface has considered four test cases for validation.

Three of the test cases considered are EULYNX BL4R2 test cases, one for establishing PDI connection at the generic SCI with test case ID TC_Eu.Gen-SCI.444.01, the second for reporting point status at the SCI-P interface with test case ID TC_Eu.P.5787.01 and the third for SCI-P with test case ID TC_Eu.P.6714.01 to command the point position to move from left to right and to receive the point position during and after the point movement.

The EULYNX test cases TC_Eu.P.5787.01 and TC_Eu.P.6714.01 were considered in this section, while the integration testing was performed using the corresponding TC_Eu.P.5789.01 and TC_Eu.P.6722.01 EULYNX BL4R2 test cases. These two pairs of test cases are functionally similar for the SCI-P IF, with the difference being that the first pair of test cases applies for a single point machine and the second pair of test cases applies for multiple point machines. The first pair of test cases were considered in this section to keep the documentation concise, as the test logic and expected results are similar for both pairs of test cases. Four point machines were considered for the integration testing.

The fourth test case with test case ID TC-004 is a fault test case sending a telegram at a different state with no transition that reacts to the telegram to ensure that the message received of the connection does not change.

The integration test cases are aligned with the *ISO/IEC/IEEE 29119-3:2013 – Software Testing* standard [32] and the *ISTQB* [33] scheme.

Test Case ID: TC_Eu.Gen-SCI.444.01

Test Case Description:

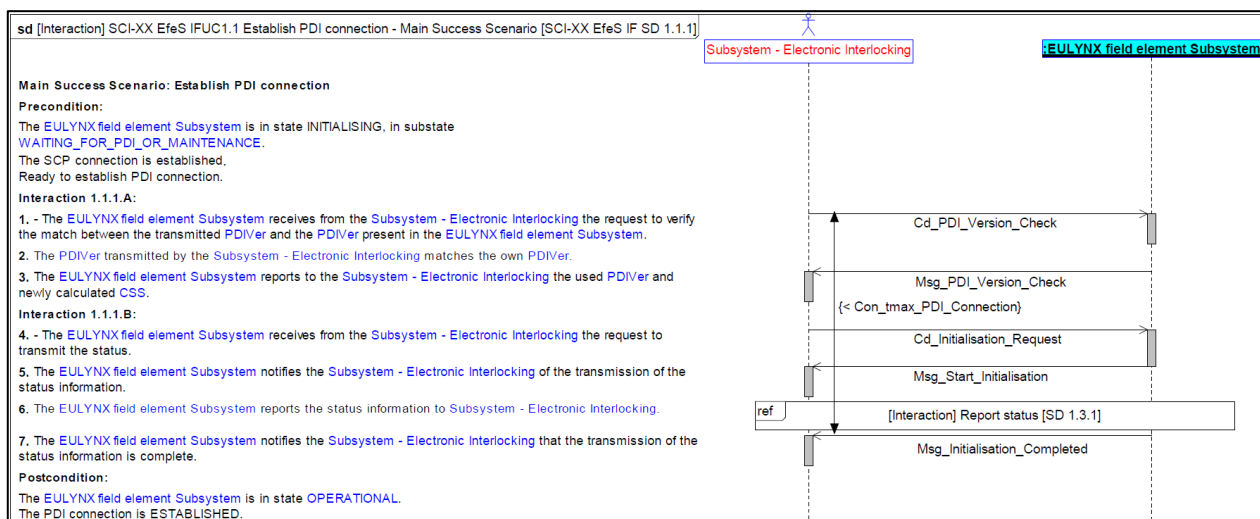


Figure 66: TC_Eu.Gen-SCI.444.01 test case description

Test Case:

Step No.	IF	Direction	Step Name	Description	Test Data	Result
Config			CONFIG_SETTING	CON_TMAX_PDI_CONNECTION is 20s.	CON_TMAX_PDI_CONNECTION = 20000;	
Pre	SCP	Output	SCP_CONNECTION	The SCP connection is established.	GEN.SCP_CONNECTION = ESTABLISHED;	
Pre	PDI	Output	PDI_CONNECTION	Ready to establish PDI connection.	GEN.PDI_CONNECTION = READY_FOR_PDI;	
1	PDI	Input	Send command Cd_PDI_Version_Check	The EULYNX field element Subsystem receives from the Subsystem - Electronic Interlocking the request to verify the match between the transmitted PDIVer and the PDIVer present in the EULYNX field element Subsystem.	COMMAND: PDI_VERSION_CHECK; PDI[00] = 0x40; PDI[01..02] = 0x0024; PDI[03..22] = DEHBS_P_99##9999; PDI[23..42] = DEHBS_EIL_11##1111; PDI[43] = 0x04;	Pass
2	PDI		Version match	The PDIVer transmitted by the Subsystem - Electronic Interlocking matches the own PDIVer.		Pass
3	PDI	Output	Send message Msg_PDI_Version_Check	The EULYNX field element Subsystem reports to the Subsystem - Electronic Interlocking the used PDIVer and newly calculated CSS.	MESSAGE: PDI_VERSION_CHECK; PDI[00] = 0x40; PDI[01..02] = 0x0025; PDI[03..22] = DEHBS_EIL_11##1111; PDI[23..42] = DEHBS_P_99##9999; PDI[43] = <P_RESULT_PDI_VERSION_CHECK>; PDI[44] = <P_PDI_VERSION_OF_SENDER>; PDI[45] = <P_CHECKSUM_LENGTH>; PDI[46..XX] = <P_CHECKSUM_DATA>;	Pass
4	PDI	Input	Send command Cd_Initialisation_Request	The EULYNX field element Subsystem receives from the Subsystem - Electronic Interlocking the request to transmit the status.	COMMAND: INITIALISATION_REQUEST; PDI[00] = 0x40; PDI[01..02] = 0x0021; PDI[03..22] = DEHBS_P_99##9999; PDI[23..42] = DEHBS_EIL_11##1111;	Pass
5	PDI	Output	Send message Msg_Start_Initialisation	The EULYNX field element Subsystem notifies the Subsystem - Electronic Interlocking of the transmission of the status information.	MESSAGE: START_INITIALISATION; PDI[00] = 0x40; PDI[01..02] = 0x0022; PDI[03..22] = DEHBS_EIL_11##1111; PDI[23..42] = DEHBS_P_99##9999;	Pass
6	PDI		SubSUC1.3: Report status	The EULYNX field element Subsystem reports the status information to Subsystem - Electronic Interlocking.		Pass
7	PDI	Output	Send message Msg_Initialisation_Completed	The EULYNX field element Subsystem notifies the Subsystem - Electronic Interlocking that the transmission of the status information is complete.	MESSAGE: INITIALISATION_COMPLETED; PDI[00] = 0x40; PDI[01..02] = 0x0023; PDI[03..22] = DEHBS_EIL_11##1111; PDI[23..42] = DEHBS_P_99##9999;	Pass
Post	PDI	Output	PDI_CONNECTION	The PDI connection is ESTABLISHED.	GEN.PDI_CONNECTION = READY_FOR_PDI;	Pass

Figure 67: TC_Eu.Gen-SCI.444.01 test case

EULYNX Requirement IDs: Eu.SCI-XX.PDI.70, Eu.SCI-XX.PDI.72, Eu.SCI-XX.PDI.73, Eu.SCI-XX.PDI.53, Eu.SCI-XX.PDI.74, Eu.SCI-XX.PDI.75, Eu.SCI-XX.PDI.76, Eu.SCI-XX.PDI.78, Eu.SCI-XX.PDI.80, Eu.SCI-XX.PDI.81, Eu.SCI-XX.PDI.125, Eu.SCI-XX.PDI.82, Eu.SCI-XX.PDI.83, Eu.SCI-XX.PDI.84, Eu.SCI-XX.PDI.86, Eu.SCI-XX.PDI.87, Eu.SCI-XX.PDI.88, Eu.SCI-XX.PDI.90, Eu.SCI-XX.PDI.91, Eu.SCI-XX.PDI.92, Eu.SCI-XX.PDI.116, Eu.SCI-XX.PDI.118, Eu.SCI-XX.PDI.93, Eu.SCI-XX.PDI.94, Eu.SCI-XX.PDI.117, Eu.SCI-XX.PDI.98, Eu.SCI-XX.PDI.99, Eu.SCI-XX.PDI.126, Eu.SCI-XX.PDI.100, Eu.SCI-XX.PDI.101, Eu.SCI-XX.PDI.105, Eu.SCI-

XX.PDI.106, Eu.SCI-XX.PDI.127, Eu.SCI-XX.PDI.107, Eu.SCI-XX.PDI.108, Eu.SCI-XX.PDI.113, Eu.SCI-XX.PDI.112, Eu.SCI-XX.PDI.128, Eu.SCI-XX.PDI.114, Eu.SCI-XX.PDI.115

Test Run Result: Pass

Test Case ID: TC_Eu.P.5787.01

Test Case Description:

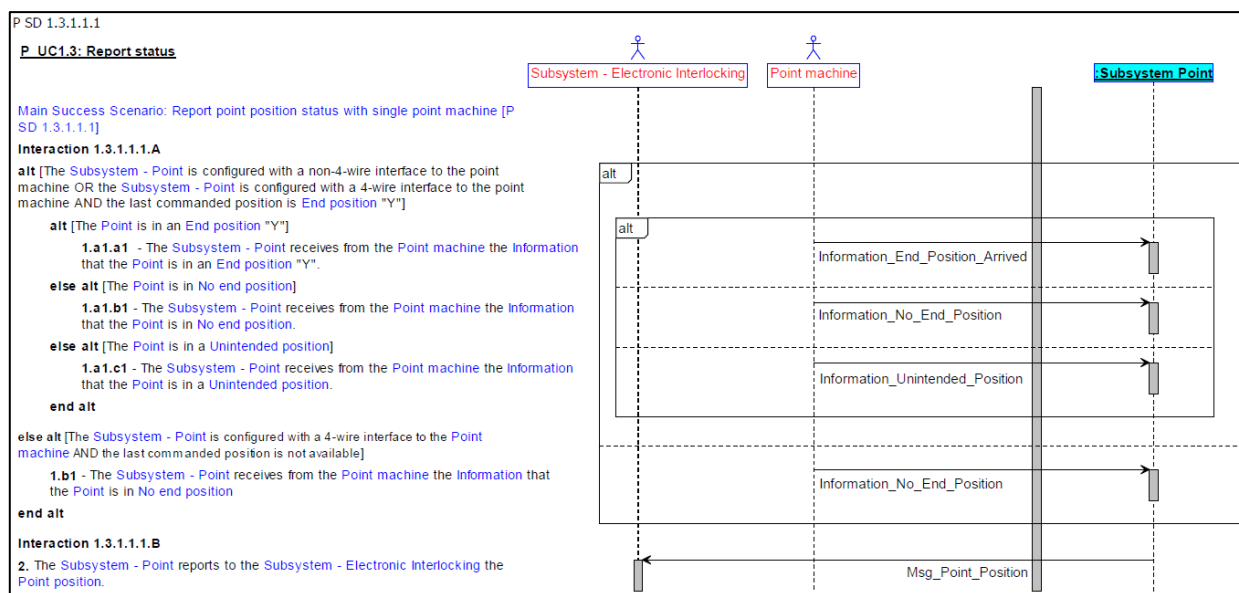


Figure 68: TC_Eu.P.5787.01 test case description

Test Case:

Step No.	IF	Direction	Step Name	Description	Test Data	Result
Config			CONFIG_SETTING	P3.NUMBER_OF_POINTMACHINES	P3.NUMBER_OF_POINTMACHINES = 1;	
Config			CONFIG_SETTING	The Subsystem - Point is configured with a non-4-wire interface to the point machine	P3.INTERFACE_4_WIRE = NON-4-WIRE;	
Pre	P3	Input	PHYSICAL_INTERFACE	Current point position is LEFT.	P3in.POINT_POSITION[1] = LEFT;	
Pre	P3	Input	PHYSICAL_INTERFACE	Point ability to move is ABLE.	P3in.ABILITY_TO_MOVE[1] = ABLE;	
Pre	P3	Output	PHYSICAL_INTERFACE	Point moving status is STOP MOVING.	P3out.MOVE_TARGET[1] = STOP_MOVING;	
1	P3	Input	Point machine status	The Subsystem - Point receives from the Point machine the Information that the Point is in No end position	P3in.POINT_POSITION[1] = LEFT;	Pass
2	PDI	Output	Send message Msg_Point_Position	The Subsystem - Point sends a Message to the Subsystem - Electronic Interlocking indicating the Point position.	MESSAGE: POINT_POSITION; PDI[00] = 0x40; PDI[01..02] = 0x000B; PDI[03..22] = <P_P_ID_OP>; PDI[23..42] = <P_ILS_ID_TEC>; PDI[43] = <P_REPORTED_POINT_POSITION>; PDI[44] = <P_REPORTED_DEGRADED_POINT_POSITION>;	Pass
Post	P3	Input	PHYSICAL_INTERFACE	Current point position is LEFT.	P3in.POINT_POSITION[1] = LEFT;	Pass
Post	P3	Input	PHYSICAL_INTERFACE	Point ability to move is ABLE.	P3in.ABILITY_TO_MOVE[1] = ABLE;	Pass
Post	P3	Output	PHYSICAL_INTERFACE	Point moving status is STOP MOVING.	P3out.MOVE_TARGET[1] = STOP_MOVING;	Pass

Figure 69: TC_Eu.P.5787.01 test case

EULYNX Requirement IDs: Eu.P.905, Eu.SCI-P.PDI.184, Eu.SCI-P.PDI.185, Eu.SCI-P.PDI.186, Eu.SCI-P.PDI.187, Eu.SCI-P.PDI.188, Eu.SCI-P.PDI.190, Eu.SCI-P.PDI.191, Eu.SCI-P.PDI.192, Eu.SCI-P.PDI.193, Eu.SCI-P.PDI.279, Eu.SCI-P.PDI.280, Eu.SCI-P.PDI.281, Eu.SCI-P.PDI.283, Eu.SCI-P.PDI.282

Test Run Result: Pass

Test Case ID: TC-004

Test Case:

Step No.	Test Description	PreCondition	Step	Expected Result	PostCondition	Actual Result	Result
1	Fault Injection	HTB PDI connection is in DISCONNECTED_NO_SCP state	HTB sends Command Initialisation Request to TC	No reaction or response from TC	HTB PDI connection is in DISCONNECTED state	No reaction or response from TC	Passed
2	Normal Functional Test	HTB PDI connection is in ESTABLISHING state	HTB sends Command PDI Version Check to TC	HTB receives Message PDI Version Check from TC	HTB PDI connection is in WAITING_FOR_VERSION_CHECK state	HTB receives Message PDI Version Check from TC	Passed

Figure 70: TC-004 test case

Test Run Result: Pass

Test Case ID: TC_Eu.P.6714.01

Test Case Description:

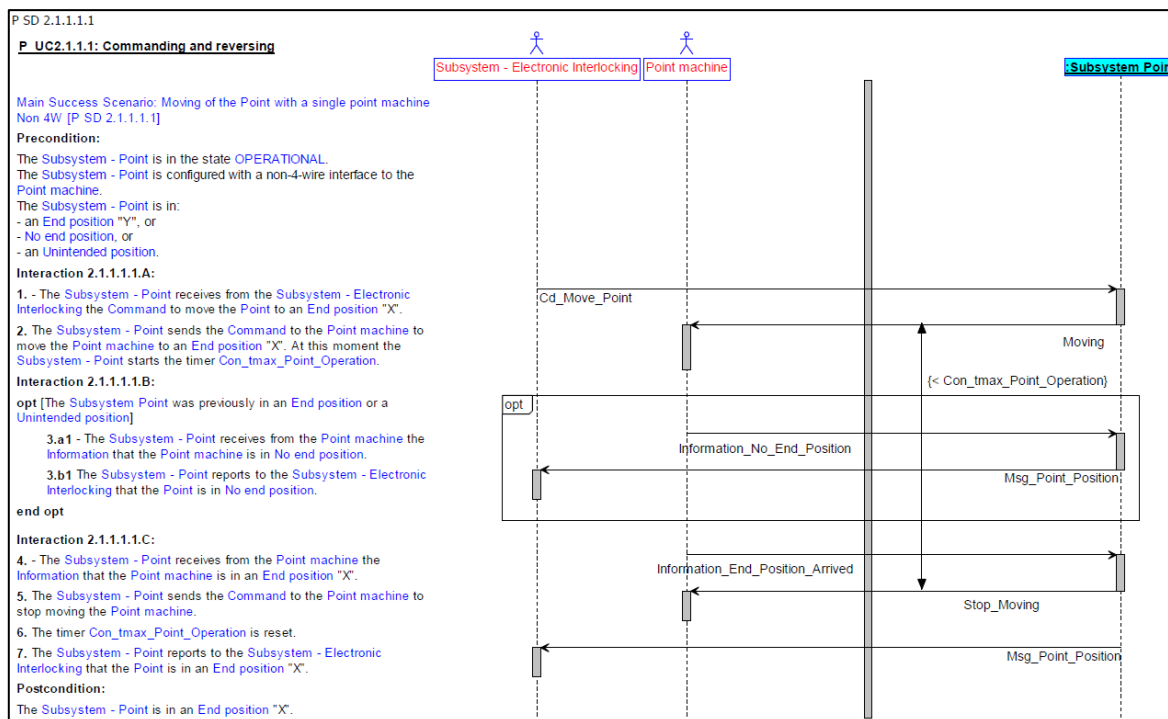


Figure 71: TC_Eu.P.6714.01 test case description

Test Case:

Step No.	IF	Direction	Step Name	Description	Test Data	Result
Config			CONFIG_SETTING	P3.NUMBER_OF_POINTMACHINES	P3.NUMBER_OF_POINTMACHINES = 1;	
Config			CONFIG_SETTING	The Subsystem - Point is configured with a non-4-wire interface to the point machine	P3.INTERFACE_4_WIRE = NON-4-WIRE;	
Pre	P3	Input	PHYSICAL_INTERFACE	Current point position is LEFT.	P3in.POINT_POSITION[1] = LEFT;	
Pre	P3	Input	PHYSICAL_INTERFACE	Point ability to move is ABLE.	P3in.ABILITY_TO_MOVE[1] = ABLE;	
Pre	P3	Output	PHYSICAL_INTERFACE	Point moving status is STOP MOVING.	P3out.MOVE_TARGET[1] = STOP_MOVING;	
1	PDI	Input	Sends Command Cd_Move_Point	The Subsystem - Electronic Interlocking sends a Command to the Subsystem - Point to move the Point to an End position "RIGHT".	COMMAND: MOVE_POINT; PDI[00] = 0x40; PDI[01..02] = 0x0001; PDI[03..22] = DEHBS_P____99##9999; PDI[23..42] = HBS_99W9_____ PDI[43] = 0x01;	Pass
2	P3	Output	Moving	The Subsystem - Point sends the Command to the Point machine to move the Point machine to an End position "RIGHT". At this moment the Subsystem - Point starts the timer Con_tmax_Point_Operation.	P3out.MOVE_TARGET[1] = MOVE_RIGHT;	Pass
3	P3	Input	Point machine status	The Subsystem - Point receives from the Point machine the Information that the Point machine is in an End position "RIGHT".	P3in.POINT_POSITION[1] = END_POSITION_RIGHT;	Pass
4	P3	Output	Stop moving	The Subsystem - Point sends the Command to the Point machine to Stop moving the Point machine.	P3out.MOVE_TARGET[1] = STOP_MOVING;	Pass
5	PDI	Output	Send message Msg_Point_Position	The Subsystem - Point sends a Message to the Subsystem - Electronic Interlocking indicating the Point position.	MESSAGE: POINT_POSITION; PDI[00] = 0x40; PDI[01..02] = 0x000B; PDI[03..22] = HBS_99W9_____ PDI[23..42] = DEHBS_P____99##9999; PDI[43] = 0x01; PDI[44] = 0xFF;	Pass
Post	P3	Input	PHYSICAL_INTERFACE	Current point position is RIGHT.	P3in.POINT_POSITION[1] = RIGHT;	Pass
Post	P3	Input	PHYSICAL_INTERFACE	Point ability to move is ABLE.	P3in.ABILITY_TO_MOVE[1] = ABLE;	Pass
Post	P3	Output	PHYSICAL_INTERFACE	Point moving status is STOP MOVING.	P3out.MOVE_TARGET[1] = STOP_MOVING;	Pass

Figure 72: TC_Eu.P.6714.01 test case

EULYNX Requirement IDs: Eu.SCI-P.PDI.162, Eu.SCI-P.PDI.163, Eu.SCI-P.PDI.164, Eu.SCI-P.PDI.165, Eu.SCI-P.PDI.167, Eu.SCI-P.PDI.168, Eu.P.907, Eu.P.904, Eu.P.909, Eu.SCI-P.PDI.184, Eu.SCI-P.PDI.185, Eu.SCI-P.PDI.186, Eu.SCI-P.PDI.187, Eu.SCI-P.PDI.188, Eu.SCI-P.PDI.190, Eu.SCI-P.PDI.191, Eu.SCI-P.PDI.192, Eu.SCI-P.PDI.193, Eu.SCI-P.PDI.279, Eu.SCI-P.PDI.280, Eu.SCI-P.PDI.281, Eu.SCI-P.PDI.283, Eu.SCI-P.PDI.282

Test Run Result: Pass

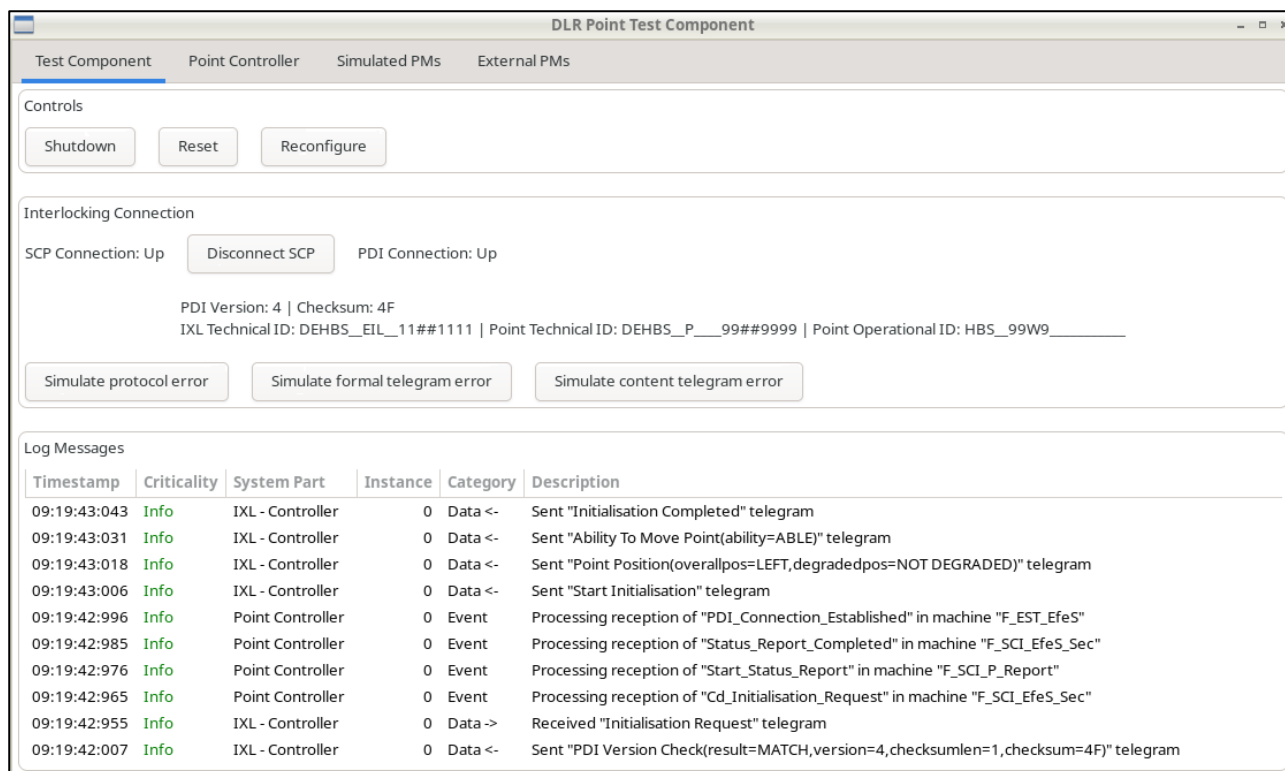


Figure 76: TC with point status reported and PDI connection established

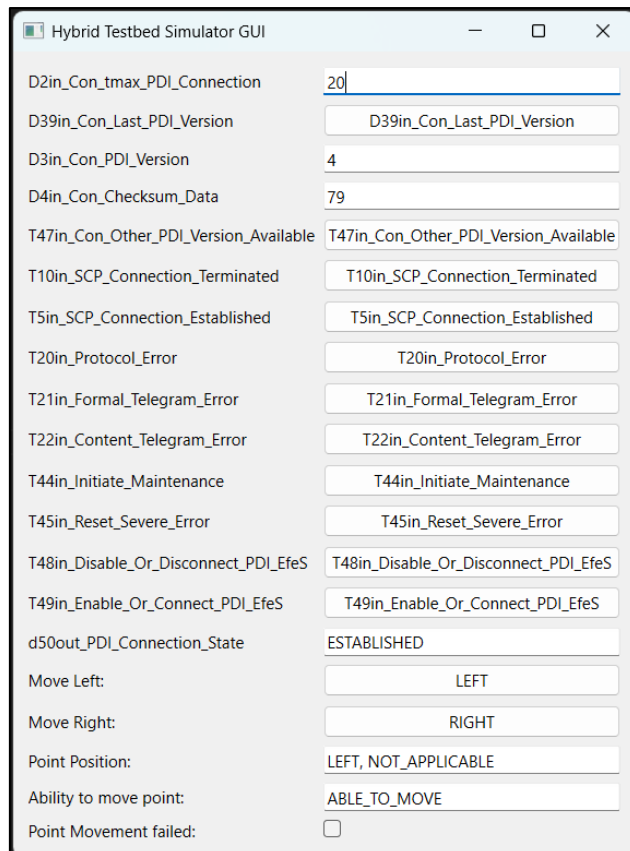


Figure 77: HTB with point status LEFT reported from TC and PDI connection established

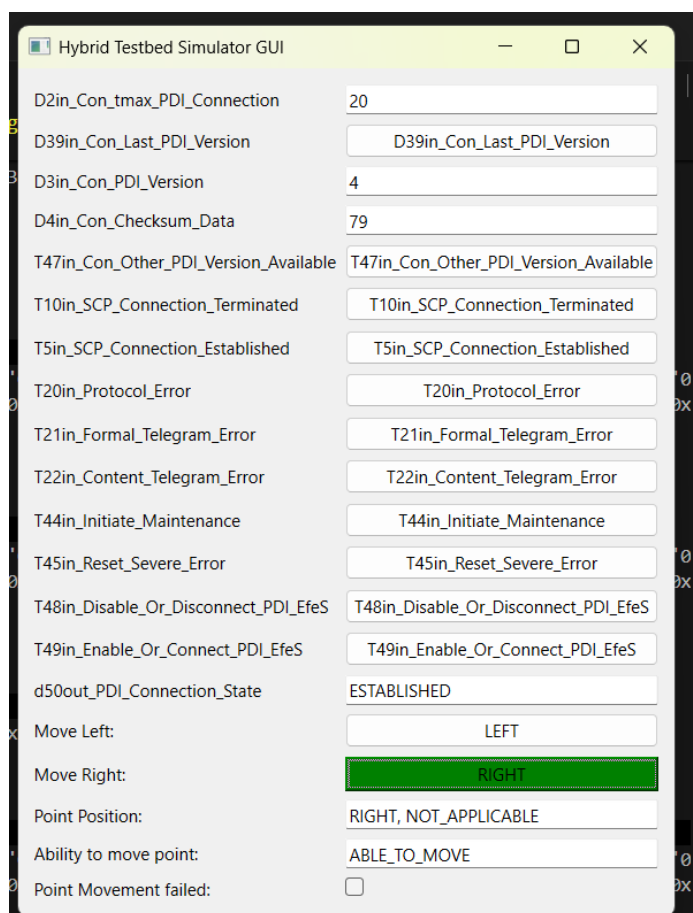
Test Run: TC_Eu.P.6714.01

```
Server: @\x0\x1DEHBS EIL 11##1111DEHBS P 99##9999\x2
Server: @\x0$DEHBS EIL 11##1111DEHBS P 99##9999\x4
Client: @\x0%DEHBS P 99##9999DEHBS EIL 11##1111\x2\x4\x10
Server: @\x0!DEHBS EIL 11##1111DEHBS P 99##9999
Client: @\x0"DEHBS P 99##9999DEHBS EIL 11##1111
Client: @\x0\xBHBS 99W9 DEHBS EIL 11##1111\x2\x3
Client: @\x0\xDHBS 99W9 DEHBS EIL 11##1111\x1
Client: @\x0#DEHBS P 99##9999DEHBS EIL 11##1111
Server: @\x0\x1DEHBS EIL 11##1111HBS 99W9 \x2
Server: @\x0\x1DEHBS EIL 11##1111HBS 99W9 \x1
Client: @\x0\xBHBS 99W9 DEHBS EIL 11##1111\x3\x3
Client: @\x0\xBHBS 99W9 DEHBS EIL 11##1111\x3\x1
Client: @\x0\xBHBS 99W9 DEHBS EIL 11##1111\x1\x1
Client: @\x0\xBHBS 99W9 DEHBS EIL 11##1111\x1\x3
```

Figure 78: SCI-P telegram exchanges to move point from left position to right position viewed from TC

Timestamp	Criticality	System Part	Instance	Category	Description
09:57:23:360	Info	Test Component	0	Status	Test component finished
09:57:20:310	Info	IXL - Controller	0	Status	SCP connection closed
09:52:03:203	Info	IXL - Controller	0	Data <-	Sent "Point Position(overallpos=RIGHT,degradedpos=NOT DEGRADED)"
09:52:02:845	Info	Controller - PM	4	Data ->	Received "PM Position(RIGHT)" input change
09:51:57:245	Info	Point Controller	0	Event	Processing reception of "Cd_Move_Point(Right)" in machine "F_SCI_P_Receive"
09:51:57:232	Info	IXL - Controller	0	Data ->	Received "Move Point(RIGHT)" telegram
09:51:50:234	Info	Point Controller	0	Data <-	Sent "Initialisation Completed" telegram
09:51:50:224	Info	Point Controller	0	Data <-	Sent "Ability To Move Point(ability=ABLE)" telegram
09:51:50:213	Info	Point Controller	0	Data <-	Sent "Point Position(overallpos=LEFT,degradedpos=NOT DEGRADED)"
09:51:50:202	Info	Point Controller	0	Data <-	Sent "Start Initialisation" telegram
09:51:50:148	Info	IXL - Controller	0	Data ->	Received "Initialisation Request" telegram
09:51:49:184	Info	IXL - Controller	0	Data <-	Sent "PDI Version Check(result=MATCH,version=4,checksumlen=1,checksum=4F)" telegram

Figure 79: Log excerpt of test run (TC_Eu.P.6722.01)



Parameter	Value
D2in_Con_tmax_PDI_Connection	20
D39in_Con_Last_PDI_Version	D39in_Con_Last_PDI_Version
D3in_Con_PDI_Version	4
D4in_Con_Checksum_Data	79
T47in_Con_Other_PDI_Version_Available	T47in_Con_Other_PDI_Version_Available
T10in_SCP_Connection_Terminated	T10in_SCP_Connection_Terminated
T5in_SCP_Connection_Established	T5in_SCP_Connection_Established
T20in_Protocol_Error	T20in_Protocol_Error
T21in_Formal_Telegram_Error	T21in_Formal_Telegram_Error
T22in_Content_Telegram_Error	T22in_Content_Telegram_Error
T44in_Initiate_Maintenance	T44in_Initiate_Maintenance
T45in_Reset_Severe_Error	T45in_Reset_Severe_Error
T48in_Disable_Or_Disconnect_PDI_EfeS	T48in_Disable_Or_Disconnect_PDI_EfeS
T49in_Enable_Or_Connect_PDI_EfeS	T49in_Enable_Or_Connect_PDI_EfeS
d50out_PDI_Connection_State	ESTABLISHED
Move Left:	LEFT
Move Right:	RIGHT
Point Position:	RIGHT, NOT_APPLICABLE
Ability to move point:	ABLE_TO_MOVE
Point Movement failed:	☐

Figure 80: HTB with point position RIGHT, NOT_APPLICABLE after commanding TC to move point to RIGHT position (NOT APPLICABLE: Degraded point position not applicable)

9 RESULTS

This chapter is an amalgamation of results obtained from the test case execution in the chapter 8, development of the HTB and TC in chapter 4, 5 and 6 respectively and the integration of the HTB and the TC in chapter 7.

9.1 RESULTS – WHITE BOX TESTBED VALIDATION APPROACH

The test cases generated from the model-based requirements in the EULYNX Baseline 3 subsystem point requirements specification document for the EULYNX SysML model of the HTB were successfully executed. The validation approach using the HTB has been explained for a EULYNX SysML model of a railway signalling subsystem in the context of the case study i.e. EULYNX Subsystem Point.

The two aspects of the white box validation approach described for the development of the HTB in chapter 4 with the implementation and analysis of the case study could enable the validation approach to be used to scale the HTB to include other railway signalling subsystems in the EULYNX system such as Train Detection System, Level Crossing etc. upon solving the limitations identified.

A EULYNX-based compliant prototype (see Figure 81) was developed and integrated into the HTB to test it against the EULYNX-based requirements specification SysML model to validate in the context of the case study i.e., the subsystem point and its related interface (F_SCI-P).

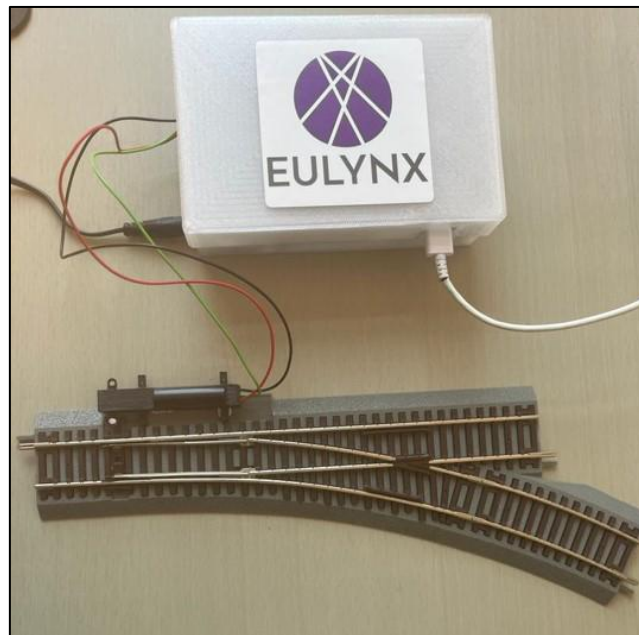


Figure 81: EULYNX-based hardware prototype for Subsystem Point

The HTB enables real-time hardware-based testing of EULYNX specifications with the bidirectional communication between the components of the testbed which could be tested using test cases and data flow could be checked. The functioning white box HTB was used as an input to the expanded and comprehensive black box HTB.

9.2 RESULTS – BLACK BOX TESTBED VALIDATION APPROACH

The HTB developed under the black box approach validated the external functional behaviour of the SUT without internal visibility into the code implementation of the SysML model. The focus of the validation was on testing the responses received by the HTB on various inputs to the SUT and observing the output behaviour of the SUT against the requirements and test cases specified by EULYNX.

The HTB operates as a self-contained testing environment integrating the SysML model, code implementation of the model and the GUI. The GUI effectively represents the subsystem's operational states, enabling user interactions such as sending command to move the point and receiving message about the point position. Through black box testing, it was confirmed that the HTB exhibited external behaviour consistent with the defined EULYNX SCI-P interface specifications.

Communication exchanges such as telegram encoding and decoding according to the ISO/IEC 8859-1:1998 were implemented. The telegram exchanges were checked for accuracy and completeness, ensuring compliance with the EULYNX interface requirements. The black box validation approach demonstrated that the HTB reflected the intended external behaviour at the SCI-P interface and observed the external behaviour of the SUT in relation with Subsystem Point. This validates the role of the HTB as a controlled platform for X-in-the-Loop testing.

9.3 RESULTS – POINT TEST COMPONENT VALIDATION AT RAILSITE LAB

The testing process revealed some issues which were removed successively until all test cases (see Section 8.3) could be executed successfully. Apart from some problems with lab configuration and test case coding, the following issues were discovered in the first version of the TC:

- a copy-and-paste error;
- a forgotten line of code;
- wrong checksum de-/encoding algorithm;
- two wrongly structured conditional (if...then...else) statements (same error).

For a handwritten software with roughly 10.000 lines of code (including empty lines, excluding the RaSTA implementation) these are extremely few errors, of which only the last bullet point refers to statechart code parts. Some explanations for that are that (a) the five test cases only cover the TC code very partially, (b) the model-based specification (SysML blocks and statecharts) could be transferred to code in a very systematic manner, and (c) a couple of GUI errors and of statechart initialization errors (visible in the log) had been identified visually and fixed in advance of testing. Nevertheless, the test results – besides validating that important functions work as expected – give rise to the assumption that the TC code has high quality.

Furthermore, an issue with the EULYNX specifications has been detected: They do not seem to constrain the scheduling of events for statecharts (intentionally) left open by SysML (refer Section 6.2.2), which can lead to presumably undesired sequences of telegrams at the SCI observed during testing.

For the evaluation of the test runs, the recorded log entries were used; a direct observation of the behaviour is sometimes not possible during the test runs due to the quick sequence of events (e.g.

during connection establishment). The interpretation of the logs was feasible, but sometimes a bit arduous because no cause-effect relation between the log entries is recorded and consequently needs to be re-established by hand.

9.4 RESULTS – INTEGRATION OF HYBRID TESTBED AND TEST COMPONENT

The integration of the HTB and the TC was successfully achieved in alignment with the integration architecture. The SCI-P communication over UDP, enabled by a *Tailscale* VPN network, established a remote connection between the HTB (server) in Frankfurt, Germany and the TC (client) in Braunschweig, Germany, demonstrating effective interoperability between the HTB and the TC.

During integration testing, the exchange of SCI telegrams between the HTB and the TC was validated. The telegrams were correctly encoded, transmitted, decoded and received on both ends, conforming with the EULYNX BL4R2 interface specifications.

The HTB demonstrated the capability to interface with external SUTs and perform as an intermediate validation platform bridging model, software and hardware domains. The integration outcome shows that the architecture supports functional interoperability, remote testing and could include other EULYNX subsystems such as train detection systems, level-crossing etc.

9.5 RESULTS – TEST CASE EXECUTION

The execution of test cases formed the final step of the validation approach using the HTB. The embedded test suite supported both automatic and manual test case execution, covering model-based (White Box Approach), functional, interface and fault-injection tests.

Automatic testing based on the test driver was shown. Normal operation test cases successfully executed and passed along with fault-injecting.

Manual testing using the GUI in compliance with EULYNX BL4R2 test cases demonstrated that a operator such as a signaller or tester could effectively control and observe subsystem behaviour. The GUI responded accurately to manual command providing visual confirmation of the SCI-P information flows and subsystem states.

Fault-injection tests were introduced to examine SUT resilience. The SUT on command from the HTB successfully handled invalid telegrams and communication delays.

To summarise, the test case execution showed that the HTB integrated with the TC satisfies the tested behavioural and interface aspects specified by EULYNX and provides a controlled environment for validation and testing.

10 CONCLUSIONS

This chapter concludes and summarises the deliverable. The observations and inferences from the work carried out in Task 30.1.2 of WP 30 to produce Deliverable 30.2 is described here. The value provided by this deliverable, limitations faced and potential for future work are discussed here.

10.1 DEPLOYING A HYBRID TESTBED FOR VALIDATION

The objective of this deliverable is to design, develop and implement a validation means for the EULYNX Subsystem Point in the form of a Hybrid Testbed (HTB), capable of supporting X-in the-Loop testing and system-level validation by integrating SysML models, requirements and interface specifications, IoT technologies and test components.

The white box validation approach verified the internal consistency and correct implementation of the SysML model within the HTB. This validation approach enabled detailed analysis of the block properties, operations and state machine behaviour according to the model-based requirements specifications, ensuring that the C code executed by the hardware prototype realised and accurately reflected the EULYNX BL3R8 Subsystem Point model-based requirements.

The black box validation approach complemented and expanded this by focusing on functional behaviour for the user and system interface perspective. Through this validation approach, the HTB was deployed to validate the functional representation of the subsystem point external behaviour at the SCI-P interface, independent of its internal structure. The SysML state machines transformed into Python code ensured that the realised HTB conformed to the model-based requirements and interface requirements specified by EULYNX in its respective phase 4 and phase 5 of the *CENELEC EN 50126* [9] process.

Finally, the integration between the HTB and TC established the SCI-P over UDP interface demonstrated successful remote bidirectional communication between the interlocking and field element subsystems. This confirms the capability of the HTB to perform validation by providing a controlled, hybrid environment for executing model-based, automatic, manual and fault-injection test cases.

This deliverable demonstrates how SysML-based system representations can be tightly integrated with software and IoT communication infrastructures to enable validation aligned with EULYNX Baseline 4 and *EN50126* lifecycle phases. The work carried out in in this deliverable also illustrates a practical application of MBSE for verifying and validating interoperability of EULYNX subsystems and provides a replicable framework which can be utilised within the scope of Europe's Rail.

Several challenges were encountered during the development of the HTB:

- Integrating various tools and languages into a seamless workflow.
- Managing reliable and deterministic communication over UDP in real-time test conditions.
- Maintaining consistency between SysML models and their respective code implementation.
- Ensuring synchronisation of the model-based behaviour with runtime system responses.

A few open points remain for future investigation:

- Automating model-to-code transformation to minimise manual implementation efforts.
- Adding state machine animations to show coverage of EULYNX model-based requirements.

- Enhancing traceability between the EULYNX SysML model-based requirements and test results for certification alignment.

10.2 HOW TO SUCCESSFULLY CREATE A VERSATILE TEST COMPONENT

The idea of a versatile TC has been successfully realised on the example of the signalling subsystems “point controller” and “point machine”: based on the EULYNX subsystem point specification, a detailed design has been created, implemented and validated. Due to limited resources, some functions have not been implemented completely (failure injection, interface for external point machines), and not all intended use cases could be validated (only use as SUT including manipulation of point machine simulations as reported in Section 6.4). Nevertheless, several conclusions can be drawn from the work already:

- The detailed and clear design of the TC has been an important precondition to arrive at a high-quality software. This included maintaining complete lists of IF parameters, configuration options, GUI controls and manipulation options for not losing the overview; it also included a detailed description of the PM behaviour via state machines and additional text, to cope with each possible configuration and combination of inputs.
- Transforming many state machines into C++ code is possible on the basis of a clear transformation scheme, but laborious. It is estimated that if more than one EULYNX subsystem specification is to be implemented, implementing an automatic transformation first and applying it to exported XMI (XML Metadata Exchange) files of the specification models provided by EULYNX would be more efficient.
- Depending on the particular kind of manipulation, this can be integrated easily into the state machine based code structure (reading/setting information which is stored with or passed between state machines, or manipulating telegrams or events) or requires major local (manipulation of transitions) or global (passing additional delay values together with information) changes.
- The detailed model-based specification leads to surprisingly high ad-hoc quality of implementations as witnessed by the results of basic testing of the TC. However, the tests also showed that it is not trivial to come up with an appropriate solution for parts (event scheduling) which remain unspecified by EULYNX and SysML.

10.3 FUTURE WORK

Future activities could build upon the foundation established by this deliverable by:

- Extending the hybrid testbed to cover other EULYNX subsystems and SCI interfaces such as Train Detection System, Level Crossing, Light Signal, SCI-ILS, SCI-RBC, etc.
- Integrating the Standard Diagnostic Interface (SDI) and the Standard Maintenance Interface (SMI) based the OPC UA protocol into the HTB and TC based on the artefacts released by the ERJU System Pillar in its version 1 release [27]. Also, the alignment of the diagnostic and configuration models with the ERA ontology in the ERJU System Pillar would make the HTB align with the ERA ontology from the diagnostic and maintenance scope.
- Integrating the RaSTA protocol into the communication stack of the HTB to enable the interaction with the TC on the SCI using RaSTA over UDP.

- Including the work and results produced by this deliverable as an input into wave 2 of Europe's Rail FP3 IAM4RAIL project in combination with Deliverable D 14.7 "Specification of Virtual Certification framework for railway infrastructure, specification of use case and demonstration" [30].
- Updating the HTB and the TC to the latest EULYNX BL4R4 and ERJU System Pillar v1 specifications.
- Further developing the HTB to align it with the ERA ontology [41]. With the ERA ontology alignment, the HTB could support automated consistency checks and semantic traceability of test cases, enabling the HTB as an ontology-aware validation platform that contributes to a unified, interoperable and model-driven system architecture across the railway domain.

10.4 FINAL REMARKS

In conclusion, the deliverable demonstrates a significant step forward towards a model-based validation approach using a Hybrid Testbed for EULYNX-compliant products. By applying both the white box and black box validation approaches and accomplishing integration of the Hybrid Testbed with the Test Component, this deliverable proves the feasibility and effectiveness of combining SysML models, IoT communication and software-hardware interaction within a unified testing environment.

This deliverable also highlights the advantages of using standard notations such as SysML for specifications as it reduces ambiguity and increases the potential for comprehension within the ERJU.

This deliverable provides a tangible contribution to Europe's Rail vision of an interoperable, digitalised and modular railway system by setting a strong foundation for the future research, standardisation, development and innovation delivering the integrated Single European Railway Area (SERA) [31].

REFERENCES

- [1] EULYNX specifications, “Trackside Asset Specifications: Baseline X”, https://rail-research.europa.eu/system_pillar/trackside-assets-specifications/.
- [2] EULYNX, “Landing Page”, <https://eulynx.eu/>.
- [3] Salunkhe, S., Berglehner, R., Rasheeq, A. (2021): Automatic Transformation of SysML Model to Event-B Model for Railway CCS Application. In: Raschke, A., Méry, D. (eds) Rigorous State-Based Methods. ABZ 2021. Lecture Notes in Computer Science, vol 12709. Springer, Cham. https://doi.org/10.1007/978-3-030-77543-8_14.
- [4] Mark Bouwman, Djurre van der Wal, Bas Luttik, Mariëlle Stoelinga, and Arend Rensink (2023): A Case in Point: Verification and Testing of a EULYNX Interface. Form. Asp. Comput. 35, 1, Article 2. <https://doi.org/10.1145/3528207>.
- [5] EULYNX (2022): Certification plan. Eu.Proc.7, version 2B from 22/11/2022, published in Baseline 4 Release 1.
- [6] INCOSE, “MBSE Initiative.”, incose.org, <https://www.incose.org/incose-member-resources/working-groups/transformational/mbse-initiative>.
- [7] Deliverable D10.2, “Proposed extension of specification approach to meet needs of RCA, WP 10, X2Rail– 5, 13 January 2022.
- [8] T. Koch, “Rechnergestützter Sicherheitsnachweis: Ein Verfahren zum Ausschluss gefährlicher Systemzustände in rechnergesteuerten Eisenbahn-Sicherungsanlagen”, Dresden: Technische Universität, Fakultät Verkehrswissenschaften “Friedrich List”, Diss. 1997.
- [9] DIN EN 50126-1 VDE 0115-103-1:2025-09, “Railway applications – The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS)”, <https://www.vde-verlag.de/standards/0100884/din-en-50126-1-vde-0115-103-1-2025-09.html>.
- [10] OMG SysML, <https://www.omg.org/sysml/>.
- [11] Jon Holt and Simon Perry, SysML for systems engineering, volume 7. IET, 2008.
- [12] L. Delligatti, “SysML Distilled: A Brief Guide to the Systems Modelling Language”, Reading, MA, USA: Addison-Wesley, 2013.
- [13] Object Management Group. “OMG Unified Modelling Language™ (OMG UML) Version 2.5.” (2015), <https://www.omg.org/spec/UML>.
- [14] ISO 81346-1:2022, “Industrial systems, installations and equipment and industrial products – Structuring principles and reference designations – Part 1: Basic rules”, <https://www.iso.org/standard/82229.html>.
- [15] S. C. Castelino, “Development of a Hybrid IoT Testbed for the Verification and Validation of EULYNX-based SysML Specification Models of a Railway Signalling Subsystem”, B.Sc. thesis, Faculty of Technology and Bionics, Rhine-Waal Univ. of Applied Sciences, Kleve, North Rhine-Westphalia, Germany, 2023.
- [16] ASAL structured action language (SySim), https://support.ptc.com/help/modeler/r10.2/en/index.html#page/Modeler/sysim/SySim_Atego_structured_action_language.html.

- [17] T. Tsokodayi, “Development of a modular Virtual Test bed for validating a SysML Model of a Standard Communication Interface-Point (SCI-P) based on EULYNX Specifications”, B.Sc. thesis, Faculty of Technology and Bionics, Rhine-Waal Univ. of Applied Sciences, Kleve, North Rhine-Westphalia, Germany, 2023.
- [18] Industry IoT Consortium, “Testbeds FAQ”, <https://www.iiconsortium.org/test-beds/test-beds-faq/>.
- [19] Schwencke, D. (2024): A Versatile Test Component Based on the EULYNX Subsystem Point. SIGNAL+DRAHT (116) 09/2024, pp. 67-78. DVV Media, Hamburg.
- [20] DIN VDE V 0831-200 (2015-06): Electric signalling systems for railways - Part 200: Safe transmission protocol according to DIN EN 50159 (VDE 0831-159), <https://www.vde-verlag.de/normen/0800234/din-vde-v-0831-200-vde-v-0831-200-2015-06.html>.
- [21] EULYNX (2023): Requirements specification for subsystem Point. Eu.Doc.36, version 4.3 (0.A) from 28/06/2023, published in Baseline 4 Release 2.
- [22] EULYNX (2023): Generic interface and subsystem requirements. Eu.Doc.20, version 4.0 (3.A) from 27/06/2023, published in Baseline 4 Release 2.
- [23] EULYNX (2023): Generic interface and subsystem requirements for SCI. Eu.Doc.119, version 1.0 (3.A) from 27/06/2023, published in Baseline 4 Release 2.
- [24] EULYNX (2023): Interface specification SCI Generic. Eu.Doc.93, version 3.2 (0.A) from 28/06/2023, published in Baseline 4 Release 2.
- [25] EULYNX (2023): Interface specification SCI-P. Eu.Doc.38, version 4.2 (0.A) from 27/06/2023, published in Baseline 4 Release 2.
- [26] EULYNX (2023): EULYNX System architecture specification. Eu.Doc.16, version 2.2. (0.A) from 27/06/2023, published in Baseline 4 Release 2.
- [27] EULYNX (2022): Scope and coverage justification for EULYNX Certification test cases. Version 1.A from 18/11/2022, published in Baseline 4 Release 1.
- [28] gtkmm Reference Manual Website, <https://gnome.pages.gitlab.gnome.org/gtkmm/>, accessed on 15/05/2024 12:47 a.m.
- [29] System Pillar – Version 1 Release, <https://rail-research.europa.eu/v1-release/>.
- [30] Flagship Project 3 FP3-IAM4RAIL deliverables, <https://rail-research.europa.eu/pages/fp3-iam4rail/deliverables>.
- [31] Europe's Rail, <https://rail-research.europa.eu/>.
- [32] ISO/IEC/IEEE 29119-3:2021 Software and systems engineering - Software testing, <https://www.iso.org/standard/79429.html>.
- [33] International Software Testing Qualification Board, <https://istqb.org/>.
- [34] PyQt, <https://pypi.org/project/PyQt6/>.
- [35] Tailscale, <https://tailscale.com/>.
- [36] PTC, “Overview of SysSim (SysSim).”, support.ptc.com, https://support.ptc.com/help/modeler/r9.6/en/#page/Integrity_Modeler/sysim/SysSim_Overview_of_Artisan_SysSim.html.

- [37] PTC, “Welcome to the Windchill Modeler 9.6 Online Help.”, support.ptc.com, <https://support.ptc.com/help/modeler/r9.6/en/>.
- [38] PTC, “Welcome to the ThingWorx Platform 9 Help Center.”, support.ptc.com. <https://support.ptc.com/help/thingworx/platform/r9/en/index.html#page/ThingWorx/Welcome.html>.
- [39] ISO/IEC 8859-1:1998(en) Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1, <https://www.iso.org/obp/ui/en/#iso:std:iso-iec:8859:-1:ed-1:v1:en>.
- [40] Capella – Open Source MBSE Tool, <https://mbse-capella.org/>.
- [41] ERA Ontology, <https://data-interop.era.europa.eu/era-vocabulary/>.