Rail to Digital Automated up to Autonomous Train Operation

D36.4 – Demonstrator implementation phase 3 concluded, and test results consolidated

Due date of deliverable: 15/02/2026

Actual submission date: 10/02/2026

Leader/Responsible of this Deliverable: SBB

Reviewed: Y

Document status		
Revision	Date	Description
0.1	15/05/2025	Document template skeleton draft for alignment
	06/11/2025	After alignment with all partners: added commitments concerning who is providing what content
0.2	09/01/2026	First WP internal review
0.3	10/02/2026	TMT submission
0.4	07/07/2026	Final document, minor corrections after JU review

Project funded from the European Union's Horizon Europe research and innovation programme		
Dissemination Level		
PU	Public	x
SEN	Sensitive – limited under the conditions of the Grant Agreement	

Start date: 01/12/2022 (WP36 Kick-off 25+26/01/2023) Duration: 42 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
Thomas Martin	SBB	Content for chapters 1, 2, 4 and review
Martin Lindenmaier	SBB	Content for chapter 3 and review
Thomas Buchmüller	SBB	Content for chapters 1, 4 and review
Oliver Mayer-Buschmann	DB	Content for chapters 2, 3, 4 and review
Benjamin Labonté	DB	Review
Ulrich Geier	Kontron	Review
Stefan Resch	Hitachi/GTS	Content chapter 1, 2, 4 and review
Julien SPANNEUT	SNCF Voyageurs	Content chapter 2 and review

EXECUTIVE SUMMARY

The R2DATO WP36 demonstrator addresses a core challenge facing European railway operators and suppliers: how to host safety-critical onboard applications (up to SIL4) together with non-safety services on a modular, vendor-agnostic computing platform to reduce lifecycle costs, accelerate deployment of new features (e.g., ATO, harmonised diagnostics services) maintaining RAM, safety and security. WP36 focused on implementing, integrating and testing an onboard modular computing platform demonstrator in the lab to implement relevant user stories from D36.1 and to produce actionable findings for industrial uptake, further standardisation and future pilots.

Scientific/technical approach and scope

WP36 combined porting of a third-party ERTMS/ETCS SIL4 application, integration of a diagnostics stack and train-integration services, and testing of platform capabilities (replication, recovery, networking, update and security mechanisms) on top of the modular platform, TAS Platform provided by Hitachi. The team executed the work in a controlled lab with a comprehensive simulation environment: offline and online toolchains (Route Map Controller, Train Behaviour Simulator, Balise and Radio simulators) and a hardware DMI. Key activities included:

- Migrating Clearsy's ETCS (Baseline 3, System Version 2.1) to the TAS Platform and running it in a two-out-of-three (2oo3) redundant configuration to demonstrate that using generic fault detection and recovery implemented on platform level is feasible for all kinds of applications.
- Integrating an FRMCS communication stack via the OB_{App} interface such that the ported ETCS application exchanges SUBSET-26 messages with the RBC Message Simulator; the architecture provides a replicated FRMCS communication stack and transparent failover so that if the replica handling FRMCS fails, another healthy replica automatically re-establishes the connection to the RBC simulator.
- Implementing the MCP-DIA diagnostics blueprint, SOVD server and TCMS Data Service components to generate, collect and analyse diagnostics and vehicle data for reproducible testing and automation.
- Testing platform behaviours including restart and reintegration of failing replicas, network fault scenarios, and diagnostics data ingestion and provision to clients.
- Analysing software update workflows and IT/OT security topics (encryption, authentication, IAM and FRMCS transport), highlighting what has been demonstrated and what is theoretical or subject to operator policy.

Main findings and conclusions

- Successful porting and functional operation of SIL4 application: The ported third-party ETCS application ran on the TAS Platform in a 2oo3 replica configuration and executed realistic ETCS control loops in the simulator environment, demonstrating that a third-party safety application can be technically integrated and operated on a modular platform with the platform's fault-management mechanisms.
- Robust FRMCS integration with transparent replica failover: The ETCS application was adapted to use an OB_{App} interface to connect to the FRMCS gateway and exchange

SUBSET-26 messages with the RBC Message Simulator. The replicated FRMCS communication stack provides reliable communication even in times of replica failure and restart. This also demonstrates practical reuse of platform communication functions for radio connectivity and transparent failover of critical communication sessions.

- Diagnostics integration: The MCP-DIA implementation, diagnostics clients (sample clients and the third-party ETCS application) provided reproducible data flows, enabling automated logging and analysis. Diagnostics communication supported cyclic and event paradigms to meet deterministic requirements for safety-critical data while also covering interactive on-demand (request/response) diagnostics for non-safety data. The introduced SOVD [1] standard enables unified, service oriented and extendable access via standard network technology and protocols in a Railway context.
- TCMS integration: Implementation and test of reproducible TCMS diagnostics data flows, fulfilling the demonstration of the TCMS Data Service as proposed in WP 31.
- Platform resilience and recovery demonstrated: Test cases showed automatic detection of failing replicas, continued operation in degraded mode, and re-integration after recovery (manual or automated depending on configuration).
- Communication safety responsibility clarified: The lab work confirmed a crucial architectural insight, that safety protection against message delay, reordering, deletion or replay (EuroRadio SUBSET-37 guidance), if implemented by the modular platform, must be tightly integrated with the supervision of the application.
- Update and security: The deliverable assessed update and security user stories theoretically. The TAS Platform provides update mechanisms and tooling, but operational integration (certificate/key management for FRMCS, IAM, secure update governance and over-the-air FRMCS updates) requires closer alignment with operator infrastructures and standard processes.
- Lab limitations and gaps: Some user stories (heterogeneous hardware diversity, full end-to-end FRMCS security pilots, fleet-scale over-the-air updates and trackside diagnostics exchange) were not fully executed due to lab constraints, and missing time and equipment availability; these remain targets for follow-on work.

Quantified and qualitative impacts

- Demonstrated technical feasibility to host a third-party SIL4 ETCS application with replicated FRMCS communication on a modular platform, indicating tangible potential to reduce onboard hardware duplication and reasonable integration complexity for multi-vendor ecosystems.
- Proven transparent FRMCS failover increases operational robustness of radio-based train-to-track sessions, an important capability for mission-critical ETCS Level 2 operation and future ATO enhancements.
- Diagnostics and TCMS integration establish a blueprint for vehicle data aggregation and condition-based maintenance workflows that can reduce downtime and maintenance costs when industrialized.

Recommendations for future work

- Use standardised application-level safety protocols (e.g., EuroRadio) for communication with external systems: analyse how platform services could assist without assuming safety responsibility.
- Industrial pilots on heterogeneous hardware: execute follow-up trials including different compute modules and vendors to demonstrate portability and heterogeneity for safety/RTE deployments.
- Operational security pilots: move from theoretical security analysis integrated operator pilots combining FRMCS transport, certificate/key lifecycle, IAM, secure update pipelines and rollback procedures for OTA updates.
- Engage standardisation and certification stakeholders: present demonstrator evidence to inform future certification approaches for hosted SIL4 functions, defining needed evidence packages and processes.

Concluding remark

WP36.4 delivers a successful lab demonstration that modular computing platforms can host third-party safety relevant applications up to SIL4 and provide replicated FRMCS communications with transparent failover, while supporting diagnostics and TCMS integration. It has been shown that functions with different SILs can be hosted side-by-side on the same onboard platform. The straightforward porting of a third-party onboard ERTMS/ETCS function to the platform, using the platform's safety and replication mechanisms, demonstrates the practicality of a platform that abstracts the safety layer from the business logic. The work illustrates key points to such platform approaches and presents technical patterns. Addressing these via targeted R&I, operator pilots and standards engagement will accelerate safe industrialisation and wider deployment of modular onboard computing across Europe.

ABBREVIATIONS AND ACRONYMS

Abbreviation	Definition
ATO	Automatic Train Operation
API	Application Programming Interface
BI	Basic Integrity
CAN	Control Area Network
CB	Communication Bridge
CCS	Command-Control and Signalling
CE	Computing Element
CG	Communication Gate
CGMS	Configuration and Group Management Server
CH	Control Handler (incl. FRMCS Agent)
CLIPS	C Language Integrated Production System
CM	Communication Manager
CN	Computing Node (M out of N configuration of CEs)
COTS	Commercial Off-The-Shelf
CSCF	Call State Control Function
DIA-VEC	Vehicle Diagnostics
DTC	Diagnostic Trouble Codes
ECU	Electronic Control Unit
ETCS	European Train Control System
FFFIS	Form Fit Function Interface Specification
FRMCS	Future Railway Mobile Communication System
FRS	Functional Requirements Specification
GRE	Generic Routing Encapsulation
GTW	Gateway
HTTP	Hypertext Transfer Protocol
HSS	Home Subscriber Server
IAM/IdMS	Identity and Access Management / Identity Management System
I/O	Input / Output
IMS	IP Multimedia Subsystem
I-CSCF	Interrogating – Call State Control Function

Abbreviation	Definition
IP	Internet Protocol
JWT	JSON Web Token
LMS	Location Management Server
MDCM	Monitoring, Diagnostics, Configuration, Maintenance
MCP-DIA	Modular Computing Platform Diagnostics
MCx/MCS	Mission Critical Services (MCPTT, MCData, …)
MCx-AS	Mission Critical Application Server
mDNS	Multicast DNS
OAuth	Open Authorization (protocol)
OB	Onboard
OB_{APP}	Onboard Application Interface
OCORA	Open CCS Onboard Reference Architecture
QEMU	Quick Emulator
ORD	Onboard Recording Device
P-CSCF	Proxy – Call State Control Function
RaSTA	Rail Safe Transport Application
REST	Representational State Transfer
RTE	Runtime Environment
SAL	Safe Application Logic
SIP	Session Initiation Protocol
SOVD	Service-Oriented Vehicle Diagnostics
S-CSCF	Serving – Call State Control Function
SDL	SOVD Definition Language"
SRS	System Requirements Specification
SW	Software
TAS Platform	Transportation Automation Systems Platform
TCMS	Train Control and Management System
TE	Technical Enabler
TIU	Train Interface Unit
TLS	Transport Layer Security
TRL	Technology Readiness Level

Abbreviation	Definition
TS	Trackside
TS_{APP}	Trackside Application Interface
UDP	User Datagram Protocol
VM	Virtual Machine
VRRP	Virtual Router Redundancy Protocol
WP	Work Package
YANG	Yet Another Next Generation (a modular language representing data structures in an XML tree format)
YAML	Yet Another Multicolumn Layout

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This document is Deliverable D36.4, "Demonstrator implementation phase 3 concluded, and test results consolidated," produced by Work Package 36 (WP36) of the Rail to Digital Automated up to Autonomous Train Operation (R2DATO) project funded by the Europe's Rail Joint Undertaking (ERJU) under Grant Agreement no. 101102001. WP36's scope is the architecture, integration, demonstration and evaluation of an on-board modular computing platform demonstrator that hosts a mix of platform native basic integrity and SIL4 capable safety applications using the generic safe API of the modular platform. D36.4 reports the activities and outcomes of the third demonstrator implementation phase: it consolidates test cases and results, documents the demonstrator deployment and the implemented user stories, and summarizes lessons learned for subsequent porting and standardization work.

Context and relation to the ERJU program and R2DATO technical structure

- R2DATO addresses the digitalisation of on-board systems and paves the way to higher Grades of Automation (GoA). By demonstrating modular, supplier-agnostic platform architectures hosting multiple safety and non-safety functions, WP36 contributes directly to the ERJU Flagship efforts around technical enablers for interoperable, upgradable on-board computing platforms by building and testing a realistic demonstrator that integrates safety-relevant applications (e.g., ERTMS/ETCS) and basic integrity functions (e.g., diagnostic, communications) and FRMCS interfaces on top of a modular platform. The demonstrator work in WP36 contributes evidence and technical input to future activities as well as to the ERJU System Pillar standardisation efforts (e.g. CONEMP or TrainCS domains).
- This deliverable specifically documents the phase-3 implementation and validation activities (deployment, porting of an ERTMS/ETCS Application, functional cluster demonstrations such as Train Integration and Diagnostics, security and networking analyses) and therefore supports future TE/WPs by delivering empirical results, tested integration patterns and clarified limitations observed in a laboratory demonstrator environment.

Technology readiness level (TRL)

- The demonstrator work reported here is positioned at the system-prototype in a relevant environment reaching the expected TRL 5/6. The lab demonstrator integrates an ERTMS/ETCS Application and the developed diagnostics stack with the assessed generic modular platform components (the TAS Platform) and networked services in a controlled environment. While several elements are demonstrated end-to-end (deployment, diagnostics data flows, FRMCS loose coupling, secure communications analysis), the work remains at prototype demonstration status requiring further steps (industrialization, field trials, certification work) before operational deployment.

Background and problem statement

- Modern on-board systems face growing demands for functional integration, multi-vendor supply chains, lifecycle upgrades, and increased automation levels. Introducing modular

computing platforms, that can host a mixture of SIL-rated safety applications and basic integrity services with well-defined interfaces, secure and predictable communication, and manageable update/migration workflows, may significantly reduce the development and certification effort needed for modern on-board systems.

- Traditional architectures couple safety-relevant applications tightly to bespoke hardware and monolithic software stacks, which hinders supplier competition, evolvability upgrades, and rapid introduction of new functions (e.g., GoA4 ATO features). A key challenge is to provide modularity and portability while preserving the safety properties (e.g., meeting SIL-4 requirements for ETCS functions) and robust security.

Objectives and aims of this deliverable

- Demonstrate the technical feasibility of a modular on-board computing platform hosting a mix of applications including a ported, fault-tolerant ERTMS/ETCS Application delivered by a third-party supplier (Clearsy ETCS Baseline 3, System Version 2.1, adapted for two-out-of-three redundancy).
- Implement selected user stories and associated test cases that exercise deployment and orchestration features, communication paradigms (flows and communication gates), diagnostics collection and exposure (MDCM / MCP-DIA), FRMCS loose coupling, and networking/security concepts relevant for on-board integration.
- Consolidate results from demonstrator phase 3: report which user stories were implemented and tested in the lab, which were omitted (and why), and provide concrete test outcomes, annotations and observed limitations.
- Provide practical insights on porting of existing SIL-rated applications to the modular platform and capture lessons learnt for future work (to be reported in D36.5).

Summary of approach and methods

- Porting and integration: As described in chapter 2.2, a third-party ERTMS/ETCS Application was ported onto the modular platform and adapted to run in a 2oo3 fault-tolerant configuration using the platform's native fault detection and recovery mechanisms. This porting exercise was used to assess technical feasibility and identify integration challenges.
- Modular functional clusters: The demonstrator implements functional clusters such as Diagnostics and Train Integration. For Train Integration, components (Data Player, TCMS Data Service, DIA-VEC brakes, and clients) were deployed either on a virtual diagnostic VM or the RTE and interconnected via dedicated onboard VLANs (OB_BULK and OB_ADMIN) as detailed in chapter 2.5 and 3. The Diagnostics cluster uses MCP-DIA and standardized interfaces (HTTP/REST and TCP socket APIs) for data collection and distribution.
- Test cases and user stories: Chapter 3 provides an inventory and the outcomes of user stories. Some are implemented and tested (e.g., restart of failing replicas, certain communication independence cases, diagnostics collection) while others were omitted or limited by lab constraints (e.g., diverse hardware swaps, platform/RTE portability where only proprietary RTEs were available).

- Security and networking analysis: Chapter 2.3 provides IT/OT security analysis (authentication/authorization, IAM changes, encryption, separation of network zones, monitoring network health). Several user stories relating to security are addressed theoretically in this deliverable.
- Simulation and scenario testing: A comprehensive test suite (Route Map Controller, Scenario Controller/Recorder, Train Behaviour Simulator, Balise and Radio simulators, and a hardware DMI) was used to drive ETCS interactions and verify behaviour under realistic message exchanges (described in chapter 2 and supporting sections).

Novel contributions and positioning against prior work

- Porting of a third-party ERTMS/ETCS Application into a modular, virtualized platform and its operation in a fault-tolerant configuration is a concrete, demonstrator-level contribution. This practical porting exercise provides actionable evidence about what integration points are feasible and which aspects require close cooperation between SIL-4 application developers and platform architects (e.g., handling message semantics that are the application's responsibility, per Subset-037).
- The report contributes a set of user stories (and annotated omissions) that form an evidence base for future standardization and TRL advancement. For example, the implementation of diagnostics data flows via MCP-DIA and the Train Integration TCMS Data Service demonstrates practical APIs and deployment patterns that future developments can reuse.
- Several security-related user stories are treated with theoretical analysis and architectural recommendations (e.g., use of TLS and key management schemes along the lines of Subset-146) that guide further prototyping and field validation.

Structure of the document

- Chapter 2 describes the methods, the porting process of a third-party safe ERTMS/ETCS Application, software update considerations, IT/OT security analysis, and details of the demonstrator deployments and testbeds (network VLANs, virtual machines, RTE instances).
- Chapter 3 contains the user stories, test cases and concrete results. Each implemented user story includes a description, the reasoning or annotation (e.g., omitted or theoretical), and the test evidence or conclusion. This chapter also details functional cluster demonstrations such as Train Integration (components, data flows and deployment) and Diagnostics.
- Chapter 4 summarizes conclusions and identifies open problems and next steps (to be continued in deliverable D36.5), while chapter 6 provides appendices and supporting materials (list of figures/tables, contributors, reference tables).

2 IMPLEMENTED AND STUDIED FUNCTIONS

2.1 FUNCTIONAL CLUSTER DIAGNOSTICS

2.1.1 MCP-DIA

This chapter concludes with the MCP-DIA software design, resulting implementation and final integration activities. For details on Diagnostics User Stories please refer to the document D36.1 "User Stories & Test Cases" [2] chapter 2.5. Evidence on test results of realised User Stories are provided later in chapter 3 next to details on testing, further explanations of results and reasons why three related User Stories have been skipped.

Another architecture update on MCP-DIA is not necessary in this deliverable. All concepts, workflows, functionality, tools and data examples that have been introduced and described in detail within the chapters 2.2 and 3.2 of the former deliverable D36.3 [3] are still valid.

In this deliverable we are setting the focus on those features that have been proposed conceptionally before but haven't been realized yet in the initial MCP-DIA "A-Prototype" software delivery. We also describe further details of the final "C-Sample" MCP-DIA reference implementation integrated on top of the TAS Platform.

The integration of this full-featured solution finally marks the closing results and achievements, addressing open points from the previous deliverables and providing a blueprint implementation, that is driven by automated DevOps deployment and testing from a cloud environment into the laboratory. Chapter 2.2.1.2 finally describes how this diagnostics blueprint has been adapted and integrated alongside the ETCS-onboard application and simulations into the full demonstrator setup of the SBB laboratory.

2.1.1.1 New Features of the final MCP-DIA "C-Sample" Software Delivery

The designs of the following features have already been introduced in the former deliverable D36.3 [3] chapter 2.2. The final implementation now includes these new features for the SOVD-Server and its toolchain, plus a set of TAS Platform applications that have been integrated as a final reference implementation called "Diagnostics Blueprint", using Periodic Sending and Request Response communication patterns supporting Model-1 and Model-3 Tasksets to utilise MCP-DIA.

- SOVD-Server with new features
  - Parallel support of two communication paradigms (two separate server TCP ports)
    - Periodic communication of Event data (unidirectional from Applications to the Server)
    - Request / Response communication between the Server and Applications
  - Data Cache for Periodic Event data reception and storage
  - Syslog-ng utilisation for logging from explicit processes on local Computing Elements
- Integration of a Model-1 Taskset (safe application "etcs_dia")
  - Model-1 Taskset etcs_dia is using periodic (Event) data communication, the sending period of the application is configurable
- Integration of a Model-3 Taskset ("dia_bridge"; in D36.3 [3] chapter 2.2.2.5 this component has been conceptionally introduced as "Basic Integrity Application Event Dispatcher")

- dia_bridge is supporting both, Request/Response and Periodic communication in parallel
- Its main functionality is to create two TCP socket connections to the SOVD-Server relaying sent and received data to POSIX message queues towards TAS Platform applications

- Integration of the “system_health_status_app”
  - The functionality did not change but the application is now using the dia_bridge and its message queues for communication to the SOVD-Server
  - Configurable, supporting both mentioned communication patterns (for testing purpose)
- Multi-platform Update of the Authoring Tool
  - SOVD-Definition-Authoring Tool (VS Code SDL plugin) to declare diagnostic data structures for the Safe and Basic Integrity Applications (uses Protobuf for transport that gets as well integrated into the applications “system_health_status_app” and “etcs_dia” to construct and serialise data that gets sent to the SOVD-Server)

2.1.1.2 Configuration

The SOVD-Server distinguishes diagnostics *Event* data in *ReadValues* and *Faults*.

In the former implementation of the MCP-DIA (A-Prototype) as described in D36.3 [3] chapter 3.2.1, the SystemHealthStatusApp did solely support the Request/Response communication pattern. In the final implementation, this Model-3 Taskset now supports Periodic sending as well. The applied pattern can be selected in the corresponding configuration file of the application:

```
8   PROCESSING_MODE = PERIODIC_SENDIND
9   #PROCESSING_MODE = REQUEST_RESPONSE
10
11  # Time period for sending ReadValue Events to the SOVD server
12  READ_VALUE_EVENT_TIME_PERIOD_SECONDS = 5
13
14  # Time period for sending Fault Events to the SOVD server
15  FAULT_EVENT_TIME_PERIOD_SECONDS = 5
```

Listing 1 Configuration of used Communication Paradigm

In case PERIODIC_SENDING gets chosen, sending intervals must be configured for both, ReadValues and Faults, and the Taskset automatically starts the periodic sending of data to the SOVD-Server after provisioning, without any further needed external interaction. The latest transferred value gets stored for each Event in the data cache of the server.

Data payloads of diagnostics Events stay the same, no matter which of both communication patterns is used. Differentiation towards the SOVD-Server happens solely through varying provisioning and the usage of the adequate of two separate TCP-Ports provided by the SOVD-Server, that can be configured.

```
21  [proto_tcp_config]
22  proto_tcp_server_url = "127.0.0.1"      #ip address for tcp connection with the basic_integrity_app
23  proto_tcp_server_port = 28385          #port for request/response based communication
24  proto_tcp_server_event_port = 28384    #port for event based communication
25
```

Listing 2 TCP Server Configuration

For a client receiving diagnostics data, the used communication pattern (between the application and the server) is fully transparent. In the case of the configuration for PERIODIC_SENDING, the

latest data gets queried from the signal cache. On a REQUEST_RESPONSE configuration, the server request the data directly from the application and forwards the received response to the client.

2.1.1.3 Testing in Azure DevOps

Automated testing has already been described in D36.3 [3] chapter 3.2.2.1 “Automated Build, Integration and Testing”. The used pipelines have been extended to build, integrate and test the software of the final Diagnostics Blueprint reference implementation in the laboratory with the ability to change configurations, test both communication paradigms, and to acquire data from an external client. Test results are evaluated using regular expressions. For more details on the tests please refer to chapter 3.

In the Azure pipeline the appropriate communication configuration gets chosen as a parameter for a test run:

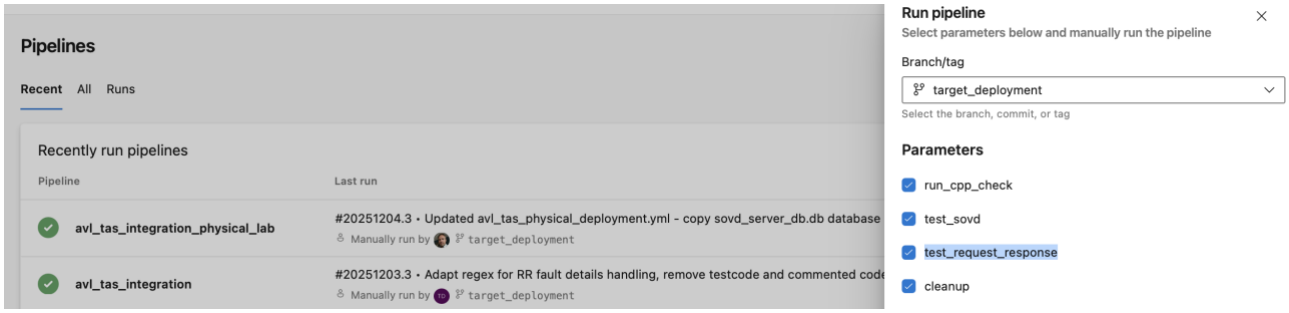


Figure 1 Azure Communication Selection

The received responses are evaluated then by regular expressions to derive reproducible *failed* or *passed* test result that are stored in text files. All test files and results can be downloaded from the artifacts view for later offline analysis, if needed. This procedure also enables the deeper analysis of created log files.

The following script snipped of the pipeline YAML demonstrates how we acquire data, write it to files and evaluate the data with regular expression to derive the test results.

```
curl -s --connect-timeout 10 http://10.36.130.20:7690/v1/apps/SystemHealthStatusApp/data/sysinfo > ~/erju/mcp-dia/logs/test_sysinfo_response
echo "sysinfo response:"
cat ~/erju/mcp-dia/logs/test_sysinfo_response && \
grep -E -q '{"id":"sysinfo","data":{"KernelVersion":"[^"]+","OSVersion":"[^"]+","HostName":"tas-plf-container"}}' ~/erju/mcp-dia/logs/test_sysinfo_response
curl -s --connect-timeout 10 http://10.36.130.20:7690/v1/apps/Model1Taskset/faults > ~/erju/mcp-dia/logs/test_faults_response
echo "faults response from Model1Taskset:"
cat ~/erju/mcp-dia/logs/test_faults_response && \
grep -E -q '\{"items":\[\{"code":"P0138","scope":"","display_code":"P0138","fault_name":"ATO Brakes Ctrl","fault_translation_id":"translation_P0138","severity":1,"status":\{"aggregated_status":"NotSet","te
curl -s --connect-timeout 10 http://10.36.130.20:7690/v1/apps/Model1Taskset/faults/P0138 > ~/erju/mcp-dia/logs/test_fault_details_response
echo "fault details response from Model1Taskset:"
cat ~/erju/mcp-dia/logs/test_fault_details_response && \
grep -E -q '{"item":{"code":"P0138","scope":"","display_code":"P0138","fault_name":"ATO Brakes Ctrl","fault_translation_id":"translation_P0138","severity":1,"status":{"aggregated_status":"NotSet","testFail
curl -s --connect-timeout 10 http://10.36.130.20:7690/v1/apps/SystemHealthStatusApp/faults > ~/erju/mcp-dia/logs/test_system_health_status_app_faults_response
echo "faults response from SystemHealthStatusApp:"
cat ~/erju/mcp-dia/logs/test_system_health_status_app_faults_response && \
grep -E -q '\{"items":\[.*\{"code":"345435","scope":"","display_code":"345435","fault_name":"cpu_load_high","fault_translation_id":"translation_345435","severity":1,"status":\{"aggregated_status":"Active",
curl -s --connect-timeout 10 http://10.36.130.20:7690/v1/apps/SystemHealthStatusApp/faults/345436 > ~/erju/mcp-dia/logs/test_system_health_status_app_fault_details_response
echo "fault details response from SystemHealthStatusApp:"
cat ~/erju/mcp-dia/logs/test_system_health_status_app_fault_details_response && \
grep -E -q '{"item":{"code":"345436","scope":"","display_code":"345436","fault_name":"free_memory_percent_low","fault_translation_id":"translation_345436","severity":1,"status":{"aggregated_status":"Activ
curl -s --connect-timeout 10 http://10.36.130.20:7690/v1/apps/Model1Taskset/data > ~/erju/mcp-dia/logs/test_data_response
echo "data response:"
cat ~/erju/mcp-dia/logs/test_data_response && \
grep -E -q "\{\"items\":\[\{\"id\":\"ato_brake_temperature\",\"name\":\"ATO Brake Temperature\",\"category\":\"currentData\"\}\]\}" ~/erju/mcp-dia/logs/test_data_response
curl -s --connect-timeout 10 http://10.36.130.20:7690/v1/apps/Model1Taskset/data/ato_brake_temperature > ~/erju/mcp-dia/logs/test_read_value_response
echo "read value response:"
cat ~/erju/mcp-dia/logs/test_read_value_response && \
grep -E -q '{"id":"ato_brake_temperature","data":{"ATO Brake Temperature":[0-9]{1,3}}}' ~/erju/mcp-dia/logs/test_read_value_response
curl -s --connect-timeout 10 http://10.36.130.20:7690/v1/apps/SystemHealthStatusApp/logs/entries > ~/erju/mcp-dia/logs/test_system_health_status_app_log_entries_response
echo "log entries response from SystemHealthStatusApp:"
cat ~/erju/mcp-dia/logs/test_system_health_status_app_log_entries_response && \
grep -E -q '{"items":\[.+\]}' ~/erju/mcp-dia/logs/test_system_health_status_app_log_entries_response
curl -s --connect-timeout 10 http://10.36.130.20:7690/v1/apps/Model1Taskset/logs/entries > ~/erju/mcp-dia/logs/test_etcs_dia_log_entries_response
echo "log entries response from Model1Taskset:"
cat ~/erju/mcp-dia/logs/test_etcs_dia_log_entries_response && \
grep -E -q '{"items":\[.+\]}' ~/erju/mcp-dia/logs/test_etcs_dia_log_entries_response
```

Listing 3 YAML Pipeline Scripting Example

The following figure shows a positive test result of a pipeline run in Azure.

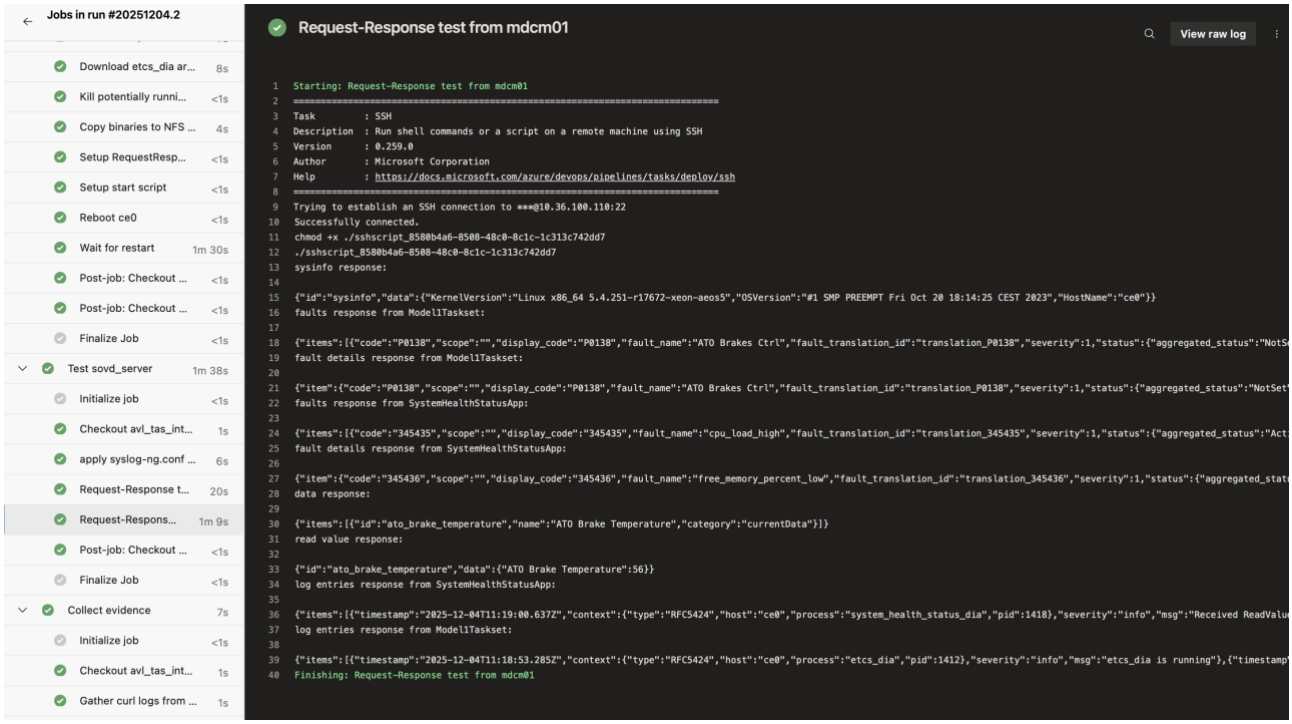


Figure 2 Request/Response Test executed from a Lab Server in Azure

2.1.1.4 Provisioning for Periodic communication

The Provisioning-Message is created by the Provisioning Event-Factory of an application as described below in Listing 4 Code Example for Provisioning (of the *system_health_status_app)*.

The SOVD Client transfers the app definition file path to the SOVD Server. The SOVD Server loads and reads the app definition binary-file from the given path.

- The `ProvisioningEvent` contains all the information about the app entity like entity type, entity ID, `LogContext` and the path to the entity definition file.
- To enable the logging functionality in the SOVD Server it is necessary to add at least one `LogContext` in the `ProvisioningEvent`.
- The payload of the `LibSovdClientEvent` is the `ProvisioningEvent`.
- The value of the `LibSovdClientMessage` is the `LibSovdClientEvent`.

```
// Provisioning for Periodic communication
sovd::LibSovdClientMessage create_provisioning_event_message() {
   sovd::LibSovdClientMessage client_message;
   auto *event = new sovd::LibSovdClientEvent();
   auto *provisioning = new sovd::ProvisioningEvent();

   provisioning->set_type(sovd::app); // Set Entity-Type
   provisioning->set_entity_id(ENTITY_ID); // Set Entity-ID

   // Setup Logging
   auto *rfc = new sovd::RFC5424();
   char hostname[256];
   if (gethostname(hostname, sizeof(hostname)) == 0) {
      rfc->set_host(hostname);
   } else {
```

```
      rfc->set_host("tas-plf-container");
   }
   rfc->set_process("system_health_status_app");
   const auto pid = getpid();
   rfc->set_pid(pid);
   const auto log_context = new sovd::LogContext();
   log_context->set_allocated_rfc5424(rfc);
   provisioning->add_log_contexts()->CopyFrom(*log_context);

   // Read entity definition file path from config and set in ProvisioningEvent
   const auto it = config.find("PROVISIONING_EVENT_ENTITY_DEFINITION_FILE");
   if (it == config.end()) {
      syslog(LOG_ERR, "Missing required config: PROVISIONING_EVENT_ENTITY_DEFINITION_FILE");
      exit(EXIT_FAILURE);
   }
   const std::string &entity_file_path = it->second;
   provisioning->set_entity_definition_file(entity_file_path);

   // LibSovdClientEvent::payload = ProvisioningEvent
   event->set_allocated_provisioning(provisioning);

   // LibSovdClientMessage::value = LibSovdClientEvent
   client_message.set_allocated_event(event);

   return client_message;
}
```

Listing 4 Code Example for Provisioning of Periodic communication

To ensure that the `ProvisioningEvent` correctly reflects the app's identity and metadata, the config-file *conf/app_xyz.conf* needs to be adjusted.

A specific config parameter `PROVISIONING_EVENT_ENTITY_DEFINITION_FILE` needs to specify the absolute path location of your compiled *.bin*-file in the TAS Platform container´s or CE´s filesystem.

For proper provisioning, it must be ensured that the *.bin*-file contains the latest SDL-compiled definition and corresponds to the same app entity ID registered with the SOVD Gateway Server.

2.1.1.5 Provisioning for Request / Response communication

The SOVD Client loads and reads the app definition binary-file from the entity definition file path and uses the application as value for the `LibSovdClientMessage`. So, in this case the SOVD Server gets the App information from the SOVD Client and therefore does not have to know about the entity definition file path.

Step 1: Provisioning-Message for Request / Response communication:

```
// Load the App from the definition file path
sovd::App app = get_app_from_definition_file(config["LOCAL_ENTITY_DEFINITION_FILE"]);

// Enable logging support for this App
app -> set_has_logs(true);

// Provision the Server with the App from the definition file with a Provisioning-Message
sovd::LibSovdClientMessage client_message;
client_message.set_allocated_app(app);
sender->send(client_message);
```

Listing 5 Code Example for Provisioning of Request/Response-based communication

Step 2: Additionally enable Logging for Request/Response Entities:

To enable the logging functionality in the SOVD Server, a `ProvisionEvent` especially containing the `LogContext` must be sent in addition to the Provisioning-Message.

- `LogContext`, Entity-Type and Entity-ID are needed in the SOVD-Server.
- Send a `ProvisioningEvent` after the Provisioning-Message.
- This reduced `ProvisioningEvent` only contains information to setup logging in the SOVD-Server.
- Information about the entity definition file path is not needed in this case.

```
// Enable Logging for Request/Response-based communication
sovd::LibSovdClientMessage create_logging_initialization_provisioning_event_message() {
   sovd::LibSovdClientMessage client_message;
   auto *event = new sovd::LibSovdClientEvent();
   auto *provisioning = new sovd::ProvisioningEvent();

   provisioning->set_type(sovd::app); // Set Entity-Type
   provisioning->set_entity_id(ENTITY_ID); // Set Entity-ID

   // Setup Logging
   auto *rfc = new sovd::RFC5424();
   char hostname[256];
   if (gethostname(hostname, sizeof(hostname)) == 0) {
      rfc->set_host(hostname);
   } else {
      rfc->set_host("tas-plf-container");
   }
   rfc->set_process("system_health_status_app ");
   const auto pid = getpid();
   rfc->set_pid(pid);
   const auto log_context = new sovd::LogContext();
   log_context->set_allocated_rfc5424(rfc);
   provisioning->add_log_contexts()->CopyFrom(*log_context);

   // LibSovdClientEvent::payload = ProvisioningEvent
   event->set_allocated_provisioning(provisioning);

   // LibSovdClientMessage::value = LibSovdClientEvent
   client_message.set_allocated_event(event);

   return client_message;
```

Listing 6 Code Example for Provisioning of a Logging Context

2.1.2 Diagnostics Blueprint

As proposed in D36.3 [6] chapter 3.2.2 the diagnostics implementation has been extended by an exemplary Model-1 taskset providing faults and test data. This blueprint reference implementation comprises the full feature set of the final software delivery and serves as a basis for the diagnostics integration into the full demonstrator setup, running an ETCS-onboard application in a safe replicated setup, as explained in chapter 2.2. The diagnostics specific part can be found in chapter 2.2.1.2. Since this blueprint is a progression of the former implementation, we concentrate here on relevant details related to the extended feature-set, its configuration and deployment in the laboratory.

The following figure provides a high-level overview of the components and communication flows. The data flows reflect the features that have been introduced in the previous sections of this chapter. etcs_dia is a Model-1 Taskset, system_health_status_dia and dia_bridge are Model-3 tasksets. The latest version of the sovd_server supports both communication paradigms in parallel and can collect data from syslog in case this has been appropriately provisioned as explained in the prior chapter.

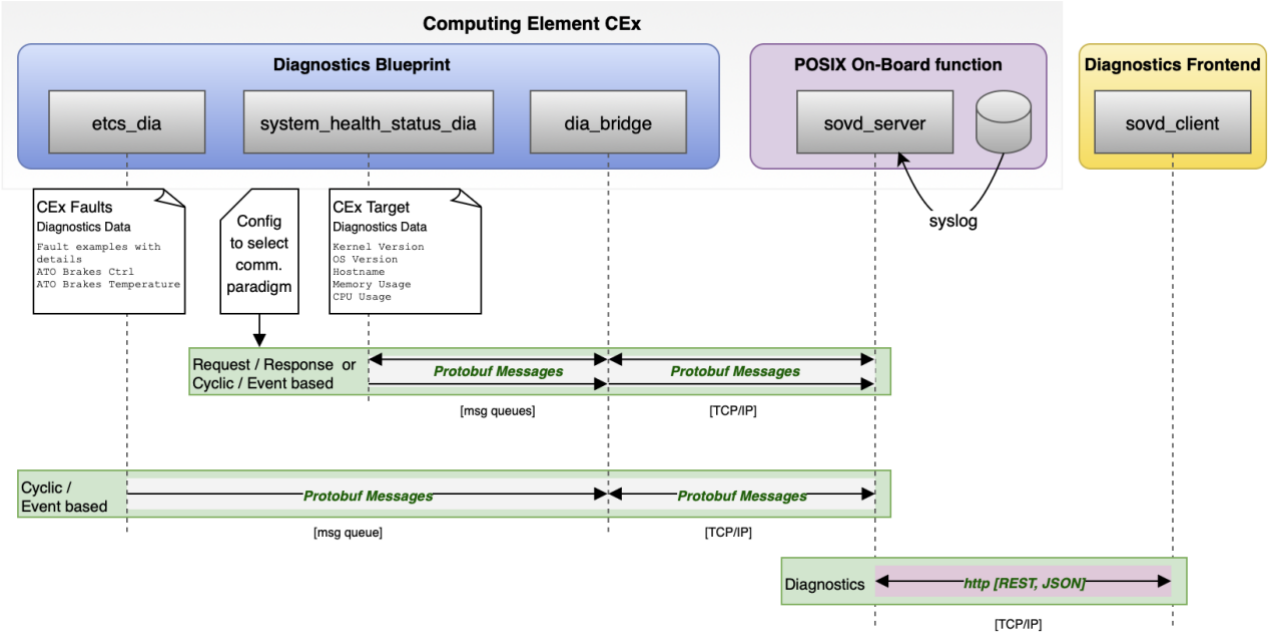


Figure 3 Diagnostics Blueprint and Communication Flows

For syslog access the configuration of syslog-ng has been extended to register the SOVD-Server as a destination for CE local system log output.

```
destination d_sovd_server {
    network("127.0.0.1"
            port(515)
            transport(tcp)
            template("Date=${ISODATE};Host=${HOST};Program=${PROGRAM};PID=${PID};Level=${LEVEL};Message=${MESSAGE}\n"));
};

log {
    source(aeos_msgs);
    destination(messages_program);
    destination(d_sovd_server);
    #destination(atvieud001_udp);
    #destination(atvieud003_udp);
};
```

Listing 7 syslog-ng.conf with registered SOVD-Server

2.1.2.1 Data Declarations for etcs_dia

In the context of the Blueprint, etcs_dia represents a safe application that periodically sends diagnostics data as Events to the SOVD-Server according to its SOVD Definition Language (SDL) configuration.

```
App Model1Taskset {
    description "Model1 Taskset Example"

    Fault ato_brakes_ctrl {
        code "P0138"
        fault-name "ATO Brakes Ctrl"
        severity 1
        supports-standard-environment-data
        environment-data
            timestamp: string description "Timestamp"
    }

    ReadValue ato_brake_temperature {
            description "ATO Brake Temperature"
            data-category currentData
                Temperature: int32 description "ATO Brake Temperature" unit DegreeCelsius
    }
}

/*Degree Celsius*/
Unit DegreeCelsius {
    display-name "°C"
    reference "UNIT0026"
    factor-si-to-unit 1.0
    offset-si-to-unit 273.15
    physical-dimension ThermodynamicTemperature
}

/*Byte*/
Unit Byte {
    display-name "B"
    reference "UNIT0185"
    factor-si-to-unit 0.125
    offset-si-to-unit 0.0
    physical-dimension StorageCapacity
}
```

Listing 8 SDL of the Model-1 Taskset etcs_dia

For the final integration of the ETCS application this demo fault declarations have been replaced with appropriate ETCS Diagnostics data points. The following SDL shows some details of the data

that gets transferred through an additional message queue from the ETCS application to etcs_dia and further on to the SOVD-Server as explained in detail in chapter 2.2.1.2:

```
App EtcsStatusApp {
    description "ETCS application diagnostics"

ReadValue trainInfo {
        description "Train Information"
        data-category currentData
            TrainRunningNumber: string description "Train Running Number"
            ActiveCabin:  string description "Active Cabin"  enum  EActiveCabin
    }

    ReadValue etcsStatus {
        description "Current ETCS status"
        data-category currentData
            Level: string description "ETCS Level" enum ETrainMode
            Mode: string  description "ETCS Mode" enum ELevel
            EBrake: string description "Emergency Brake state" enum EEmergencyBrake
            Speed: double description "Train Speed" unit MeterPerSecond
            EoALocation: double description "End of Authority (EOA) location" unit Meter
            EFrontEndPos: double description "Estimated front position" unit Meter
    }
}

Enum EEmergencyBrake {
    "0" : "Released"
    "1" : "Engaged"
}

Enum ELevel {
    "0" : "Level 0"
    "1" : "Level STM"
    "2" : "Level 1"
    "3" : "Level 2"
    "4" : "Level 3"
    "5" : "Level Unknown"
}

Enum ETrainMode {
    "0" : "FS: Full supervision"
    "1" : "OS: On sight"
    "2" : "SR: Staff responsible"
    "3" : "SH: Shunting"
    "4" : "UN: Unfitted"
    "5" : "SL: Sleeping"
    "6" : "SB: Stand by"
    "7" : "TR: Trip"
    "8" : "PT: Post trip"
    "9" : "SF: System failure"
    "10" : "IS: Isolation"
    "11" : "NL: Non leading"
    "12" : "LS: Limited supervision"
    "13" : "SN: STM national"
    "14" : "RV: Reversing"
    "15" : "PS: Passive shunting"
    "16" : "Unknown"
    "17" : "No Power"
}

Enum EActiveCabin {
    "0" : "No active cabin"
    "1" : "Cabin A activated"
    "2" : "Cabin B activated"
    "3" : "Cabin A and B activated"
}
```

Listing 9 Extension of etcs_dia for the ETCS Application

2.2 PORTING OF AN ERTMS/ETCS ONBOARD APPLICATION TO THE PLATFORM

Having a modular platform with a standardized interface enables multiple suppliers to provide applications that leverage the platform. Suppliers can offer basic integrity applications - such as the diagnostics reference implementation described in the previous chapter - but crucially, they can also supply SIL4 safety-critical applications, for example an ERTMS/ETCS Application.

To demonstrate the technical feasibility and identify challenges of the modular platform approach, Clearsy's ERTMS/ETCS On Board Simulator has been ported to the TAS Platform and is used in the demonstrator as ETCS on-board Application (ETCS Baseline 3, System Version 2.1). The target of the porting was the 2oo3 (two-out-of-three) fault-tolerant configuration. The ported application uses the platform's native fault-detection and recovery mechanisms so that, if one replica fails, the remaining replicas can continue operation and recovery actions can be invoked manually or automatically.

This deliverable describes the technical porting and the execution of corresponding test scenarios to verify all related user stories (see results in chapter 3). The report on the experiences and lessons learnt during the porting process will be reported on in the subsequent deliverable 36.5.

2.2.1 Porting Process

The porting required a reorganisation of the application's potentially safety-relevant and non-safety-relevant functions to run in their respective TAS Platform domains(model-1 and model-3 Tasksets respectively): the EVC core business logic had to be ported to and executed within the TAS Platform provided computation model for safe functions (model-1 Tasksets), while peripheral communication and other non-safety-relevant functions had to be placed in the TAS Platform provided computation model for non-safety-relevant functions, i.e., model-3 Tasksets.

Moving input and output management into the basic integrity domain means that all peripheral communication including operator inputs and other non-safety-relevant sensor/actuator processing are performed outside the safe runtime and hence may use a wider set of POSIX functions, as e.g., sockets for network communication.

The EVC core itself was ported to the TAS Platform without changing its business logic, failsafe behaviours or timing constraints. However, mutexes, watchdogs and fault propagation mechanisms had to be adjusted to comply with the TAS Platform programming model.

The porting of Clearsy's ERTMS/ETCS Application proceeded smoothly. With support from Hitachi and the SBB team, key issues - such as memory management and task structure - were quickly identified and addressed with sound conceptual solutions. It is important to note that Clearsy has safety expertise, but the migrated ETCS application was originally developed without safety-critical requirements and used only in a test environment. Within six months, Clearsy completed the porting of a slightly streamlined version of the ETCS application. This result is encouraging for the planned future modularisation and supports integrating applications of multiple suppliers.

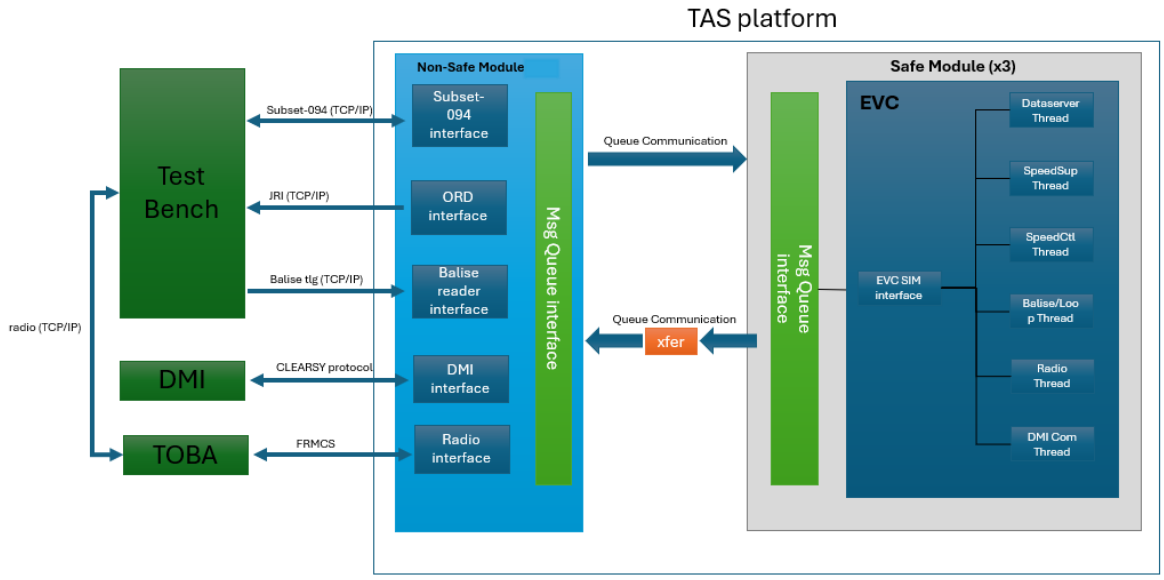


Figure 4 Separation of peripheral communication functions from EVC core business logic

To keep the demonstrator manageable, all onboard peripheral communications (for example, with the balise simulator, the DMI, the ORD, etc.) have been concentrated on a single dedicated computing element (CE). That CE is excluded from automatic restart during recovery tests, so its connections are stable and do not interfere with test scenarios.

This simplification was necessary to complete the porting within the available timeframe and budget. To demonstrate that it can be overcome in a full implementation, the FRMCS link to the trackside RBC simulator is handled by the replicated FRMCS communication stack as specified in deliverable D36.3, chapter 2.1.1 *Communication Architecture*. FRMCS traffic is therefore subject to a transparent failover: if the replica currently handling FRMCS fails, the FRMCS connection is automatically migrated to one of the remaining healthy replicas.

Similarly, any peripheral communication that is, for simplicity, routed through a single dedicated replica could adopt the same communication architecture to enable redundancy and transparent failover to one of the remaining replicas for availability.

2.2.1.1 Integration of FRMCS communication stack

The ported ERTMS/ETCS Application is based on ETCS baseline 3, System Version 2.1 which does not include native FRMCS support. Hence, the current implementation is focusing on establishing an FRMCS connection via the OB_{App} interface and exchanging application level SUBSET26 messages with the RBC Message Simulator. The implementation of the EuroRadio protocol was intentionally excluded. The communication stack defined in deliverable D36.3, chapter 2.1.1 *Communication Architecture* nevertheless acknowledges the need for a safety protocol and explicitly identifies the components that would be responsible for implementing the EuroRadio safety layer.

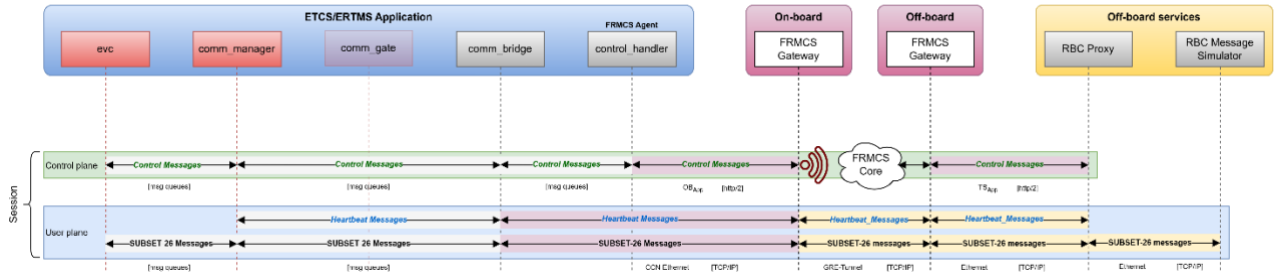


Figure 5 FRMCS communication stack (architecture)

2.2.1.2 Integration of diagnostics stack

The diagnostics stack supports two communication paradigms: cyclic/event-based communication and request–response operations (see also chapter 2.1.1 *MCP-DIA*). Safety-critical applications must use the periodic approach to guarantee deterministic behaviour of the safety function. Although (CEx)-specific non-safe system information (for example, hostname and memory usage) could be implemented using the request–response pattern, we chose to also use the cyclic/event-based approach for simplicity. The diagram below shows the involved entities and the highlevel- diagnostics data flow.

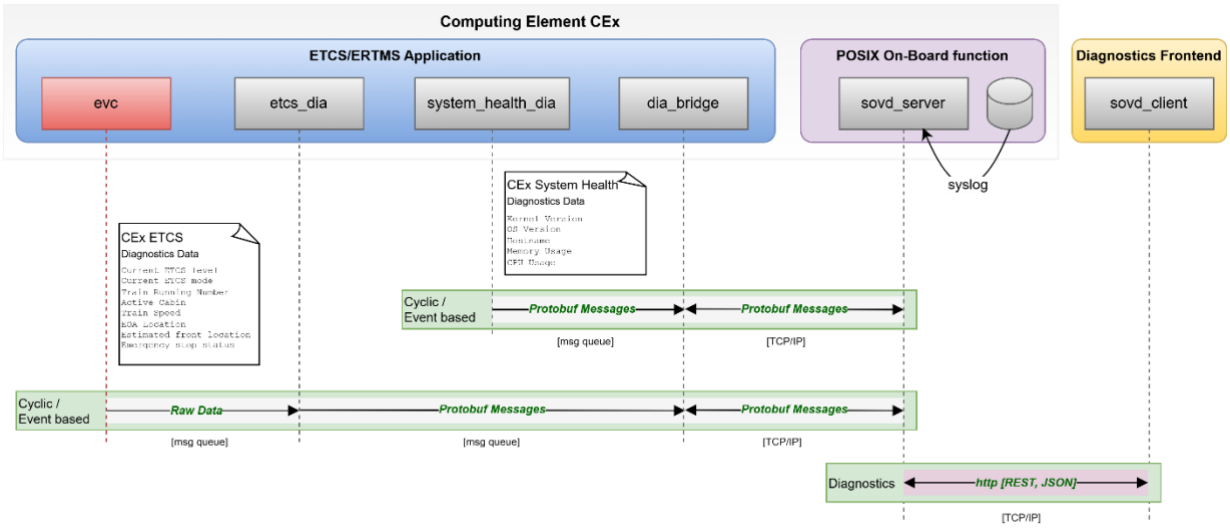


Figure 6 Diagnostics communication stack (cyclic / event-based architecture)

The proposed architecture uses a generic *dia_bridge* taskset that receives messages from a queue and forwards them over TCP/IP to the SOVD server. It is agnostic to message contents and therefore does not use the Google Protobuf library. A single *dia_bridge* handles all cyclic/event-based diagnostics communication for all diagnostics tasksets on a specific CE (in our case the two tasksets *etcs_dia* and *system_health_dia*).

The diagnostics taskset *system_health_dia* is responsible for Computing Element (CEx)-specific system (health) information. It makes use of the Google Protobuf library to construct the diagnostics messages expected by the SOVD server.

The *evc* periodically writes raw diagnostic data to a POSIX message queue consumed by the *etcs_dia* taskset. *etcs_dia* reads this data, constructs the corresponding Protobuf messages, and forwards them - again via a POSIX message queue - to the *dia_bridge* taskset.

Because the *evc* implements the safety-critical logic, it must comply to the TAS Platform programming model for safety-relevant functions. To minimise the certified code base and reduce the potential certification scope, the Google Protobuf library is therefore not linked into the *evc* taskset. Therefore, the *evc* does not create Protobuf messages to send them directly to the *dia_bridge*.

Based on the provisioning data, the sovd_server gathers the evc taskset's syslog entries and exposes them to the sovd_client.

The described architecture applies to each replica; consequently, if required, the diagnostics frontend must aggregate the information collected from each replica .

2.2.2 Simulation Environment

A sophisticated simulation environment provides the infrastructure necessary for the ERTMS/ETCS Application to behave as if it were installed on a real train. All peripherals and trackside elements and services - including balises and the RBC - are simulated. The only physical hardware in the loop is the ETCS driver machine interface (DMI) provided by Clearsy, nevertheless we have also used a DMI simulation to document certain test scenarios.

The following diagram provides an overview of the simulation architecture.

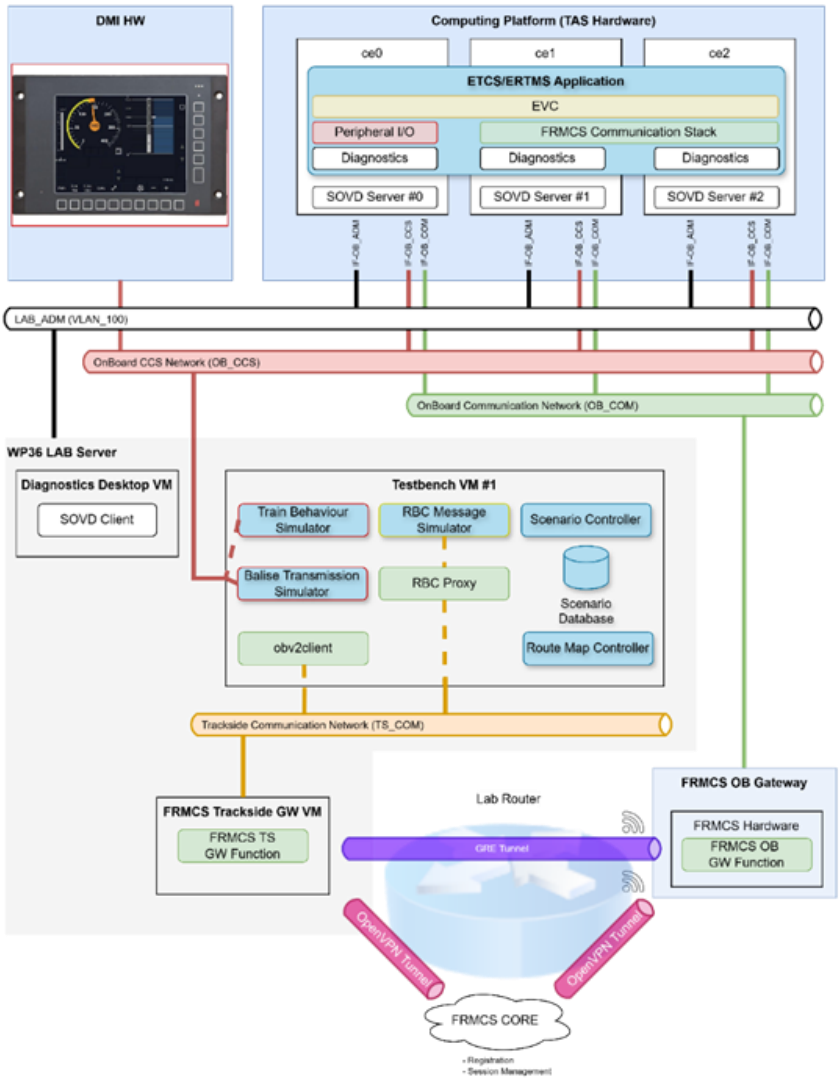


Figure 7 High-level architecture of test environment

The simulation environment comprises both offline and online toolchains. Offline tools are used to design and configure test scenarios for the ported ERTMS/ETCS Application, while online tools execute those scenarios and drive the runtime testbed.

2.2.2.1 Offline toolchain

The offline toolchain is used to create scenarios that reproduce the operational environment of the ERTMS/ETCS Application as if it were installed on a real train.

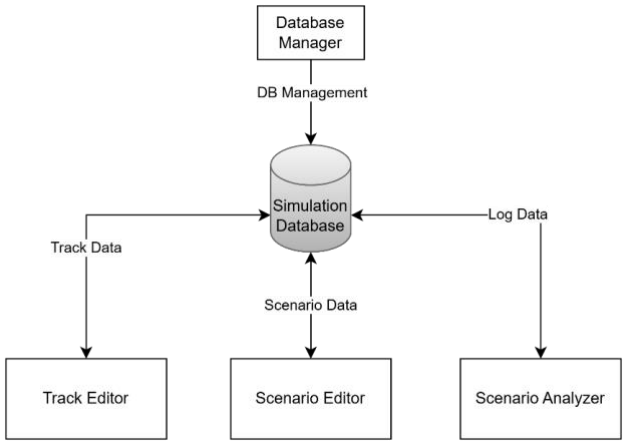


Figure 8 Offline toolchain

The following section gives a brief overview of the above-mentioned offline modules.

- ***Database Manager***: A simple tool for managing scenario databases that store all information required to run a scenario. It lets you create a new empty database and supports copying, deleting, backing up, and restoring databases.
- ***Track Editor:*** A graphical user interface for defining railway topologies used in scenarios. It builds track layouts from segments and nodes and supports the placement of points and crossings as well as position infrastructure elements such as balises, loops, and signals.

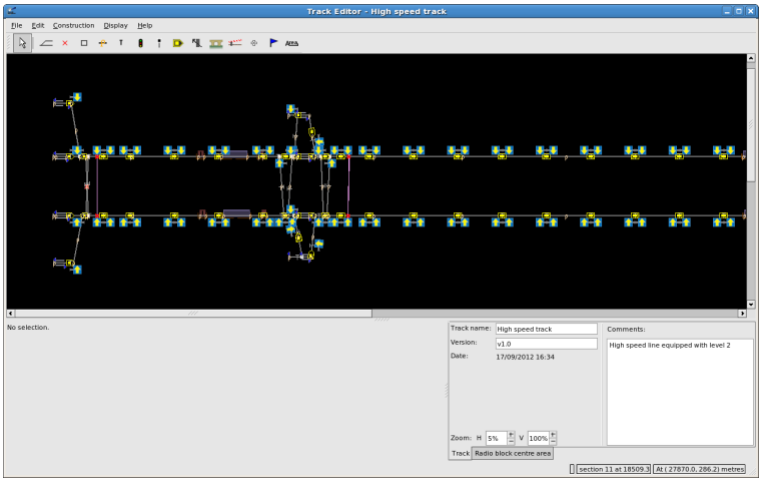


Figure 9 Track Editor

- ***Scenario Editor:*** A tool for defining and managing the track and train parameters and behaviour used as the testbed for the ERTMS/ETCS Application. Used to specify train characteristics and place the train on the track, and can create balise telegrams, radio messages and other event definitions.
- ***Scenario Analyzer:*** During the execution of a testcase based on a defined scenario, log files are generated by the different simulators as well as by the ETCS onboard system. The

Scenario Analyzer is used to visually analyse the log data. The logs can be filtered and visualized in graphs and tables.

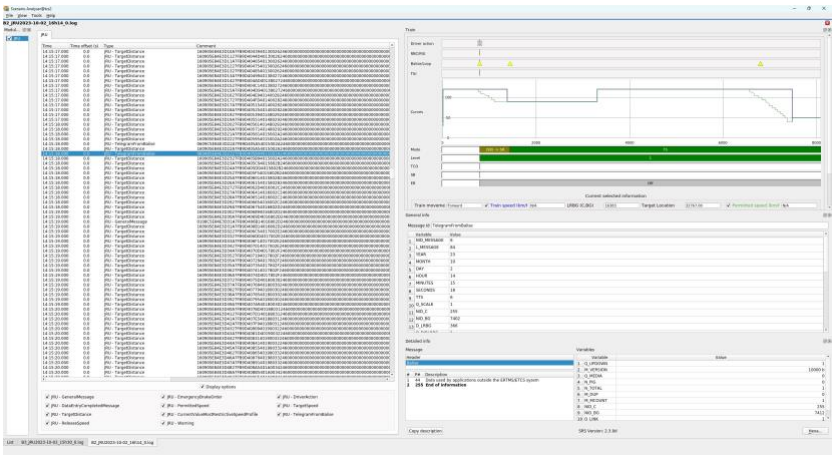


Figure 10 Scenario Analyzer

2.2.2.2 Online toolchain

A dedicated online toolchain runs the defined scenarios: its modules exchange data and stay synchronized via a CORBA software bus, and they expose to the ERTMS/ETCS Application the interfaces defined by the TSI CCS. The online toolchain includes simulations of on-board CCS components, train equipment, trackside elements, and trackside systems. The diagram below gives an overview of the main online modules, their internal communication over the CORBA bus, and the ERTMS/ETCS Application interfacing through the standard TSI CCS interfaces.

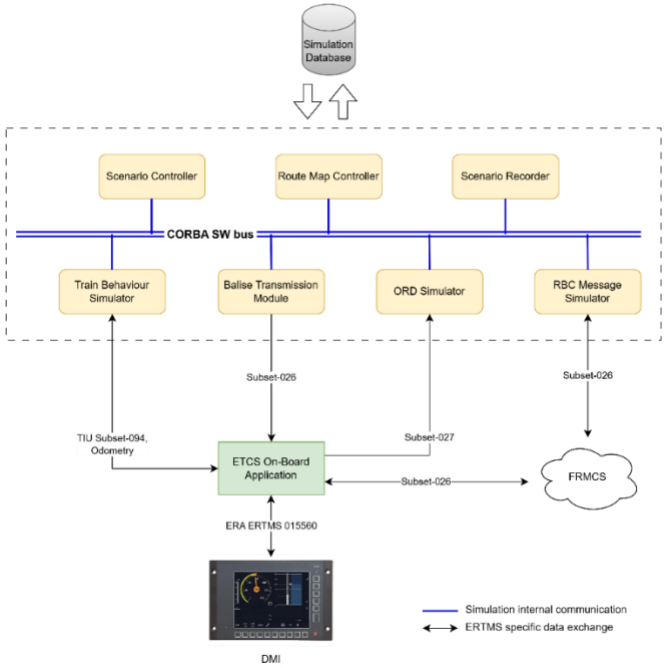


Figure 11 Online toolchain

The following section gives a brief overview of the above-mentioned online modules.

- ***Route Map Controller:*** The Route Map Controller serves as the centralised module responsible for exchanging data with the other online modules. Its functions include displaying the track topology and signalling objects, visualising the train as it runs on the track, and forwarding balise messages to the Balise Transmission Simulator module. In addition, the Route Map Controller transmits the track description to the Train Behaviour Simulator module.

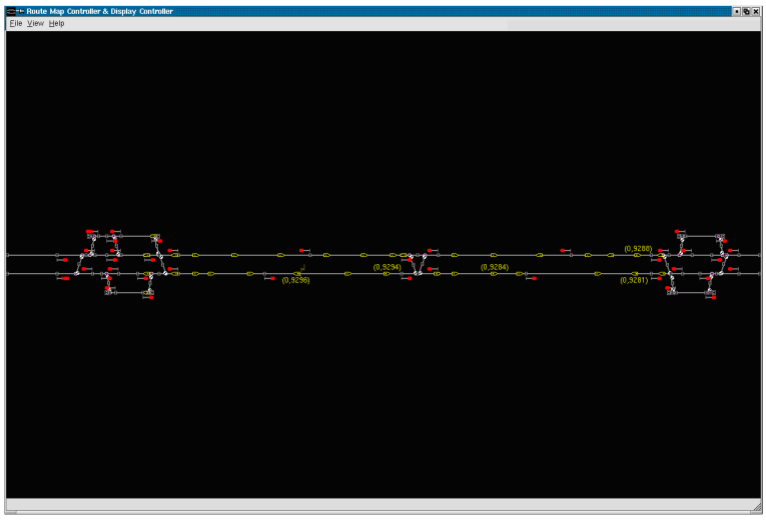


Figure 12 Route Map Controller user interface

- ***Scenario Controller:*** The Scenario Controller is responsible for selecting scenarios from the database and managing their lifecycle. Its existing functions include choosing a scenario, starting, controlling and stopping scenario execution, activating database logging through the Scenario Recorder feature, and enabling the automatic simulation running capabilities.
- ***Scenario Recorder:*** The Scenario Recorder collects log data from all modules and stores it in the database so the Scenario Analyser can retrieve and process it. It is started by the Scenario Controller when a scenario begins. No graphical user interface is provided, as the Recorder is integrated into the Scenario Controller.
- ***Train Behaviour Simulator:*** The Train Behaviour Simulator computes train dynamics from the received track description - including train configuration, gradient and power profiles, static speed profile and related data - and manages message exchanges according to SUBSET-94. Its current functions cover management and visualization of TIU inputs and outputs, management, desk control, and computation of train dynamics based on forces, gradient, adhesion and interventions from the ETCS Test Object.

  The module's user interface includes a graphical train simulator view (see Figure). Additional functions include automatic driving according to a Predefined Speed Profile and automatic generation of TIU inputs. The module uses simulated odometrical values, to avoid using Digital I/O cards for SUBSET-094 message exchange. The Train Behaviour Simulator interfaces with the ERTMS/ETCS Application for SUBSET-94 messaging, with the Balise Transmission Simulator module to share train position data, and with the Route Map Controller to receive track description and train data.

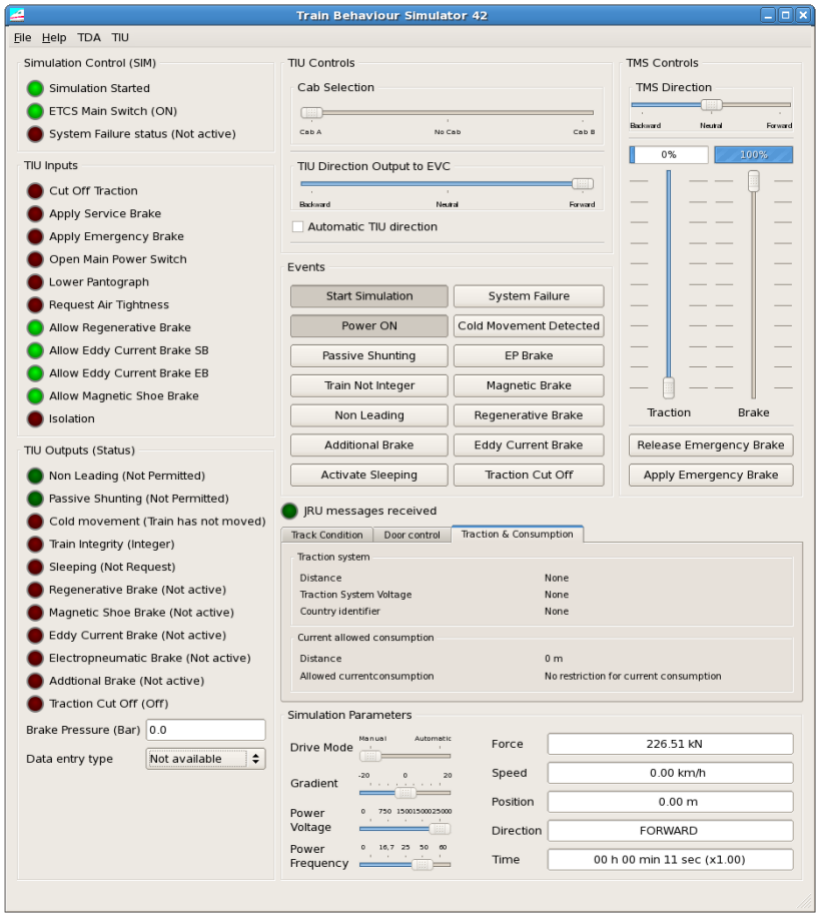


Figure 13 Train Behaviour Simulator user interface

- ***Balise Transmission Simulator:*** The Balise Transmission Simulator is responsible for delivering balise telegrams to the on-board unit at the correct track locations. Its implemented functions include sending balise telegram data to the OBU when the train reaches the corresponding location and receiving odometry data from the Train Behaviour Simulator module.
- ***ORD Simulator:*** The ORD Simulator acts as the onboard recording device to receive and store juridical records from the ERTMS/ETCS Application.
- ***RBC Message Simulator*** *(Radio Transmission Simulator):* The Radio Transmission Simulator transmits predefined radio messages to the ERTMS/ETCS Application based on the train's current location. Its implemented functions include receiving the train position from the Train Behaviour Simulator, sending track-to-train radio messages to the ERTMS/ETCS Application, receiving train-to-track radio messages from the ERTMS/ETCS Application, and supporting simulation of up to two RBCs.

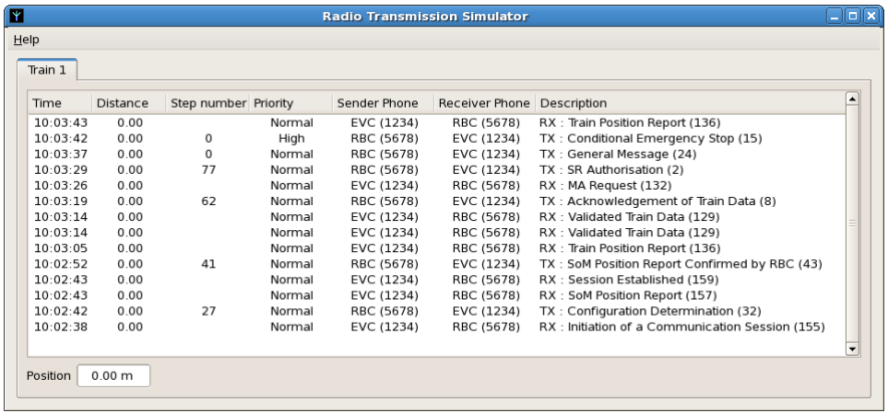


Figure 14 RBC Message Simulator (Radio Transmission Simulator) user interface

- ***Driver Machine Interface (DMI):*** The DMI is not simulated, but instead a hardware DMI is used. It is compliant with the ERA_ERTMS_015560 specification and uses TCP/IP to exchange information with the ERTMS/ETCS Application. For certain test scenarios we have also used a simulated DMI purely because it simplifies the documentation of the test cases.

Using the simulation environment described above, we created scenarios which, together with the ported ERTMS/ETCS Application, were used to verify several user stories documented in chapter 3.

2.3 IT/OT Security

Having a solid security architecture and all necessary defence mechanisms in place is crucial for critical infrastructure. The TS 50701 provides guidance on how to apply the IEC 62443 series of standards for industrial automation security in the railway context. The standard for safety-relevant railway communication EN 50159 states with respect to security, that for transmission systems that are potentially subject of malicious access, encryption is required.

From the modular platform point of view, in particular the integration with the security infrastructure of the railway operator is a key point. With this platform approach we can separate safety from security functions.

2.3.1 Realize Authentication and Authorization Mechanisms

The corresponding Foundational Requirement of TS 50701 is FR 1 "Identification and authentication control (IAC)", listing many system requirements.

Depending on the needs of the overall security concept, the approach here ranges from basic rights management for users and groups, to the integration of RADIUS, LDAP or Kerberos services and protocols. In any case it is important that the railway operator needs to have or build up the respective infrastructure and processes for access control.

2.3.2 Change Identity and Access Management Settings

The corresponding Foundational Requirement of TS 50701 is again FR 1 "Identification and authentication control (IAC)", the solutions from "Realize Authentication and Authorization Mechanisms" likewise applies.

2.3.3 Encrypt all Data on the Network

The corresponding Foundational Requirement of TS 50701 is FR 3 "System integrity (SI)" and as part of it the system requirement SR 3.1 RE(1) " Cryptographic integrity protection". Here, VPN solutions like IPsec or Wireguard can be used. Recently, also standardized safe protocols started including encryption with TLS, e.g. in SUBSET-146 v4.0.0. It is important to integrate with the proper infrastructure for certificate management on the railway operator side.

2.3.4 Realize Secure Communication over FRMCS

As above, the corresponding Foundational Requirement of TS 50701 is FR 3 "System integrity (SI)" and as part of it the system requirement SR 3.1 RE(1) " Cryptographic integrity protection". Secure communication over FRMCS could be realized as described in "Encrypt all Data on the Network" above. Note, that end-to-end encryption is key here.

2.3.5 Separation of Network Zones

Zoning is defined in B.3.2 in TS 50701 and requested that "*The boundaries of security zones should be protected by security gateways, firewalls or data diodes*". The relevant requirements are part of the foundational requirement FR 5 “Restricted data flow (RDF)”. As recommended, zones can be separated in an architecture such as that of the demonstrator. Firewall rules could be enforced on the individual CEs and the routers could be configured accordingly.

2.3.6 Detect Unusual Network Traffic

This can be seen as part of B.3.3 of TS 50701 where " Detection – Detect abnormal behaviour and trigger alerts for the rapid identification of a security breach, incident or suspect activity. " is part of defense in depth. Monitoring and inspecting network traffic on the individual CEs, as well as on the network infrastructure can be set up to detect irregular patterns. However, other than reporting such abnormal behaviour, no direct action can be taken by the safe system not to compromise safety and availability.

2.3.7 Detect Unauthorized Network Device

The same applies as for “Detect Unusual Network Traffic".

2.4 SOFTWARE UPDATE

Software update and configuration management is becoming a crucial topic for onboard modular platforms, as it will be increasingly necessary to update the software in the coming years:

- for cyber-security reasons, with the increased risks of cyber-security attacks and implementation of the Cyber Resilience Act.

- for technical reasons, to have the latest versions on the market available to improve regularly the overall RAMS of the systems, with the latest innovations and bug fixes.

As of 2026, most of the onboard software is updated manually, by a maintainer on the train, connecting locally to the onboard electronic unit and updating the onboard software using a maintenance laptop. Slowly, the number of train manufacturers and subsystem suppliers that offer remote software update of their electronics is increasing. Not least because railway operators are requesting remote updateability to decrease maintenance costs. Remote updates will become the standard way of updating onboard software in the coming years. With this new opportunity, it will be much easier to deploy a software update on a train fleet, as a maintainer will be able to initiate software and/or configuration update for a complete fleet, from his office.

Today, the software update and configuration management of onboard software is supplier and even solution specific. Several initiatives have been launched to standardise it:

- SNCF, DB, ÖBB, NS, SBB and RDG have published the EuroSpec "Specification for software updates of railways vehicles" in 2023.
- The System Pillar CONEMP domain is currently working on the standardisation of the update process applicable to Control Command and Signalling Systems. As the WP36 and the System Pillar started at the same time, the SP CONEMP deliverables related to software updates were not available as input data for this demonstrator.

For this deliverable we have decided to analyse the functionalities offered by the Hitachi GTS TAS Platform and to check whether the following software update related user stories defined in deliverable D36.1 can be fulfilled:

- 2.6.1 Update a Basic Integrity Application on Platform
- 2.6.2 Update a Basic Integrity Application on Safety Layer / RTE
- 2.6.3 Update a Safe Application on Safety Layer / RTE
- 2.6.4 Update the Safety Layer / RTE
- 2.6.5 Update the Platform
- 2.6.6 Rollback a Software / Configuration Update
- 2.6.7 Perform a Software / Configuration Update over the Air (FRMCS)

The following analysis is based on the information available in the Hitachi GTS "TAS Platform User Guide".

TAS Platform provides a disk image toolkit to generate disk images. Those disk images contain file-system images that bundle together all relevant files for an item, e.g. a file containing the root-filesystem of TAS Platform. For software updates a disk image file is used as input and the relevant file-system images are updated. An application can now use different filesystem images for their application binaries, the configuration data, etc.

Note that Hitachi GTS also provides an add-on to TAS Platform, the TAS Platform MNT, that allows safe and secure remote software update and maintenance, but this was not subject of this

demonstrator, its details are not described below. All use cases below could be fulfilled with this add-on, as well, since it also uses TAS Platform software_update command.

- 2.6.1 Update a Basic Integrity Application on Platform
  - One possible solution to update a Basic Integrity Application on the TAS Platform would be to create new disk image file using the disk image tools provided by Hitachi GTS.
  - The disk image file could be copied manually on the TAS Platform, and the software could be updated using the "software_update" command provided by Hitachi GTS.
- 2.6.2 Update a Basic Integrity Application on Safety Layer / RTE
  - The process would be similar as "2.6.1 Update a Basic Integrity Application on Platform".
- 2.6.3 Update a Safe Application on Safety Layer / RTE
  - The process would be similar as "2.6.1 Update a Basic Integrity Application on Platform".
- 2.6.4 Update the Safety Layer / RTE
  - An update of the Safety Layer or of the RTE would be provided by Hitachi GTS.
  - The update of the Safety Layer would be provided by Hitachi GTS as images that could be deployed as defined in "2.6.1 Update a Basic Integrity Application on Platform".
- 2.6.5 Update the Platform
  - The TAS Platform can be updated using the OS Update feature designed by Hitachi GTS.
  - A new disk image would be provided by Hitachi GTS as for 2.6.4 and deployed as in "2.6.1 Update a Basic Integrity Application on Platform".
- 2.6.6 Rollback a Software / Configuration Update
  - There are 2 ways to rollback a software running on the TAS Platform.
  - The first one is to use the "software_update" command provided by Hitachi GTS, with the parameter " -- switch".
  - The other one is to simply update the software with the previous version as described in "2.6.1 Update a Basic Integrity Application on Platform".
- 2.6.7 Perform a Software / Configuration Update over the Air (FRMCS)
  - Without the MNT add-on, it is not directly possible to update the software over the air. However, if a connection to TAS Platform could be established via FRMCS or other means of communication accessible by the onboard system, the update could be performed as described in "2.6.1 Update a Basic Integrity Application on Platform".

2.4.1 Summary

For all user stories potential implementations/demonstration options have been described with respect to the functionality available in the demonstrator. Furthermore, using the TAS Platform add-on MNT these user stories would be fulfillable without executing manual steps on the TAS Platform system.

It would be very beneficial for the sector to standardise the software update and configuration management of onboard software functionalities and protocols, to allow smooth integration between the on-board software delivered by the suppliers / train manufacturers and the operators' software configuration and update tools.

This should be studied and tested in future R&D projects.

2.5 FUNCTIONAL CLUSTER TRAIN INTEGRATION

The functional cluster 'Train Integration' shall demonstrate how TCMS data can be transmitted and aggregated from a (virtual) train to CCS client applications. Especially in future GoA4 train operation scenarios, such aggregated data is most likely needed to optimize train operations.

The functional cluster 'Train Integration' is related to the user stories 'Collect and Aggregate Diagnostics Data from Vehicle' and 'Provide Access to Train Hardware via a TCMS Data Service ', as introduced in chapters 2.5.10 and 2.9.2 of the document D36.1 User Stories & Test Cases [2].

Activities in WP36 reflect the realization of concepts from WP31, including deployment and testing of the TCMS Data Service (TCMS_DS) as defined in WP31. Please refer to the Deliverable D31.2 "Validation of TCMS Data Service concept and formalisation" [4] for further details.

It has to be noted that the WP36 realisation as presented in this chapter is not connected to MCP-DIA (please refer to chapter 2.1). This is due the fact that both solutions got developed separately and a final integration has never been foreseen due to a lack of resources and project timeline (the integration of TCMS_DS was originally planned already for the previous deliverable D36.3. The reasonable aim of providing diagnostics vehicle data as well via MCP-DIA may be valid but won't happen in WP36. Another reason is that the main objective of DIA-VEC, the provision of aggregated TCMS data to potential CCS applications, is already fulfilled with the available setup.

The following figure provides a high-level overview of the data flow and involved components:

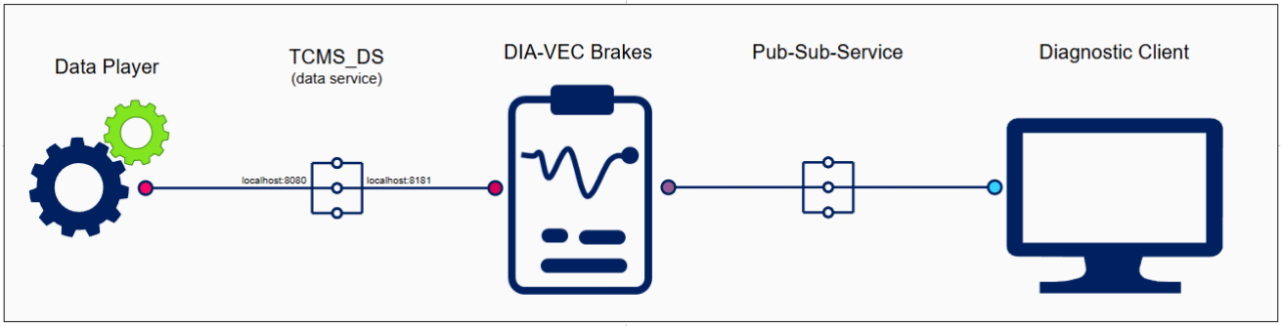


Figure 15 Data Flow and Involved Components

In D36.2 [5] chapter 2.4.1 proposed software components have already been listed and are now further described in the following table:

Software Component	Description
Simulation of a brakes system of a train (provides data as input for TCMS Data Service)	For reproducible results and easier handling, a bash script ('Data Player') is used. The Data Player basically provides the same messages as the MATLAB Simulink simulation, but in a fully reproducible way. The messages can be adapted to specific use cases and contain states as well as error messages. The Data Player is executed on the virtual machine 'Diagnostic VM' on the lab server (see chapter 2.3.1.8 of D36.3 [3]).
TCMS Data Service (TCMS_DS)	TCMS_DS is also executed on the virtual machine 'Diagnostic VM' and accessibly by the modular platform via the VLAN (OB_BULK).
The Basic Integrity application DIA-VEC (Brakes)	The publish subscribe service is part of the Basic Integrity application DIA-VEC Brakes and provides the possibility for clients to subscribe to provided data. A client can subscribe to all data, but also to just a specific part in respect to reduce the amount of transmitted data. The DIA-VEC runs as basic integrity application on the modular platform.
DIA-VEC client	A Node Red client will store all received data in a logfile. This logfile is used in an automatically executed integration test, to check the correctness of the dataflow. This placeholder for a downstream CCS application is also running on the virtual machine 'Diagnostic VM', subscribing to the DIA-VEC via the VLAN (OB_BULK).

Table 1 Train integration Software Components

2.5.1 Deployment in the SBB Laboratory

For the demonstrator of WP36 functional cluster 'Train Integration", the components are running either on the virtual machine 'Diagnostic VM' or on the modular platform itself. Both are connected to the same network: OB_BULK (VLAN_130), so all communication between components is done via this ethernet network. The deployment of the components is realised via the communication network OB_ADMIN (VLAN_100) with an Azure DevOps pipeline agent hosted on the "Cloud Integration VM".

The following Figure visualises the deployment of the software components and communication network in the laboratory setup.

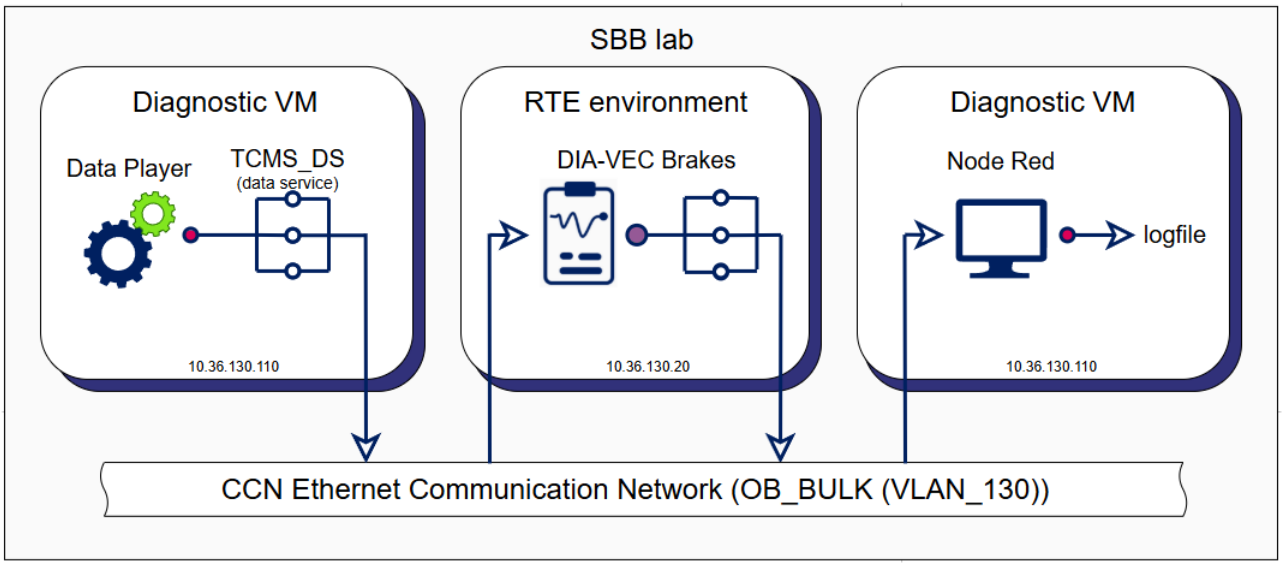


Figure 16 Deployment and Communication Network in SBB lab

2.5.2 Automated Deployment and Test

The following figure shows latest runs of the realized deployment pipeline, which deploys the presented software components automatically to the lab, runs the tests and archives the test results that are presented in chapter 3.24.

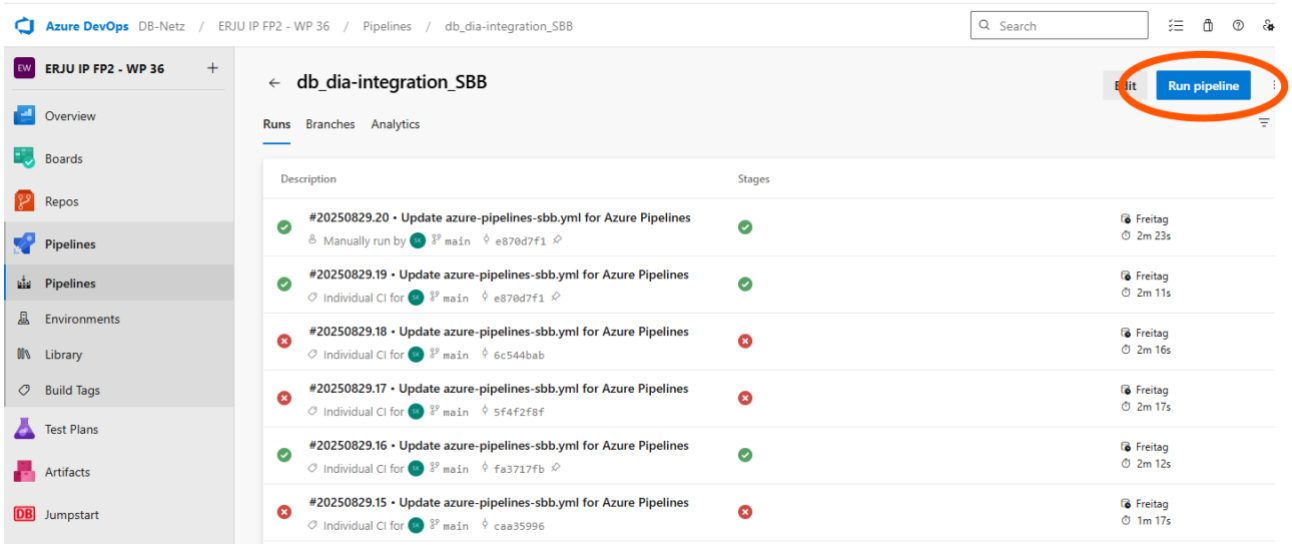


Figure 17 Automated Deployment and Test

2.5.3 Simulation of Train Data

Simulated vehicle data gets replayed as a recording from DB's MATLAB/Simulink model CoSimBrakes for the sake of reproducibility.

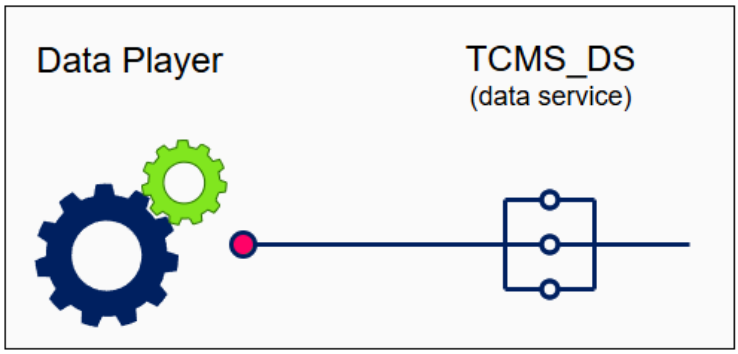


Figure 18 Data Player

The Data Player is provided in an Azure artifact and executed in a Docker container on the virtual machine 'Diagnostics VM'. The Docker container is started automatically inside the pipeline 'db_dia-integration_SBB', which can be started with the click of a button in a browser for deployment and testing.

The Data Player is designed to send states and errors in a reproducible manner to the TCMS_DS.

2.5.3.1 Definition of states

S01='{"base":{"fs":"inop","hm":"noerr"},"stat":{"sb":"unknown","pb":"released"},"perfm":{"brkp":"0","perf":"0"}}'

Hereby the value of the states are marked in blue.

For the first state "fs":"inop" you could also define "fs":"op", when the functional state shall state 'operational' instead of 'inoperative'.

2.5.3.2 Definition of errors

EC148ON='{"id":"C148","ts":"2025-07-01T11:10:00Z","cond":"come"}'

Hereby the error is defined via an error code, a timestamp and the information whether this error is set or reset.

The order of states and errors in the bash script file can be chosen as needed for a specific use case, also it is possible to in- or decrease the number of states and errors. The components for the definition of error codes can be found in the YANG model provided as an Azure artifact.

For the actual test case the following states and errors are defined as follows:

```
# State definitions
S01='{"base":{"fs":"inop","hm":"noerr"},"stat":{"sb":"unknown","pb":"released"},"perfm":{"brkp":"0","perf":"0"}}'
S02='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"selfTest","pb":"engaged"},"perfm":{"brkp":"0","perf":"0"}}'
S03='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"holdingBrake","pb":"engaged"},"perfm":{"brkp":"140","perf":"100"}}'
```

S04='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"holdingBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}'

S05='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"noServiceBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}'

S06='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"serviceBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}'

S07='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"noServiceBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}'

S08='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"serviceBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}'

S09='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"noServiceBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}'

S10='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"slowingBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}'

S11='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"stoppingBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}'

S12='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"holdingBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}'

S13='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"holdingBrake","pb":"engaged"},"perfm":{"brkp":"140","perf":"100"}}'

S14='{"base":{"fs":"inop","hm":"noerr"},"stat":{"sb":"unknown","pb":"engaged"},"perfm":{"brkp":"0","perf":"0"}}'

S100='{"base":{"fs":"op","hm":"error"},"stat":{"sb":"noServiceBrake","pb":"released"},"perfm":{"brkp":"115","perf":"80"}}'

Listing 10 States of the Brakes Simulation

In case of manual modification, the following values can be used:

'fs' defines the functional state and has one of the following values

State	Description
op	the Brakes system is operational, provided data by the brakes system are reliable
inop	the Brakes system is not operational, e.g. during a self-test

'hm' defines the health state and has one of the following values

State	Description
noerr	the Brakes system has no error
error	the Brakes system has active error(s)

'sb' defines the status of the service brake system and has one of the following values

State	Description
selftest	the Selftest of the Brakes system is active
servicebrake	the Servicebrake is active
slowingbrake	the Slowingbrake is active
stoppingbrake	the Stoppingbrake is active
holdingbrake	the Holdingbrake is active
quickbrake	the Quickbrake is active
noservicebrake	the Servicebrake is not active

'pb' defines the status of the brakes system and has one of the following values

State	Description
released	the Parkbrake is released
engaged	the Parkbrake is engaged

‘brkp’ is the value of the available brake percentage of the brake system. In our model with 5 pneumatic brakes (one in each bogie) 135 is defined as maximum.

‘perf’ is the normalised value of brkp, which means perf = 100 (%) when brkp = 135.

The errors can be set and reset by using a specific hexadecimal error code with a timestamp and the condition, whether the error is set (come) or reset (gone).

```
# Error definitions
EC148ON='{"id":"C148","ts":"2025-07-01T11:10:00Z","cond":"come"}'
EC148OFF='{"id":"C148","ts":"2025-07-01T11:10:00Z","cond":"gone"}'
E278AON='{"id":"278A","ts":"2025-07-01T11:20:00Z","cond":"come"}'
E278AOFF='{"id":"278A","ts":"2025-07-01T11:20:00Z","cond":"gone"}'
E7116ON='{"id":"7116","ts":"2025-07-01T11:30:00Z","cond":"come"}'
E7116OFF='{"id":"7116","ts":"2025-07-01T11:30:00Z","cond":"gone"}'
E7118ON='{"id":"7118","ts":"2025-07-01T12:00:00Z","cond":"come"}'
E7118OFF='{"id":"7118","ts":"2025-07-01T12:0:00Z","cond":"gone"}'
```

Listing 11 Error Definitions of the Brakes Simulation

2.5.4 TCMS_DS

TCMS_DS represents a prototypical implementation of a model-based approach for specifying and implementing a data service that hosts operational information about onboard rolling stock systems, such as doors, brakes and others. Further details are available in the document: ERJU WP31 'Validation of TCMS Data Service concept and formalization' [4].

TCMS_DS is expected to be managed by the central Train Control and Management System (TCMS). For the demonstrator in the WP36 functional cluster 'Train Integration', DB InfraGO developed a prototypical TCMS_DS implementation focusing on the brake system.

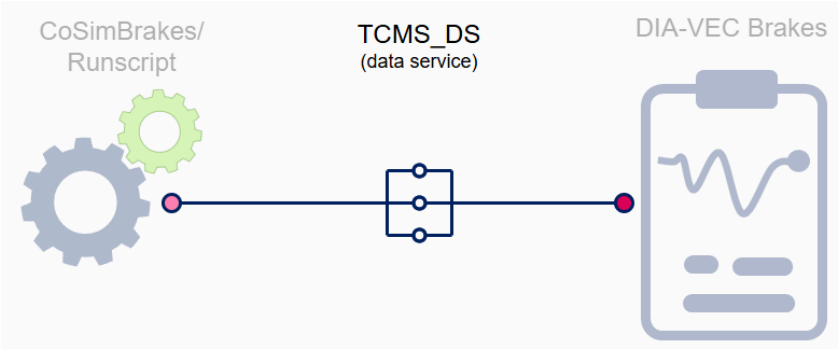


Figure 19 TCMS_DS

TCMS_DS is provided in an Azure artifact and executed in a Docker container on the virtual machine 'Diagnostics VM'. The Docker container is also started automatically inside the pipeline 'db_dia-integration_SBB'.

Before the TCMS_DS can receive data from the TCMS (here represented by a Data Player) and transmit data to DIA-VEC Brakes, the data structure must be defined in TCMS_DS beforehand. The definition of data structures for TCMS_DS is done using a YANG[1] model.

Examples of data definition for the TCMS_DS and clustering in the YANG model are shown for the sb-status (service brake status) listing below:

```
typedef sb-status {
      type enumeration {
          enum unknown {value 0;}
          enum selfTest {value 1;}
          enum serviceBrake {value 2;}
          enum slowingBrake {value 3;}
          enum stoppingBrake {value 4;}
          enum holdingBrake {value 5;}
          enum quickBrake {value 6;}
          enum noServiceBrake {value 7;}
      }
  }


  grouping brake-stat {
      leaf sb {
          type sb-status;
```

[1] Yet Another Next Generation (YANG) is a data modelling language for the definition of data sent over network management protocols such as the NETCONF and RESTCONF. The YANG data modelling language is maintained by the NETMOD working group in the Internet Engineering Task Force (IETF) and initially was published as RFC 6020 in October 2010, with an update in August 2016 (RFC 7950).

```
            default noServiceBrake;
        }

    grouping bcu-events {
        notification onUpdate {
            leaf sb {
                type leafref {
                    path '../../stat/sb';
                }
            }

    container brakes {
        list bcu {
            key "id";
            unique "consist car";
            uses bcu-events;
        }
    }
```

Listing 12 TCMS Data Service Definition Example

The data in a YANG model can be clustered according to specific needs. The figure below visualises the clustering of data used in the demonstrator.

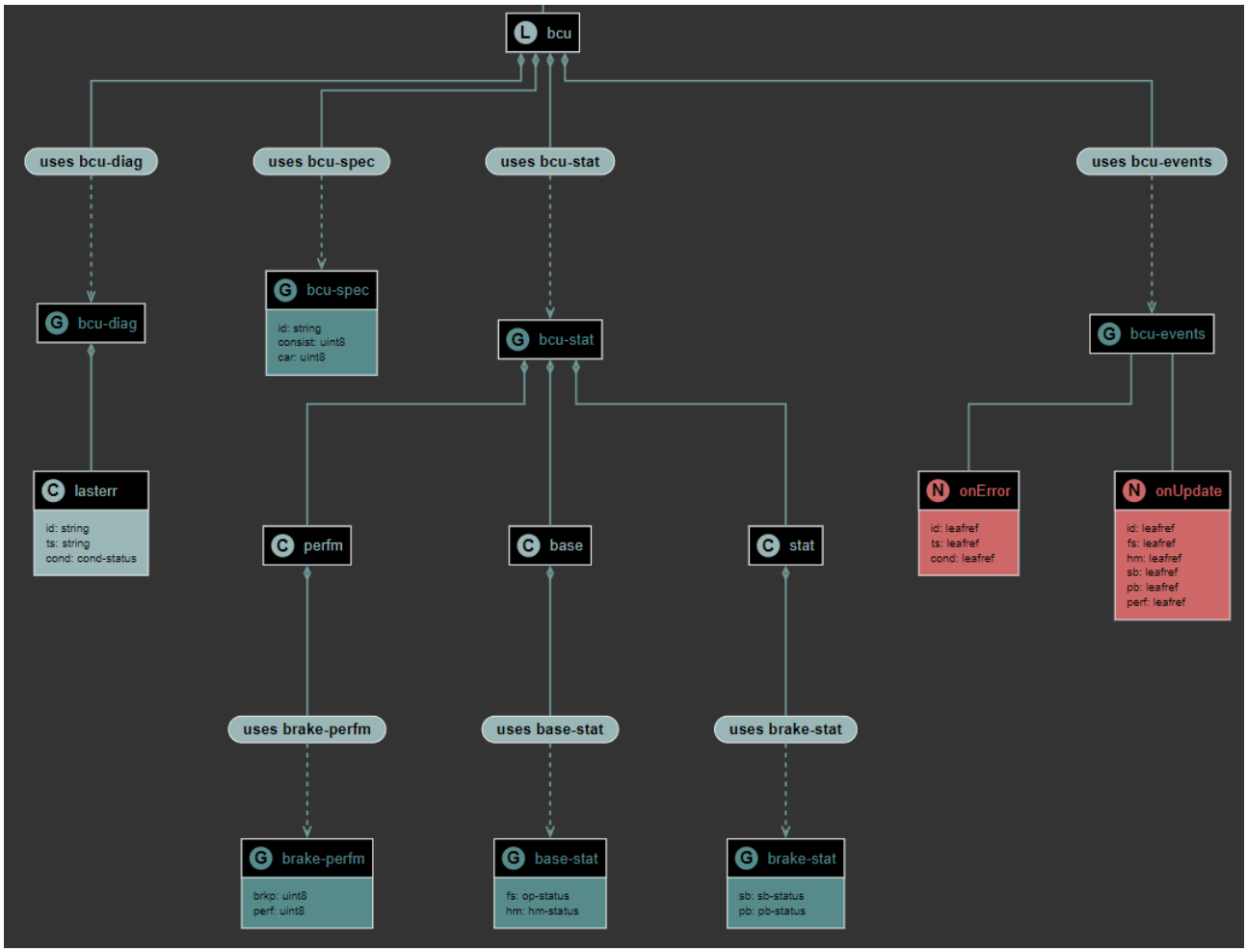


Figure 20 Visualisation of YANG model

2.5.5 DIA-VEC Brakes

DIA-VEC is a MATLAB Simulink library developed by DB InfraGO that can be configured to receive, aggregate, and transmit data from various train systems. The specific configuration of DIA-VEC

determines which signals from a train system are processed, effectively transforming the generic library into a system-specific one. For R2DATO WP36, a configuration was developed to handle input data from the brake system; this combination is referred to as 'DIA-VEC Brakes'. While configurations for additional train systems (such as doors) are possible, the demonstrator within the WP36 functional cluster 'Train Integration' is focused exclusively on the brake system.

In this demonstrator DIA-VEC is provided as an Azure artifact and executed on the modular platform.

DIA-VEC is automatically deployed and started by the pipeline ‘db_dia-integration_SBB’.

DIA-VEC Brakes comprises a subscription block, three distinct types of software modules – collectively referred to as ‘agents’ – and a publish-subscribe service. Each agent serves a specific function:

- Subscription block: The subscription block of DIA-VEC uses a REST-based service with RPC (remote procedure call) to establish communication with TCMS_DS. Notifications from TCMS_DS are handled through WebSockets.
- Mon-Agent (Monitoring Agent): This agent receives data and checks whether the current value differs from the previous one. If changes are detected, the Mon-Agent extracts the updated signals and values, routing them to the Test-Agent via a message queue. Multiple Mon-Agents can be configured.
- Test-Agent: The Test-Agent receives signals and values from the Mon-Agent and evaluates them against specific conditions, as required for further aggregation in the Fusion-Agent. After successful validation, it passes the relevant signals, values and conditions to the Fusion-Agent. Multiple Test-Agents can be configured.
- Fusion-Agent: The Fusion-Agent collects signals, values and conditions from at least one Test-Agent and applies expert rules to this data. These rules are defined in a configuration file (BDD_xyz.xls), generated from a DrawIO configuration. For advanced logic, CLIPS[2] can be utilized. There is always just one single Fusion-Agent, which integrates input from all Test-Agents.
- Publisher: The Publisher receives signals, values, and expert rule evaluation results from the Fusion-Agent and distributes them to subscribed clients. The Publisher provides a publish-subscribe service, allowing clients to subscribe and specify their data interests. In the WP36 demonstrator, Node-RED subscribes to the Publisher to receive all available data. As already mentioned, DIA-VEC does not integrate with MCP-DIA. This would technically of course be feasible to be realised, by replacing the current Publisher with an implementation, representing an MCP-DIA application (as described in chapter 2.1.1.). For the integration with MCP-DIA it doesn’t make any difference if such a Publisher runs as Model-3 Taskset on the TAS Platform RTE or as a parallel application on the underlying operating system, like the DIA-VEC currently does.

[2] *CLIPS was developed at NASA's Johnson Space Center from 1985 to 1996, the ‘C Language Integrated Production System’ (CLIPS) is a rule-based programming language useful for creating expert systems and other programs where a heuristic solution is easier to implement and maintain than an algorithmic solution. Written in C for portability, CLIPS can be installed and used on a wide variety of platforms. Since 1996, CLIPS has been available as public domain software.*

Depending on the system under consideration (train system), the number of Mon-Agents and Test-Agents might differ but at least one Mon-Agent and one Test-Agent must be used. DIA-VEC is using exactly one Fusion-Agent independently from the system under consideration.

DIA-VEC Brakes as configured for WP36, consists of two Mon-Agents and two Test-Agents. This setup is based on train types with two master brakes gateway units (GUs), so each GU is connected to one Mon-Agent.

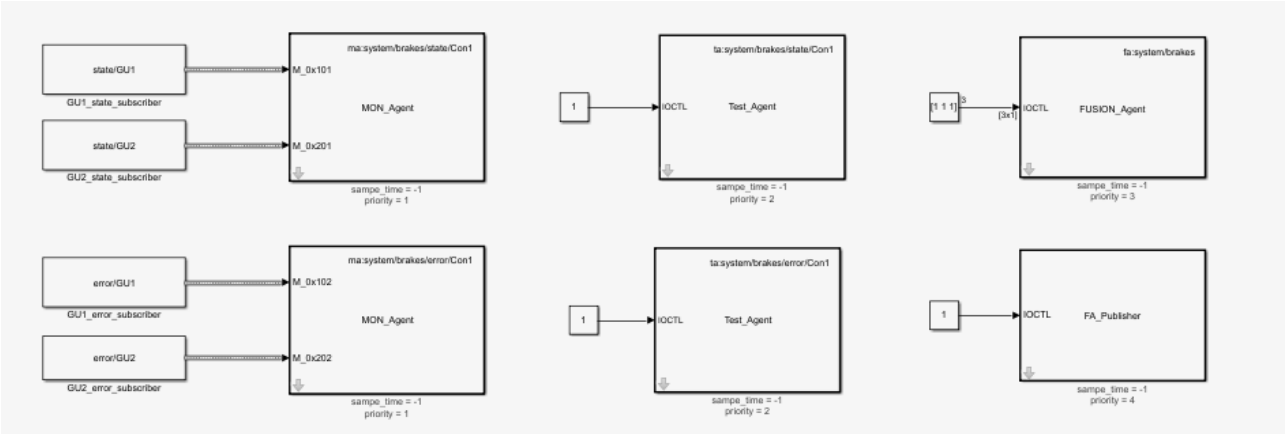


Figure 21 Modules/Agents of DIA-VEC

2.5.5.1 DrawIO Configuration

As stated in the above chapter, the ‘configuration’ for DIA-VEC turns the generic library into a specific one. This ‘configuration’ is provided as an Excel file named BDD_<xyz>.xls. <xyz> is replaced with the specific train system, for the WP36 demonstrator of DIA-VEC in combination with data from a Brake system, it will be then: BDD_brakes.xls.

There are two different methods to set up the configuration file:

a) create an Excel file manually, based on a template

b) create an XML file using DrawIO and automatically parse and convert it into Excel format

Method a) requires just Excel, but working in the specific template file is difficult, therefore method b) is preferred.

Method b) consists of 2 steps:

1) Create a DrawIO file

2) Parse the DrawIO file (automatically) and extract the content into the BDD file (also automatically)

DrawIO is a tool for creating drawings and provides the option to use predefined objects. Further on these objects contain properties, which enable the usage of DrawIO as a configuration tool for the DIA-VEC MATLAB/Simulink model.

The following DrawIO configuration example displays how a diagnostic failure degrades the associated train service. Corresponding train operation consists of different services, such as service brake, emergency brake, driving/acceleration, door opening, closing and locking, etc. Each service

is realised via hardware components/resources and their corresponding software. Every hardware component is monitored via self-tests or by other components to ensure correct functionality.

Failures are indicated by error codes. The configuration in the figure below shows how a specific error code affects a particular train service.

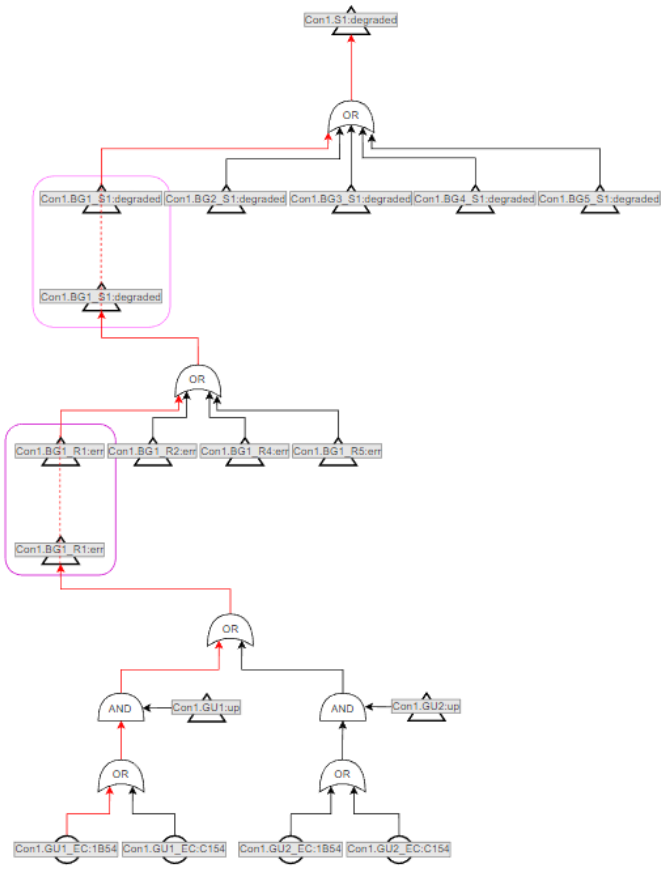


Figure 22 DrawIO configuration

DIA-VEC Brakes receives the error code 0x1B54 (Con1.GU1_EC:1B54), in the bottom left corner of the above figure. The error code 0x1B54 is correlated to the pneumatic brakes system and leads to a resource failure R1 for bogie BG1 (Con1.BG1_R1:err). The resource failure R1 at bogie BG1 results in a degradation of the service brake service for bogie 1 (Con1.BG1_S1:degraded). The service brake uses besides the pneumatic brakes hardware also the electro dynamic propulsion system. The electro dynamic propulsion system is excluded in the figure above.

Finally, a degradation of the service brake for bogie 1 results in a degradation of the service brake for the complete train (Con1.S1:degraded). This information will then be published to the clients of DIA-VEC Brakes.

This example displays the correlation between an error code and the degradation of services. When several error codes at the same time are active, the evaluation of degraded services is helpful especially in future operation scenarios.

2.5.5.2 Agent Sequence Diagram

The components of the DIA-VEC MATLAB/Simulink model act ‘onUpdate’: any task inside DIA-VEC is only executed when values of input data had changed. This reduces working load in the agents

because they act only after a detection of modified/updated input values. In case the identical input value is received by an agent, no task will be executed.

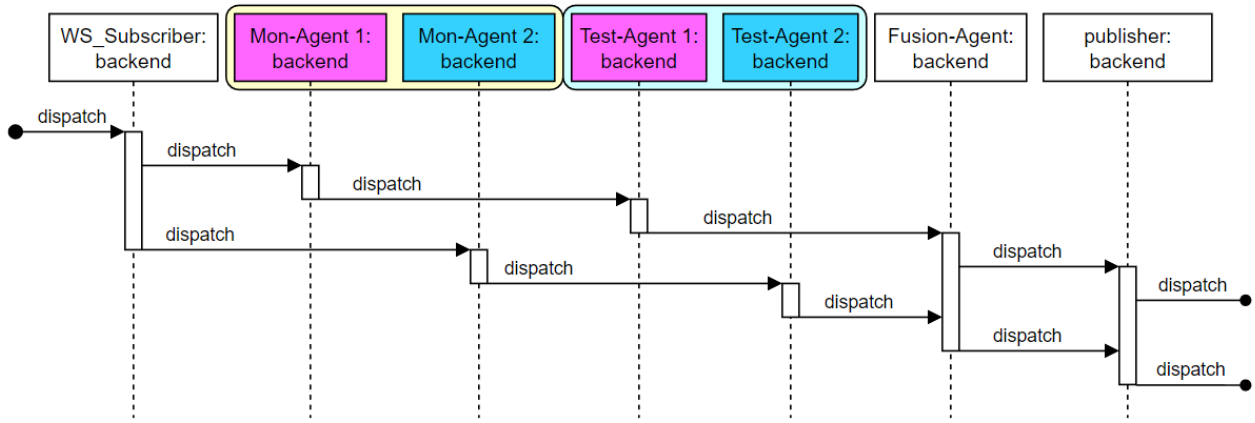


Figure 23 Sequence diagramm

2.5.5.3 Logfile DIA-VEC

DIA-VEC stores information in a logfile (*.log). This information contains activities and states, that are helpful when it comes to trouble shooting but are also used during the integration test.

When analysing the content of the DIA-VEC logfile, the first entry after an open bracket (“[“) indicates which agent/application of DIA-VEC reports this entry. Besides the three agents, there are two additional modules which provide content in the log file:

- WS_Subscriber: WebSocket subscriber towards TCMS_DS
- MA: Monitoring Agent
- TA: Test Agent
- FA_FDM: Fusion Agent
- FA_Publisher: Publisher towards the (Node Red) test client

As a brief intro, some examples of entries of the logfile are given below.

After startup, the application DIA-VEC adds information about initialisation of specific services, e.g.:

```
[FA_FDM_0x077F::mdlInitializeSampleTimes] running ...
[FA_Publisher_0x077F::mdlInitializeSampleTimes] running ...
[FA_RxSocket_0x077D::mdlInitializeSampleTimes] running ...
[FA_RxSocket_0x077E::mdlInitializeSampleTimes] running ...
[TA_IVT_0x077D::mdlInitializeSampleTimes] running ...
[TA_IVT_0x077E::mdlInitializeSampleTimes] running ...
[TA_RxSocket_0x0701::mdlInitializeSampleTimes] running ...
[TA_RxSocket_0x0702::mdlInitializeSampleTimes] running ...
[WS_Subscriber_BrakeError::mdlInitializeSampleTimes] running ...
[WS_Subscriber_BrakeState::mdlInitializeSampleTimes] running ...
```

During runtime the application DIA-VEC adds information about the data each service is dealing with, e.g.:

```
[FA_Publisher_0x077F::mdlOutputs pubData]
{"key":"dbea115a6715bf032a8df94e8479f48c","cat":"Test","cond":1.000000,"ts":"2025-02-
18T16:06:18.653-1:00"}
[FA_Publisher_0x077F::mdlOutputs pubTopic]  dbs.diavec.brakes:system/brakes/state/Con1/GU1/fsop
```

```
[FA_Publisher_0x077F::mdlOutputs pubData]
{"key":"2ece7ff1d9dd3f374db828eb51213d71","cat":"Logic","cond":1.000000,"ts":"2025-02-
18T16:06:18.656-1:00"}
[FA_Publisher_0x077F::mdlOutputs pubTopic] dbs.diavec.brakes:system/brakes/status/Con1/GU1/up
[FA_Publisher_0x077F::mdlOutputs
pubData]{"key":"91bd4cbc28c40fa793b3468e604ee61b","cat":"Logic","cond":1.000000,"ts":"2025-02-
18T16:06:18.659-1:00"}
[FA_Publisher_0x077F::mdlOutputs pubTopic] dbs.diavec.brakes:system/brakes/service/Con1/SandUnit/up
```

During shut down DIA-VEC adds information about closing and cleaning of the services, e.g.:

```
[WS_Subscriber_BrakeError::mdlTerminate] finished
[MA_TxSocket_0x0702::mdlTerminate] Closing ZMQ PUSH socket
[MA_TxSocket_0x0702::mdlTerminate] finished
[WS_Subscriber_BrakeState::mdlTerminate] finished
[MA_TxSocket_0x0701::mdlTerminate] Closing ZMQ PUSH socket
[MA_TxSocket_0x0701::mdlTerminate] finished
[TA_TxSocket_0x077E::mdlCleanupRuntimeResources] Terminating ZMQ context
[TA_TxSocket_0x077E::mdlCleanupRuntimeResources] Freeing runtime memory
[TA_TxSocket_0x077E::mdlCleanupRuntimeResources] finished
[FA_FDM_0x077F::mdlCleanupRuntimeResources] Destroying CLIPS environment
[FA_FDM_0x077F::mdlCleanupRuntimeResources] finished
```

2.5.6 Node Red client

Node Red is a graphical UI tool for collecting and handling data. It is easy and intuitive to use, which makes it a good choice for demonstrator projects like WP36. Like TCMS_DS and DIA-VEC Brakes, Node Red is an application that can handle event-driven data, so it integrates perfectly in the overall data flow of this demonstrator. For further information on the tool itself, please refer to https://nodered.org/

In the deployment and execution pipeline, Node Red gets provided as an Azure artifact and gets deployed and executed in a Docker container on the virtual machine ‘Diagnostics VM’. The Docker container is started automatically by the pipeline ‘db_dia-integration_SBB’.

Node Red features predefined objects (called Nodes), each node provides specific functions. These Nodes can be interconnected to combine and connect their functionalities. Additional functionality can be integrated through plug-ins, allowing Node Red to be tailored for specific requirements.

Predefined objects are e.g. ‘connect via Ethernet’ using ZeroMQ, ‘convert data into a specific format’ (e.g. *.json), ‘display data for debugging’, ‘display data for visualisation’ (on user defined dashboards), ‘save data into files’, etc.

In the demonstrator setup Node Red acts as a visualisation and test client, implementing the destination for the data flow and aggregated data (please refer to chapter 2.5.5 above).

In real train environments, the output of DIA-VEC can be provided to such systems that need health states of the train and its subsystems. Such TCMS diagnostics data might be useful and potentially needed especially in future GoA4 operation, of course with the limitation that the data is not provided with a Safety Integrity Level applied.

3 USER STORIES WITH TEST CASES AND RESULTS

3.1 OVERVIEW USER STORIES

The following table provides a status overview of all User Stories addressed in task 36.4:

Group / Cluster	User Story according to D36.1	Title (for the description, please refer to [2] D36.1 User Stories document)	Status
Deployment & Orchestration			
	[2.1.4]	Run Multiple Applications	tested
	[2.1.6]	Restart Failing Replicas	tested
	[2.1.7]	React to Hardware Failure	tested
	[2.1.8]	Failover to Cold Backup Hardware	omitted
Modularity			
	[2.2.1]	Change to Different Hardware	omitted
	[2.2.2]	Change to Different Platform or Safety Layer / RTE	omitted
	[2.2.3]	Use Diverse Hardware	omitted
Communication			
	[2.3.2]	Communicate Safe Application on RTE <-> External Safe Application	omitted
	[2.3.9]	Communicate Parallel Basic Integrity App <-> External Basic Integrity App	tested
	[2.3.10]	Communicate Independent of RTE Instance	omitted
	[2.3.11]	Communicate Independent of Replication	tested
FRMCS			
	[2.4.2]	Reuse FRMCS Platform Functions	tested
	[2.4.3]	Integrate an Application with FRMCS over Loose Coupling	tested
MDCM			
	[2.5.1]	Collect Diagnostics Data from External Basic Integrity App	tested
	[2.5.2]	Collect Diagnostics Data from Basic Integrity App on RTE	tested
	[2.5.3]	Collect Diagnostics Data from Safe Application on RTE	tested
	[2.5.4]	Collect Diagnostics Data from Safety Layer / RTE	tested
	[2.5.5]	Provide Diagnostics Events to External Basic Integrity App	tested
	[2.5.6]	Provide Diagnostics Events to Basic Integrity App on RTE	tested
	[2.5.7]	Provide Diagnostics Events to Safe Application on RTE	omitted
	[2.5.8]	Provide Diagnostics Events to Safety Layer / RTE	omitted
	[2.5.9]	Exchange Diagnostics Data with Trackside Diagnostics Service	omitted
	[2.5.10]	Collect and Aggregate Diagnostics Data from Vehicle	tested
Software Update			
	[2.6.1]	Update a Basic Integrity Application on Platform	analysed
	[2.6.2]	Update a Basic Integrity Application on Safety Layer / RTE	analysed
	[2.6.3]	Update a Safe Application on Safety Layer / RTE	analysed
	[2.6.4]	Update the Safety Layer / RTE	analysed
	[2.6.5]	Update the Platform	analysed
	[2.6.6]	Rollback a Software / Configuration Update	analysed
	[2.6.7]	Perform a Software / Configuration Update over the Air (FRMCS)	analysed
Onboard Network			
	[2.7.1]	Allow Exchange of Data over the Network	postponed to WP36.5
	[2.7.2]	Provision of Uninterrupted Connectivity	tested
	[2.7.3]	Allow Management Access to Onboard Components	omitted
	[2.7.4]	Allow Ease of Access to the Network	omitted

Group / Cluster	User Story according to D36.1	Title (for the description, please refer to [2] D36.1 User Stories document)	Status
	[2.7.5]	Supply List of Onboard Network Components	omitted
	[2.7.6]	Provision of Date Time Reference	tested
	[2.7.7]	Prioritise Specific Network Traffic	postponed to WP36.5
	[2.7.8]	Monitor the Network Health Status / Guaranteed Characteristics	postponed to WP36.5
IT/OT Security			
	[2.8.1]	Realize Authentication and Authorization Mechanisms	analysed
	[2.8.2]	Change Identity and Access Management Settings	analysed
	[2.8.3]	Encrypt all Data on the Network	analysed
	[2.8.4]	Realize Secure Communication over FRMCS	analysed
	[2.8.5]	Separation of Network Zones	analysed
	[2.8.6]	Detect Unusual Network Traffic	analysed
	[2.8.7]	Detect Unauthorised Network Device	analysed
Train Abstraction			
	[2.9.1]	Provide Safe Access Train Hardware via a Functional Vehicle Adapter	omitted
	[2.9.2]	Provide Access to Train Hardware via a TCMS Data Service	tested
Functional Apps			
	[2.10.1]	Execute a Simplified ATO Control Loop	omitted
	[2.10.2]	Execute a Simplified ETCS Control Loop	tested
	[2.10.3]	Cache Map Data from Digital Register	omitted

Table 2: Status overview of addressed User Stories

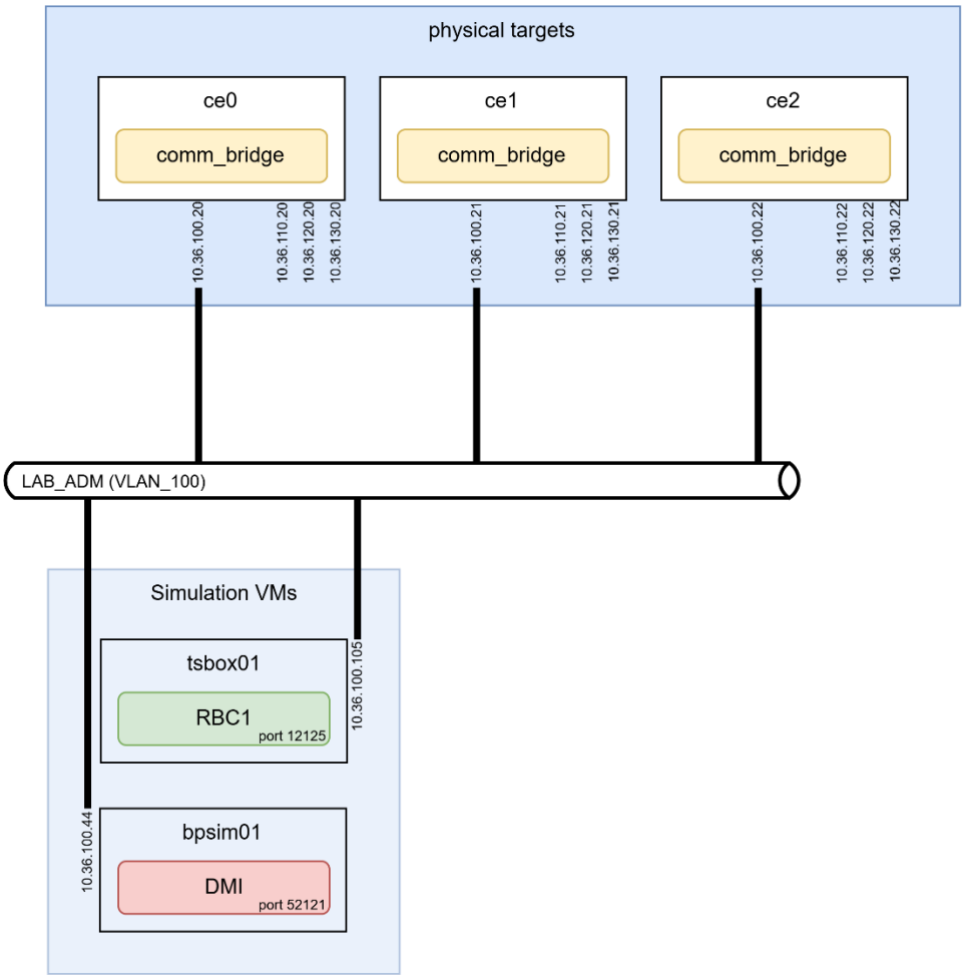
3.2 Run Multiple Applications [2.1.4]

Description User Story	As R2DATO WP36, we want to run multiple combinations of platform native basic integrity and safe (up to SIL 4) applications on the platform as well as the safety layer / runtime environment, so that we can demonstrate the freedom of interference between the deployments.
Annotation	Covered with 3.17 Collect Diagnostics Data from Safe Application on RTE [2.5.3].

3.3 Restart Failing Replicas [2.1.6]

Description User Story	As R2DATO WP36, we want the safety layer / runtime environment to automatically restart failing replicas of safe applications (up to SIL 4), so that we can demonstrate the resilience in case of spontaneous (not recurring) software / hardware failure.

Test Case Title	**Re-integrate one CE after a software failure**
Test Case ID	2.1.6-A
Description	Based on a setting in a configuration file, one replica / computing element (CE) will return an invalid result to the voting system (safety layer) of the platform. Thus, the failed CE shall be restarted and re-integrated into the communication/voting system of the TAS Platform and continue normal operation.
Test Environment	For this user story, the architecture blueprint was configured to connect directly to the trackside application without using the FRMCS communication setup. Replica #1 (ce1) was configured to simulate a computation failure resulting in a voting error on the platform.



The following entry in the configuration file will trigger ce1 to send an invalid result to the voting system, causing the TAS Platform to isolate and shut down the failed replica.

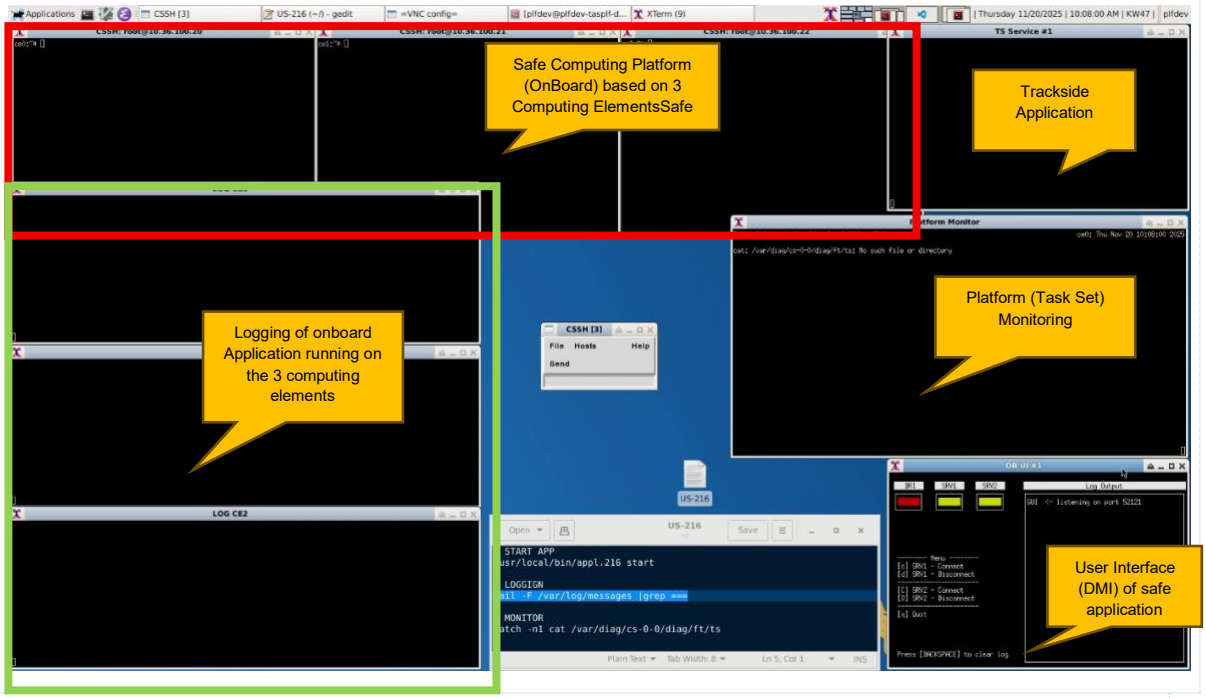
```
…
    "testconfig": {
        "votingError": {
            "computingElement": "ce1",
            "sequence": 3
        }
....}

…
```

Preconditions	Set up Test environment in a manner that runs long enough for this Test Case.
Test Steps	1. Prepare testbench 2. Start testbench components and monitoring a. Start trackside application b. Start onboard user interfaces (DMI) c. Start the onboard safe application (architecture blueprint) d. Start platform monitor

	3. Initiate connections from onboard application to the trackside service. 4. Verify message exchange between onboard applications and trackside applications. 5. Monitor failing replica at message #3 6. Application continued to operate in a 2oo2 configuration until replica #1 is re-integrated. 7. Manually reboot failing replica, since the platform is configured to not automatically reboot. 8. Monitor re-integration of restarted replica.
Test Data	None.
Expected Result	During the full test, the system must continue to perform its intended functions, without any detectable issues or degradation.

Actual Result

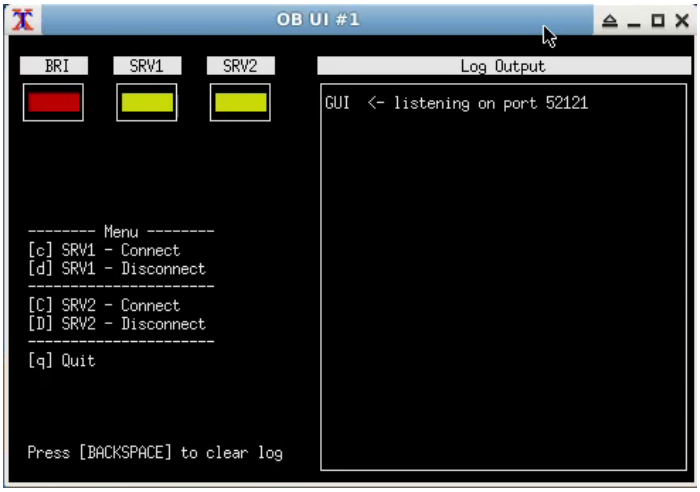
1



2

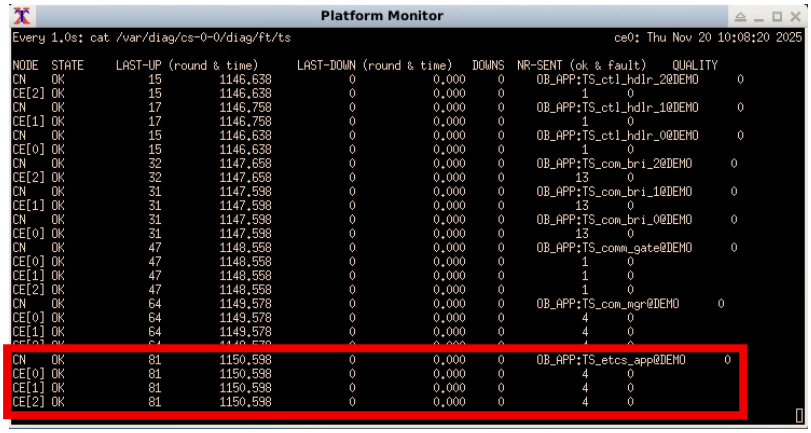
a) Trackside application running

b) Onboard UI started



c) Onboard application started – refer to Platform Monitor

d) Platform monitoring started – safe application Tasksets (etcs_app) running on all 3 computing elements.

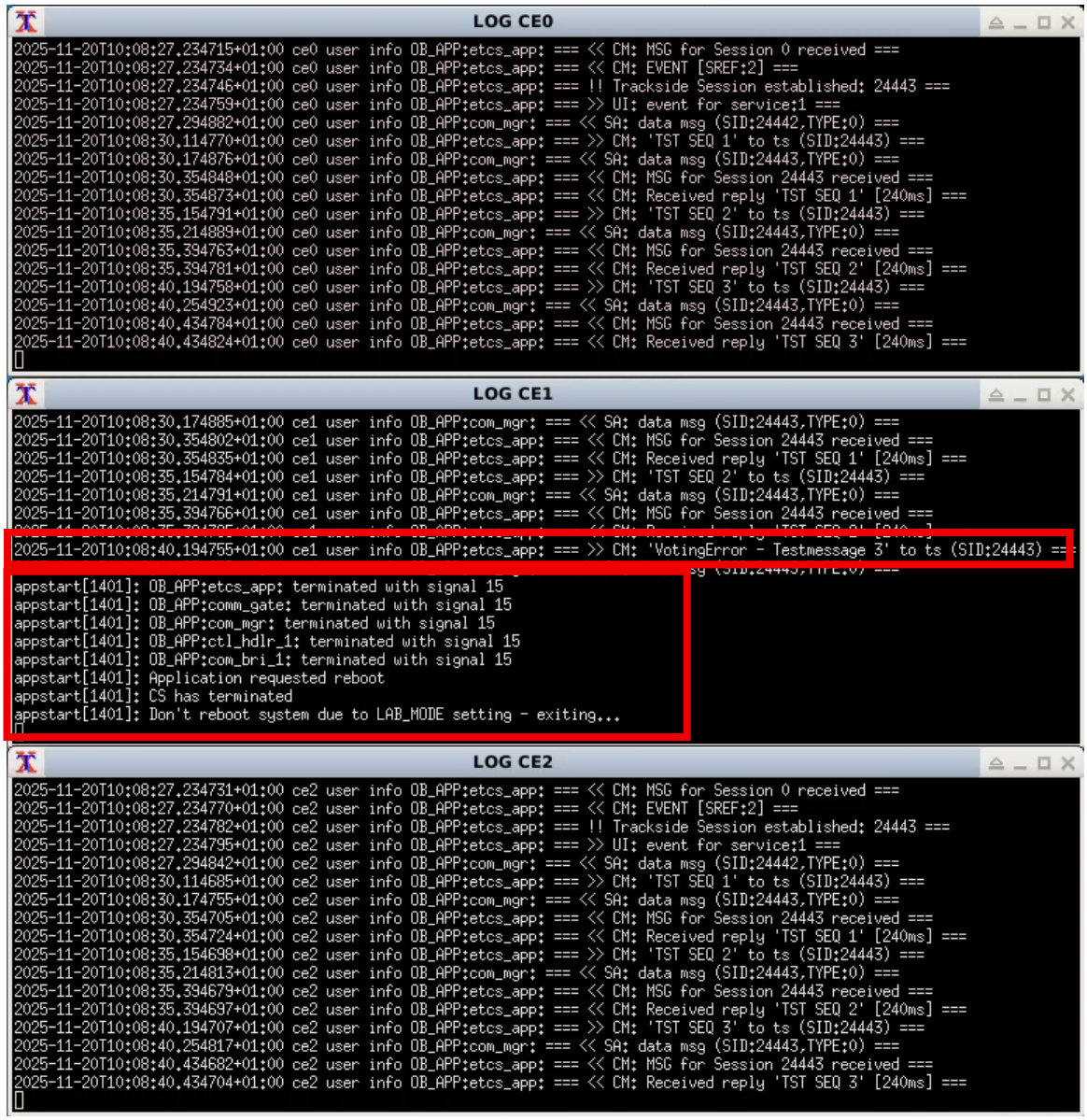


3

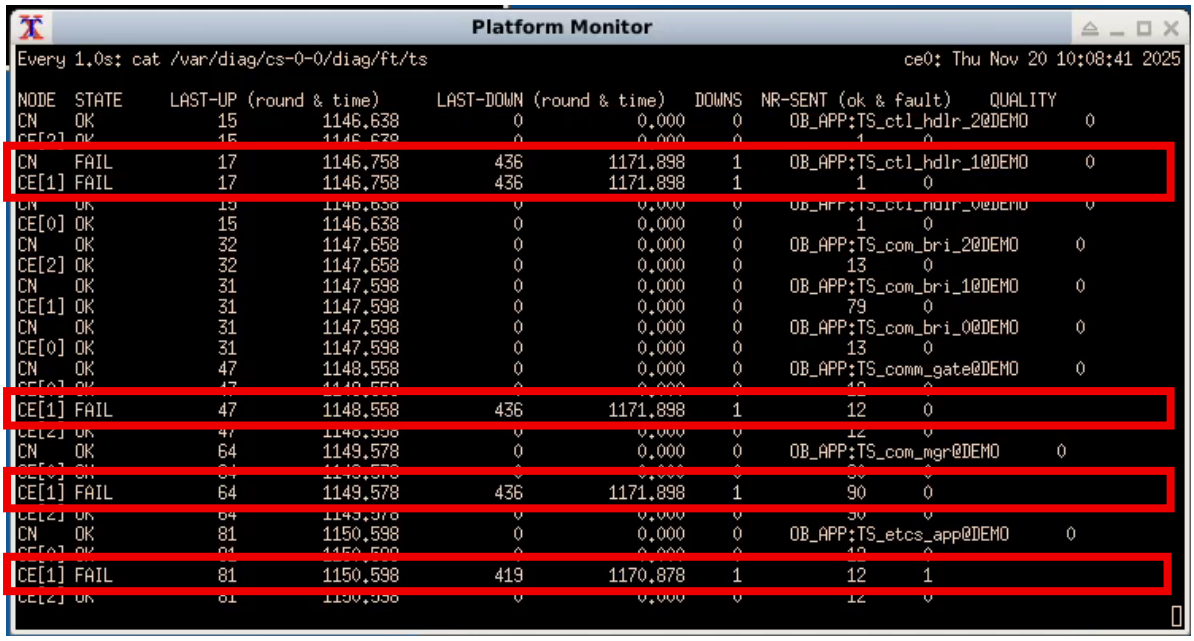
Connection established, 1st message exchanged

4

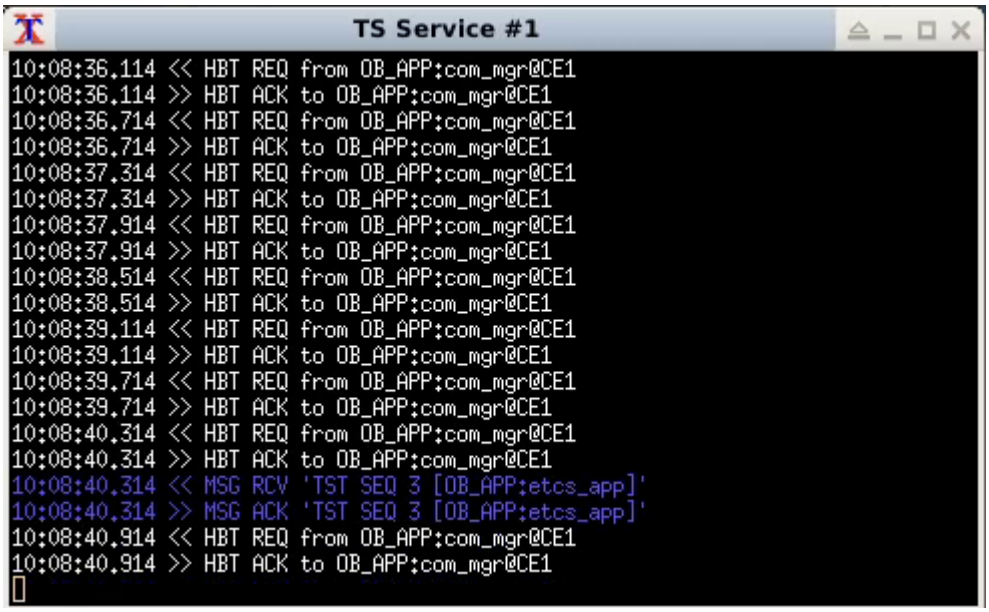
5	Replica #1 sends an invalid result to the voting system and is terminated



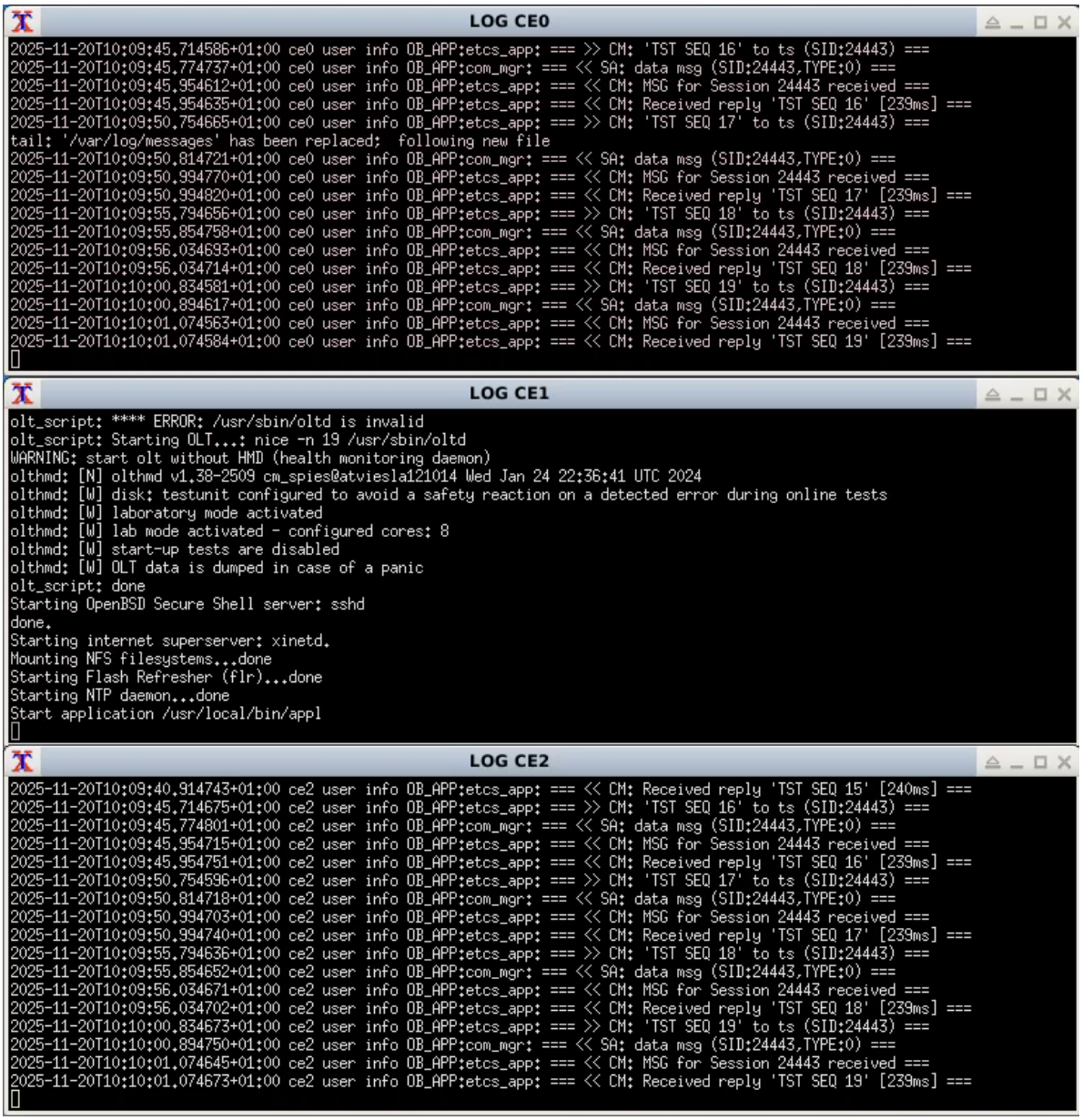
All tasksets on replica #1 are terminated



6 Message #3 successfully exchanged between onboard and trackside application.

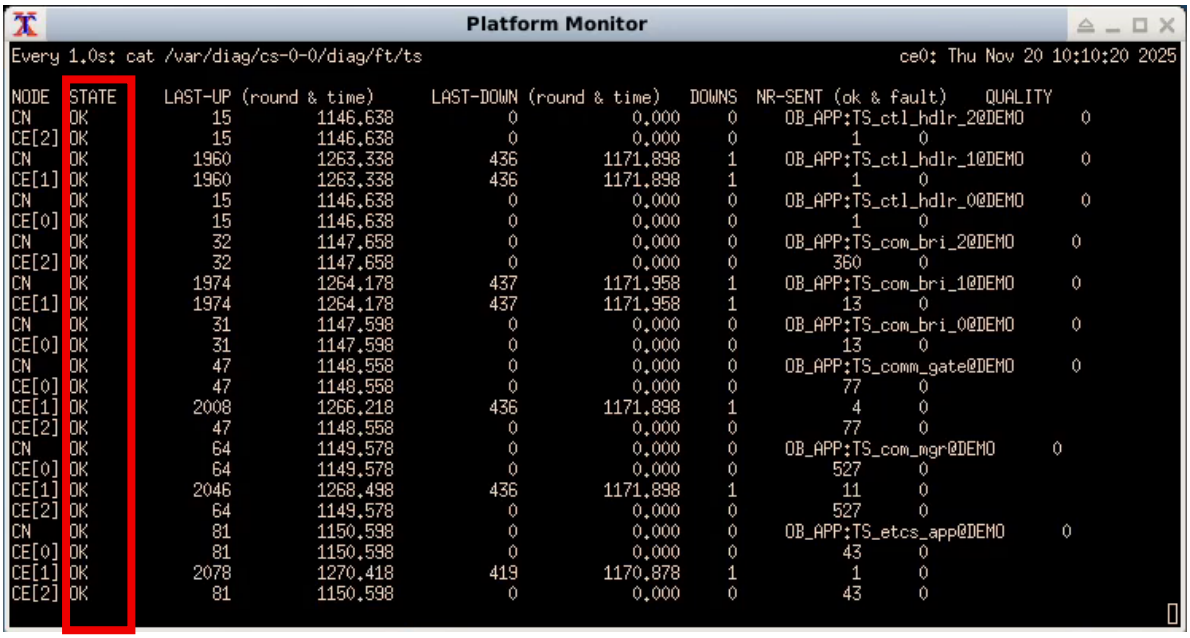


7 a) Replica #1 is rebooted



b) Message exchange between onboard and trackside application continued while ce1 restarted

8 Replica #1 successfully re-integrated and the system is running again in a 2oo3 configuration.

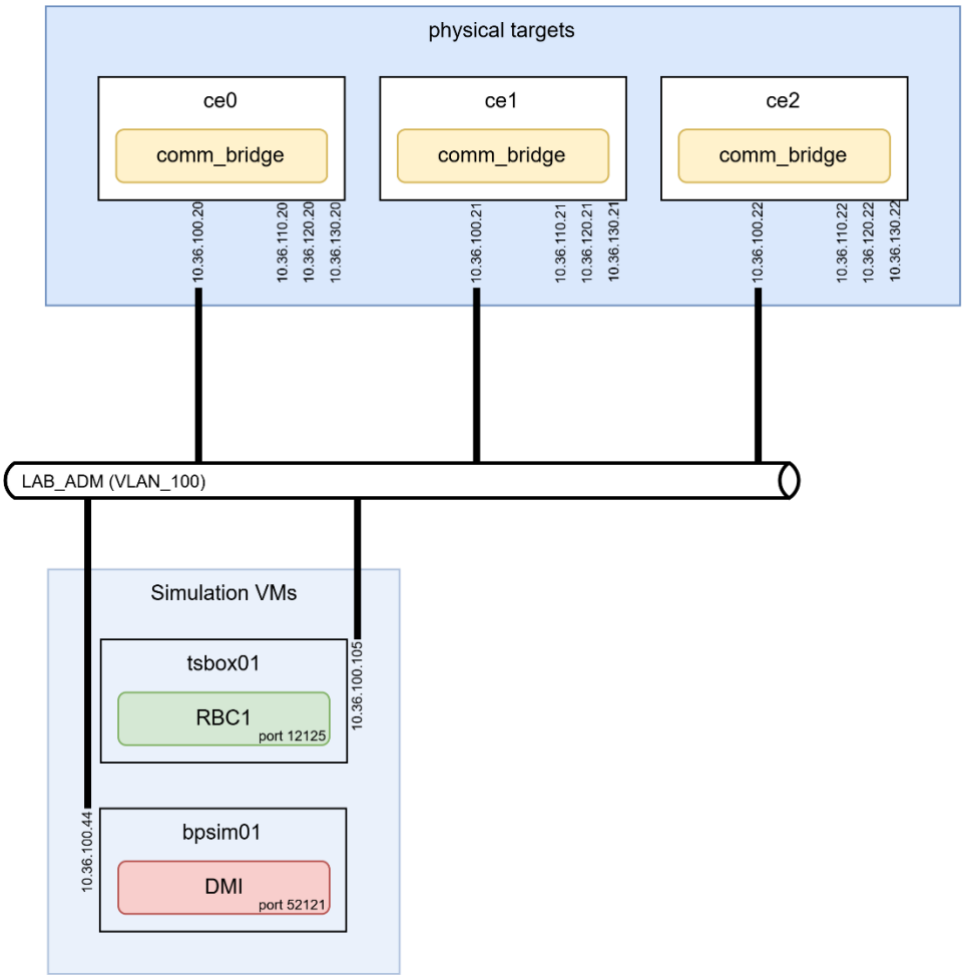


Status	Pass
Comments	…
Attachments	

3.4 REACT TO HARDWARE FAILURE [2.1.7]

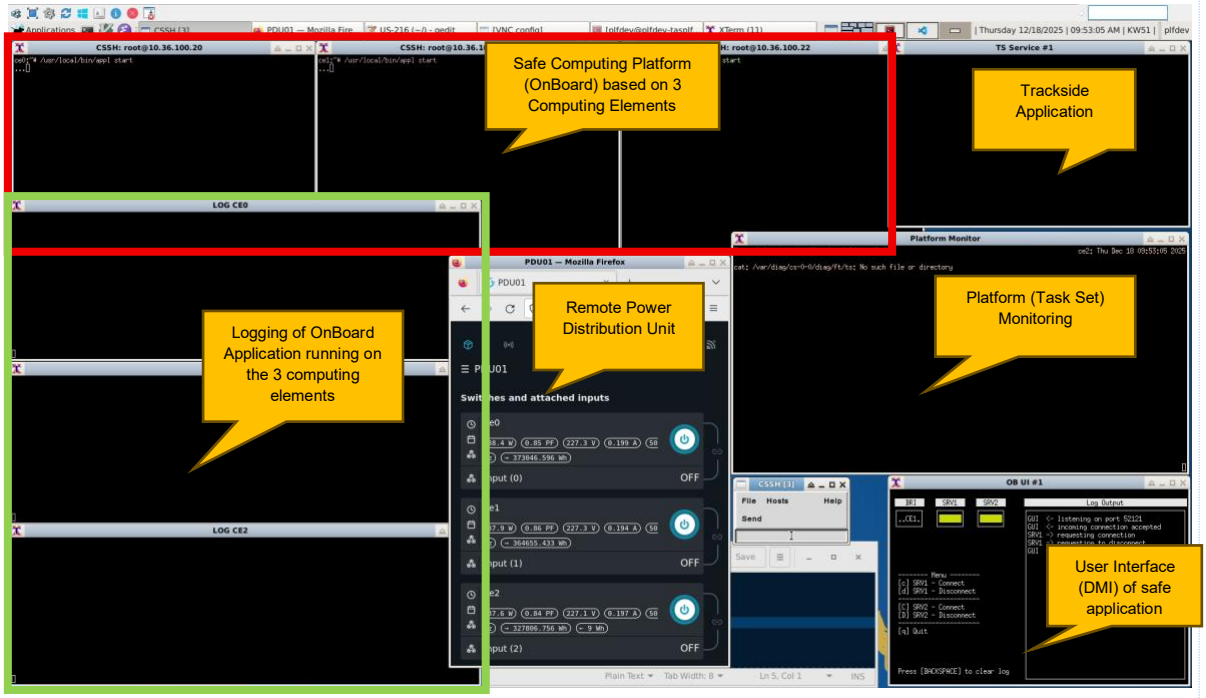
Description User Story	As R2DATO WP36, we want the safety layer / runtime environment to detect consistently failing replicas of safe applications (up to SIL 4), so that we can demonstrate the detection and reaction of persistent (reoccurring) hardware failure.
Annotation	The TAS Platform has a sophisticated built-in detection if a CE can no longer be used for safe computation due to it being no longer reliable. For efficient testing this mechanism was disabled. Hardware faults were simulated using software induced faults (see previous test case) and powering off the hardware boards (this test case).

Test Case Title	**Re-integrate one CE after simulated external failure**
Test Case ID	2.1.7-A
Description	While the system is running normally an external failure for one CE will be applied (e.g. power switch off, restart CE from console). Thus, the failed CE shall be restarted automatically and be re-integrated into the communication/voting system of the TAS Platform and continue normal operation.
Test Environment	For this user story, the architecture blueprint was configured to connect directly to the trackside application without using the FRMCS communication setup. Replica #1 (ce1) was configured to simulate a computation failure resulting in a voting error on the platform.

	
	While the system operates normally, one computing element is powered off to simulate a hardware failure.
Preconditions	Set up Test environment in a manner that runs long enough for this Test Case.
Test Steps	1. Prepare testbench 2. Start testbench components and monitoring a. Start trackside application b. Start onboard user interfaces (DMI) c. Start the onboard safe application (architecture blueprint) d. Start platform monitor 3. Initiate connections from onboard application to the trackside service. 4. Verify message exchange between onboard applications and trackside applications. 5. Remove power from ce1 (computing element 1) 6. Application continues to operate in a 2oo2 configuration until replica #1 I re-integrated.

	7. Apply power to ce1. 8. Monitor re-integration of restarted replica.
Test Data	None.
Expected Result	During the full test, the system must continue to perform its intended functions, without any detectable issues or degradation.

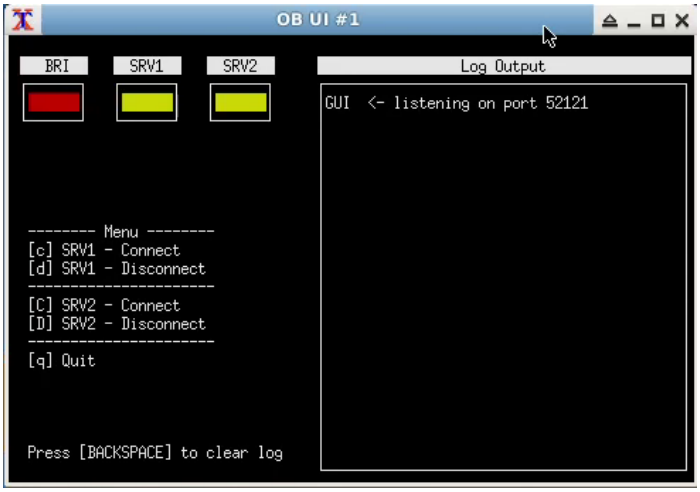
Actual Result	
1	Testbench setup completed



2

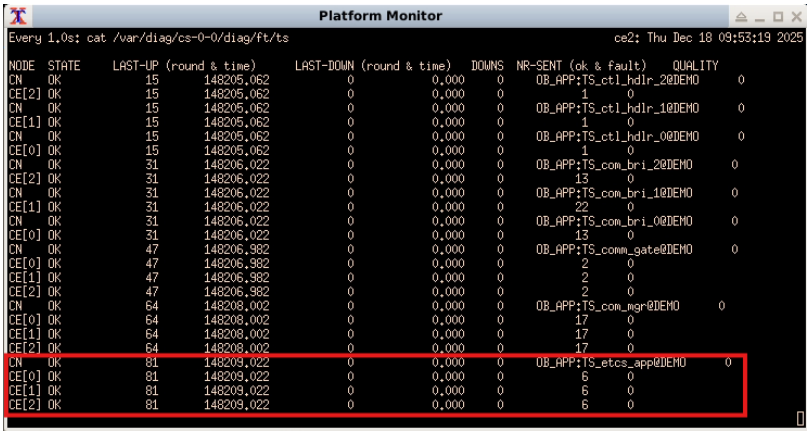
a) Trackside application running

b) Onboard UI started



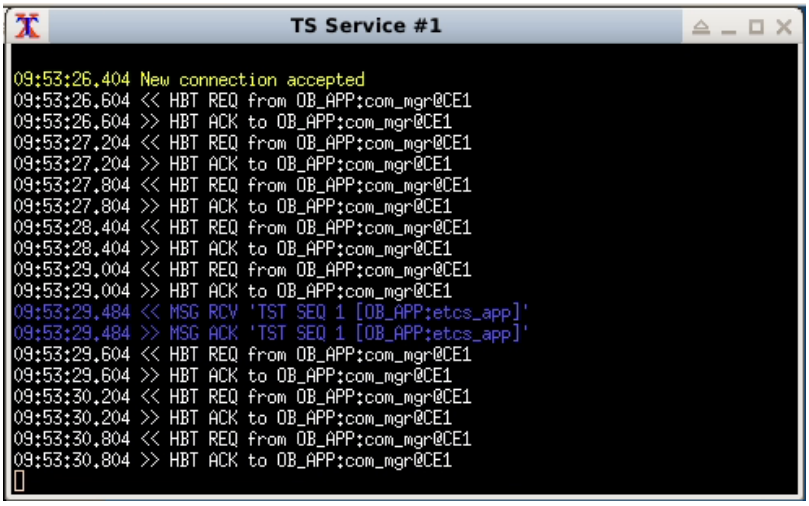
c) Onboard application started – refer to Platform Monitor

d) Platform monitoring started – safe application Tasksets (etcs_app) running on all 3 computing elements.

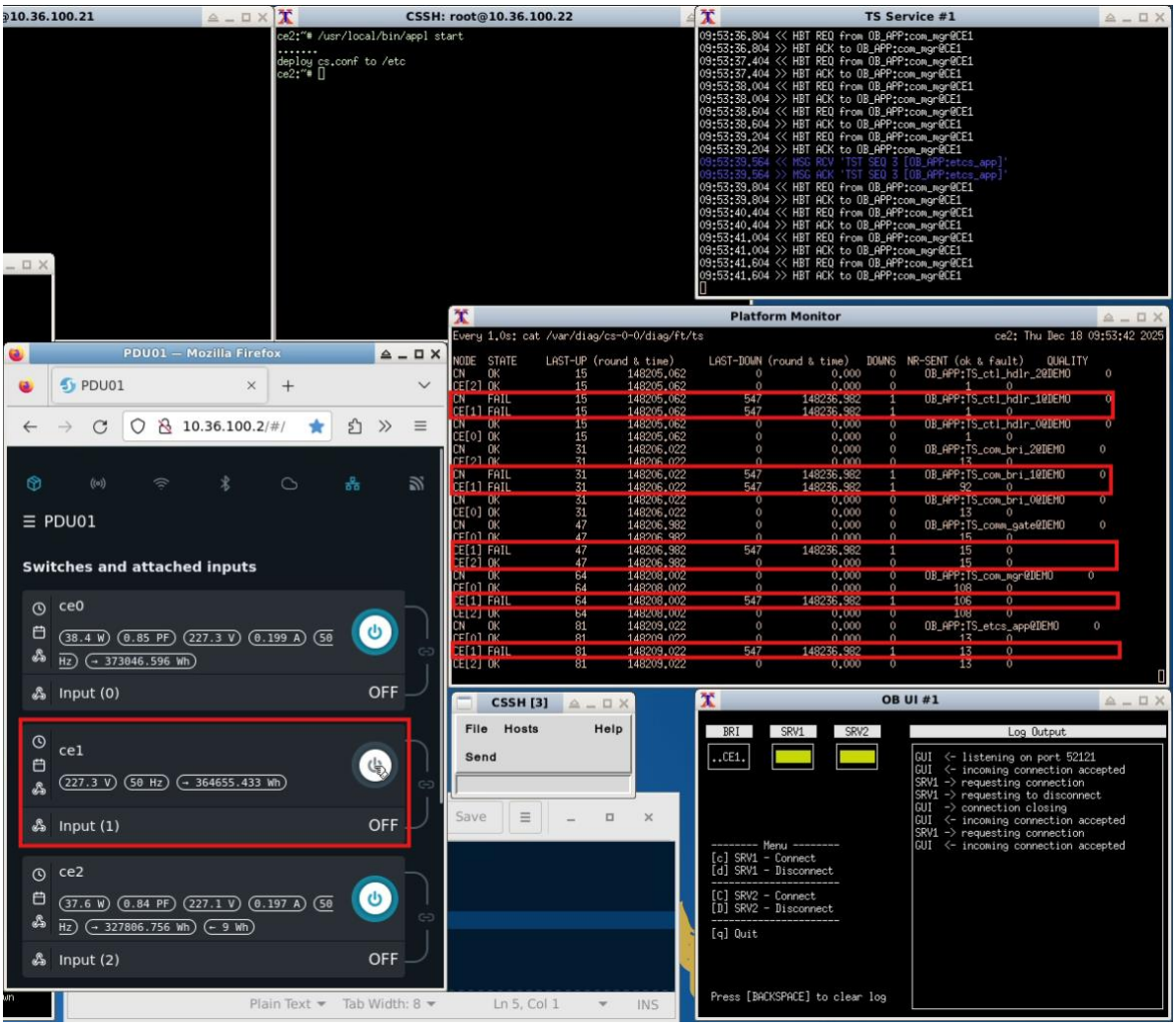


3 Connection established, 1st message exchanged

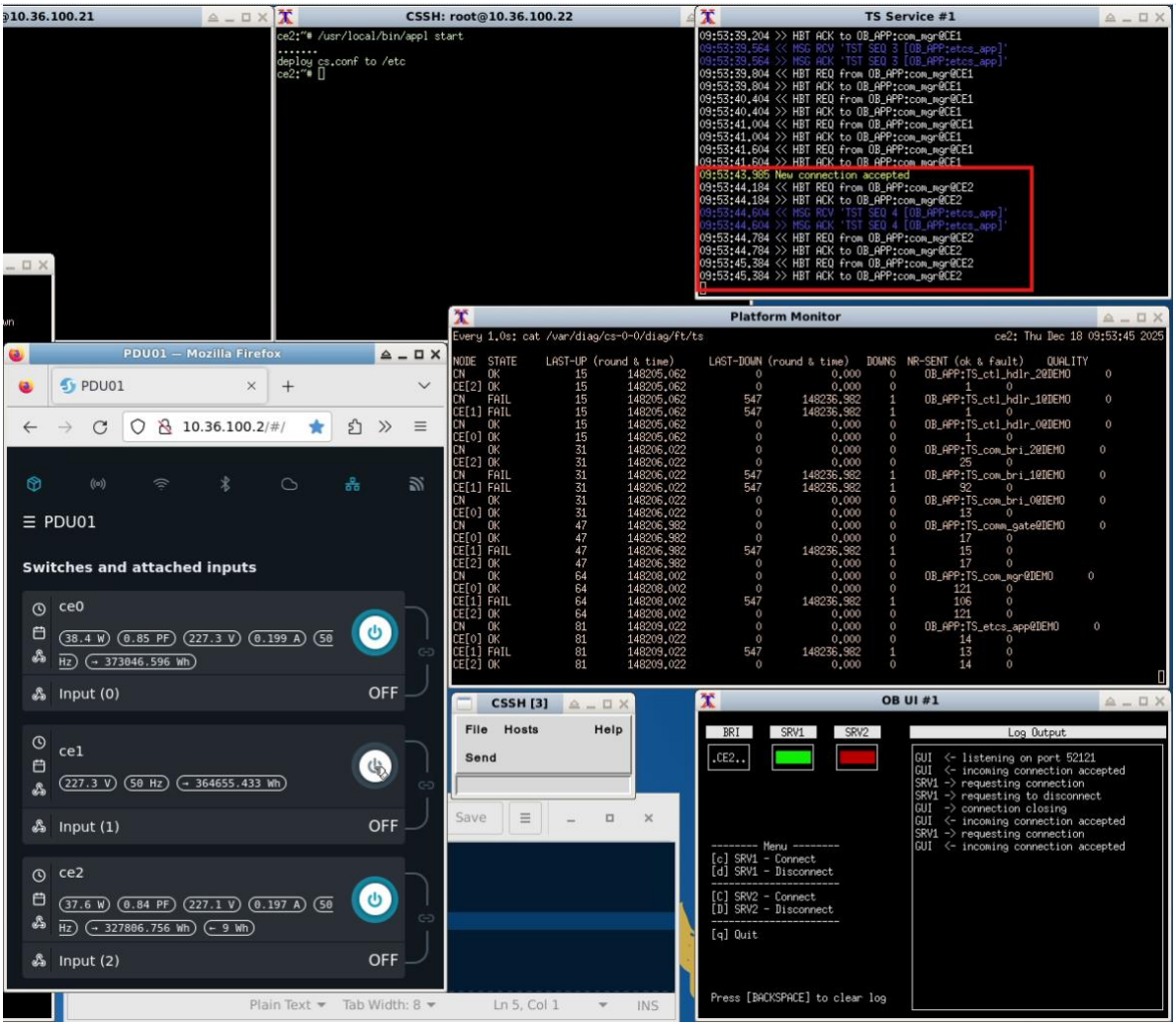
4



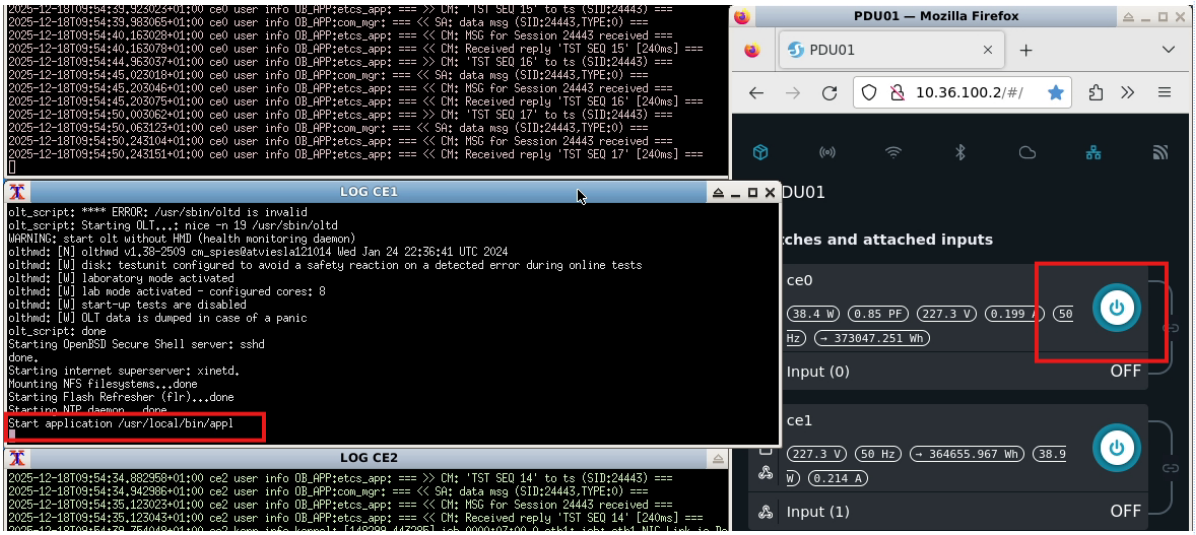
5	Power is removed from Replica #1 (ce1) and the Platform monitor reports the failing replica #1 (ce1)

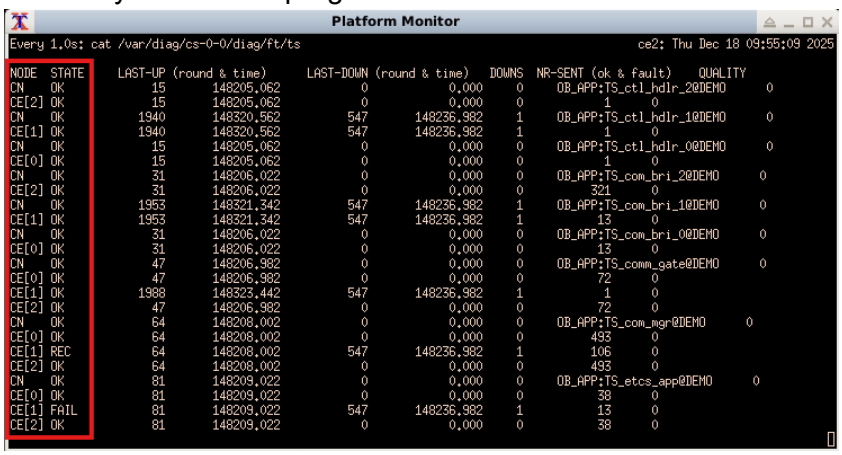
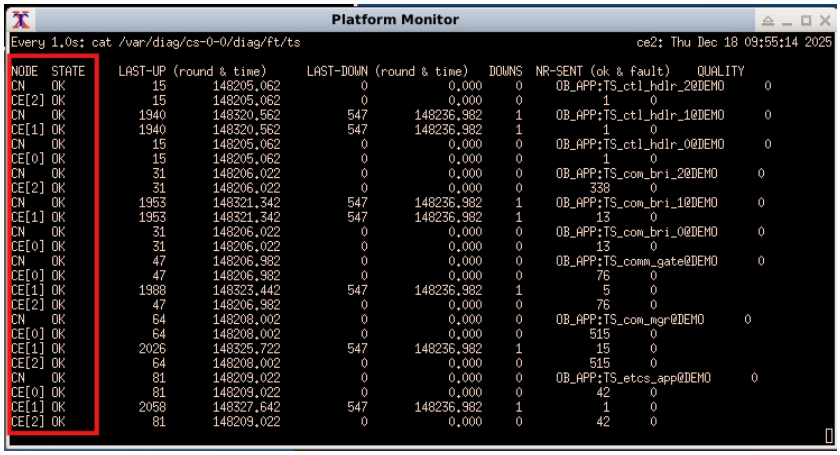


6	Replica #2 takes over the gateway functionality of replica #1, reconnects to the trackside application and the system continues to exchange messages between the onboard and the trackside applications.



7 Power is re-applied, Replica #1 automatically start the application



8	a) Recovery of ce1 is in progress  b) Recovery completed 
Status	Pass
Comments	…
Attachments	

3.5 FAILOVER TO COLD BACKUP HARDWARE [2.1.8]

Description User Story	As R2DATO WP36, we want the safety layer / runtime environment to shift load to additional backup hardware, so that we can demonstrate the short-term mitigation of persistent (reoccurring) hardware failure.
Annotation	Omitted: The User Story exceeds the capabilities of the lab demonstrator as there is no spare hardware available.

3.6 Change to Different Hardware [2.2.1]

Description User Story	As R2DATO WP36, we want the safety layer / runtime environment to integrate replaced hardware of a different kind, so that we can demonstrate the upgrade of obsolete hardware.
Annotation	Omitted: The User Story exceeds the capabilities of the lab demonstrator as there is no spare hardware available.

3.7 Change to Different Platform or Safety Layer / RTE [2.2.2]

Description User Story	As R2DATO WP36, we want to exchange the platform or safety layer / runtime environment (RTE), so that we can demonstrate the portability of existing applications.
Annotation	Omitted: The User Story exceeds the capabilities of the lab demonstrator as the used safety/runtime layers are proprietary solutions.

3.8 Use Diverse Hardware [2.2.3]

Description User Story	As R2DATO WP36, we want the safety layer / runtime environment to run on diverse hardware, so that we can demonstrate future upgrade / migration scenarios.
Annotation	Omitted: The User Story exceeds the capabilities of the lab demonstrator as only one type of hardware module is available to the project.

3.9 COMMUNICATE SAFE APPLICATION ON RTE <-> EXTERNAL SAFE APPLICATION [2.3.2]

Description User Story	As R2DATO WP36, we want a platform native safe application (up to SIL 4) on the safety layer / runtime environment to communicate with an external safe application (up to SIL 4), so that we can demonstrate safe communication via the paradigm of flows that leaves the modular computing platform via a communication gate.
Annotation	Omitted: No demonstration was carried out for this User Story. The demonstrator does not implement a production-grade safety protocol (EuroRadio); consequently, safe communication with an external system could not be demonstrated. The proposed architecture for safe external communication (see chapter 2.1.1 in [3]) specifies the components involved and allocates responsibility for the safety protocol to a dedicated Taskset. Concretely, the Communication Manager (model1 Taskset) is assigned responsibility for implementing the protocol. In the presented architecture, each safety-critical application contains its own Communication Manager Taskset; therefore, the external-communication safety protocol is considered part of the safe application.

3.10 COMMUNICATE PARALLEL BASIC INTEGRITY APP <-> EXTERNAL BASIC INTEGRITY APP [2.3.9]

Description User Story	As R2DATO WP36, we want a parallel basic integrity application on the modular computing platform hardware pool to communicate with an external basic integrity application, so that we can demonstrate a generic communication that leaves the modular computing platform.
Annotation	Covered with 3.17 Collect Diagnostics Data from Safe Application on RTE [2.5.3]

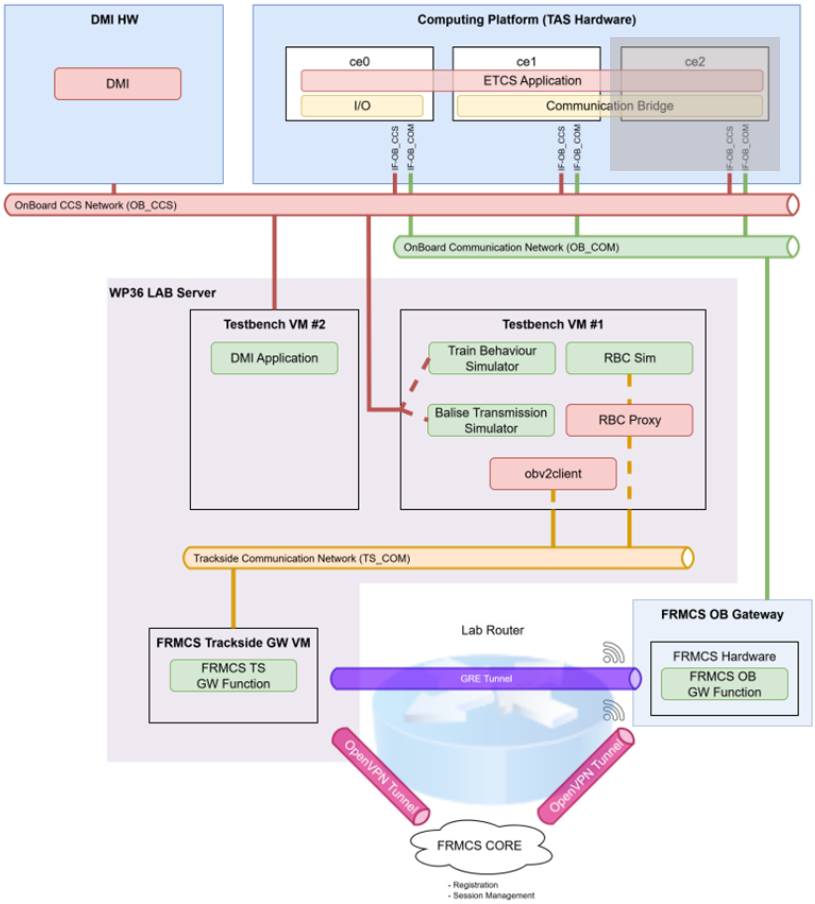
3.11 COMMUNICATE INDEPENDENT OF RTE INSTANCE [2.3.10]

Description User Story	As R2DATO WP36, we want to deploy two communicating applications – unchanged – either on the same or on different instances of the safety layer / runtime environment, so that we can demonstrate the location transparency of the communication model.
Annotation	Omitted: The User Story is not demonstrated because location transparency in internal communication cannot be shown in our setup.

3.12 COMMUNICATE INDEPENDENT OF REPLICATION [2.3.11]

Description User Story	As R2DATO WP36, we want to deploy two communicating applications – unchanged – with different replication configuration on the safety layer / runtime environment, so that we can demonstrate the replication transparency of the communication model.

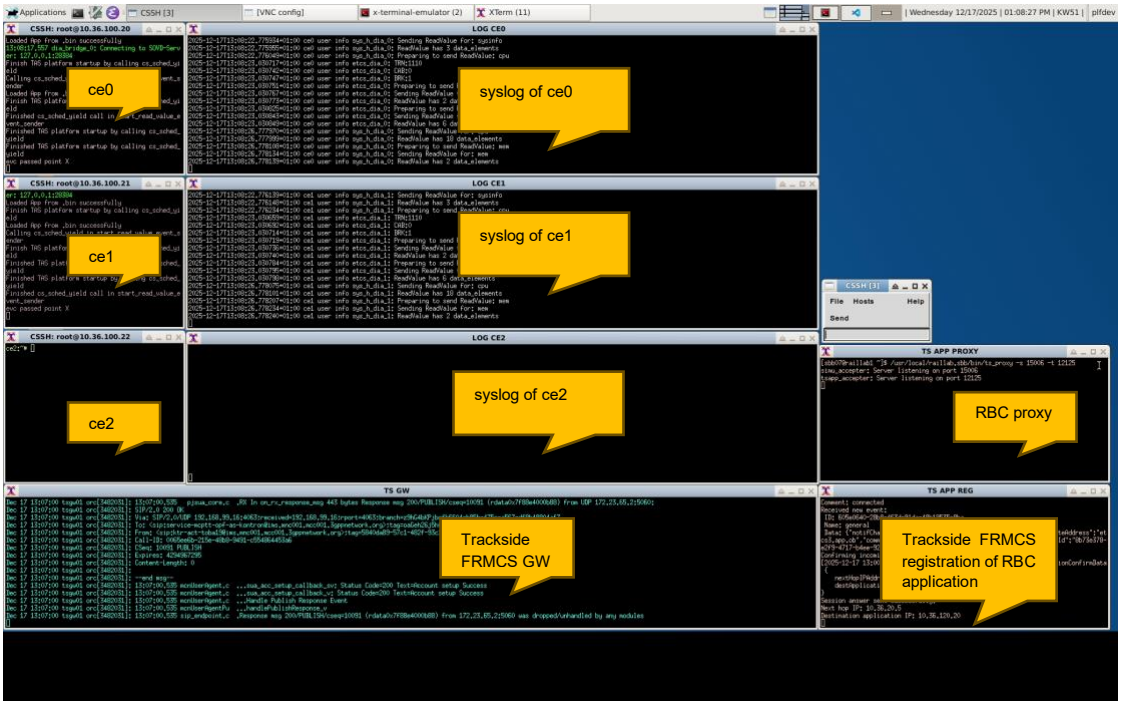
Test Case Title	**Run unchanged application in 2oo2 configuration**
Test Case ID	2.3.11-A
Description	The normally used (i.e. the 2oo3 configuration) application software shall be used. Only the configuration will be changed from 2oo3 to 2oo2 (no other changes) for this test. The system should run functionally identically as with the 2oo3 configuration.

Test Environment	• DMI Application running on a VM • Modular Platform (TAS Platform) • CCS Communication Network (CCN) • Testbench VMs running on the Lab Server • FRMCS trackside Gateway on a VM • Physical FRMCS onboard Gateway • ETCS Application running on the Modular Platform  For this user story, we use the same setup as in Test Case 2.10.2-A for User Story [2.10.2] and adapt only the configuration to run in a 2oo2 setup.
Preconditions	Set up Test environment in a manner that runs long enough for this Test Case.

Test Steps	1. Prepare testbench (FRMCS and DMI) a. Start FRMCS onboard gateway b. Start FRMCS trackside gateway c. Register trackside application to trackside GW d. Start DMI Application 2. Start testbench components and monitoring a. Start the onboard safe application (ETCS) b. Start platform monitor c. Start the testbench simulators using the scenario controller 3. Open the train driver cab 4. Start mission on DMI 5. Drive train 6. Stop train 7. Close cabin
Test Data	None.
Expected Result	During the full test, the system must continue to perform its intended functions, without any detectable issues or degradation.

Actual Result	
1	a) FRMCS onboard gateway is running and connected to the FRMCS Core b) FRMCS trackside gateway is running and connected to the FRMCS Core c) The trackside application “rbc3.app.ts” is registered to the FRMCS Core d) The DMI Applications is running and ready to accept connections from the ETCS application

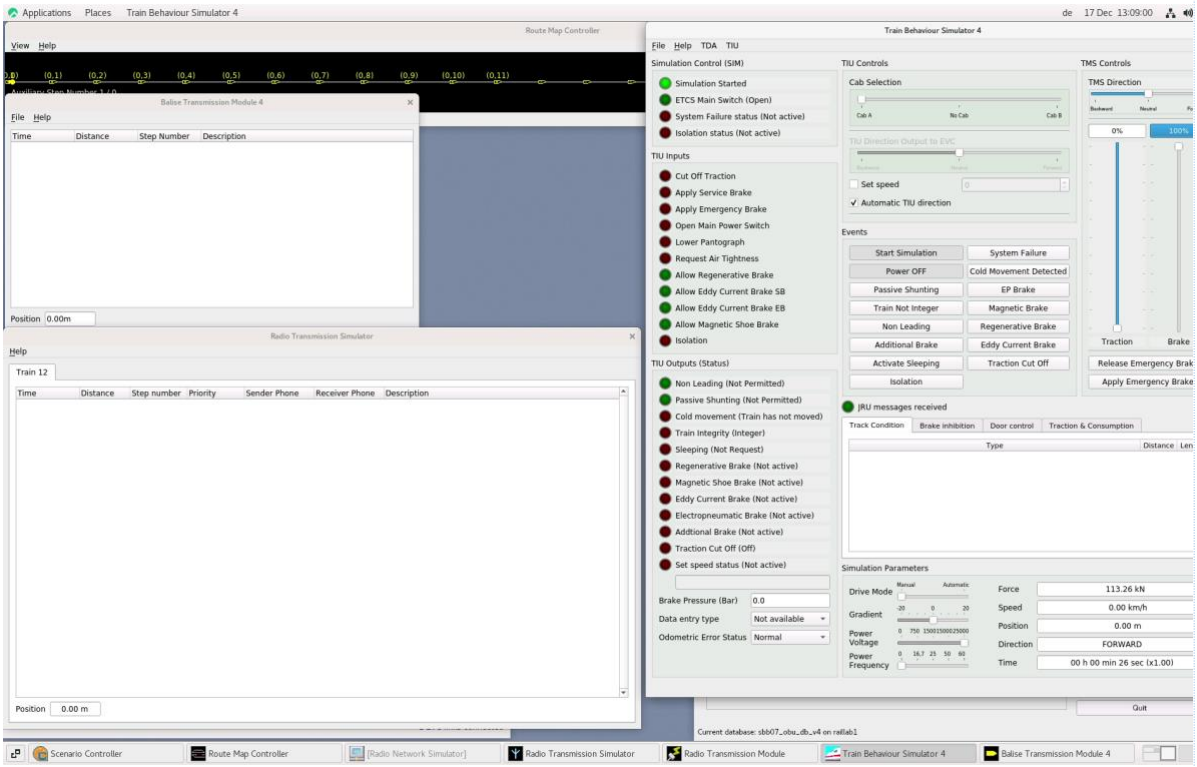
2 a) The ETCS application is started only on ce0 and ce1. Next to each console of the individual computing elements (ce), the syslog output is shown.



b) The platform monitor provides the status of all Tasksets running on the different computing elements. In this setup the (safe) Model 1 Tasksets are running in a 2oo2 setup on the two separate computing elements ce0 and ce1.

```
Every 1.0s: cat /var/diag/cs-0-0/diag/ft/ts                          ce0: Wed Dec 17 13:08:23 2025

NODE  STATE     LAST-UP (round & time)      LAST-DOWN (round & time)   DOWNS  NR-SENT (ok & fault)    QUALITY
CN    OK              11          73512.478          0          0.000      0     TS_etcs_dia_1@DEMO      0
CE[1] OK              11          73512.478          0          0.000      0           0      0
CN    OK              11          73512.478          0          0.000      0     TS_etcs_dia_0@DEMO      0
CE[0] OK              11          73512.478          0          0.000      0           0      0
CN    OK              11          73512.478          0          0.000      0     TS_sys_h_dia_1@DEMO      0
CE[1] OK              11          73512.478          0          0.000      0           0      0
CN    OK              11          73512.478          0          0.000      0     TS_sys_h_dia_0@DEMO      0
CE[0] OK              11          73512.478          0          0.000      0           0      0
CN    OK              11          73512.478          0          0.000      0     TS_dia_bridge_1@DEMO      0
CE[1] OK              11          73512.478          0          0.000      0           1      0
CN    OK              11          73512.478          0          0.000      0     TS_dia_bridge_0@DEMO      0
CE[0] OK              11          73512.478          0          0.000      0           1      0
CN    OK              11          73512.478          0          0.000      0     TS_ctl_hdlr_1_1@DEMO      0
CE[1] OK              11          73512.478          0          0.000      0           1      0
CN    OK              11          73512.478          0          0.000      0     TS_ctl_hdlr_1_0@DEMO      0
CE[0] OK              11          73512.478          0          0.000      0           1      0
CN    OK              11          73512.478          0          0.000      0     TS_com_bri_1_1@DEMO      0
CE[1] OK              11          73512.478          0          0.000      0          13      0
CN    OK              11          73512.478          0          0.000      0     TS_com_bri_1_0@DEMO      0
CE[0] OK              11          73512.478          0          0.000      0          13      0
CN    OK              11          73512.478          0          0.000      0     TS_comm_gate_1@DEMO      0
CE[0] OK              11          73512.478          0          0.000      0           1      0
CE[1] OK              11          73512.478          0          0.000      0           1      0
CN    OK              11          73512.478          0          0.000      0     TS_com_mgr_1@DEMO      0
CE[0] OK              11          73512.478          0          0.000      0           4      0
CE[1] OK              11          73512.478          0          0.000      0           4      0
CN    OK              11          73512.478          0          0.000      0     io@DEMO      0
CE[0] OK              11          73512.478          0          0.000      0          31      0
CN    OK              10          73512.368          0          0.000      0     xfer@DEMO      0
CE[0] OK              10          73512.368          0          0.000      0          10      0
CE[1] OK              10          73512.368          0          0.000      0          10      0
CN    OK              13          73512.698          0          0.000      0     evc@DEMO      0
CE[0] OK              13          73512.698          0          0.000      0          84      0
CE[1] OK              13          73512.698          0          0.000      0          84      0
```

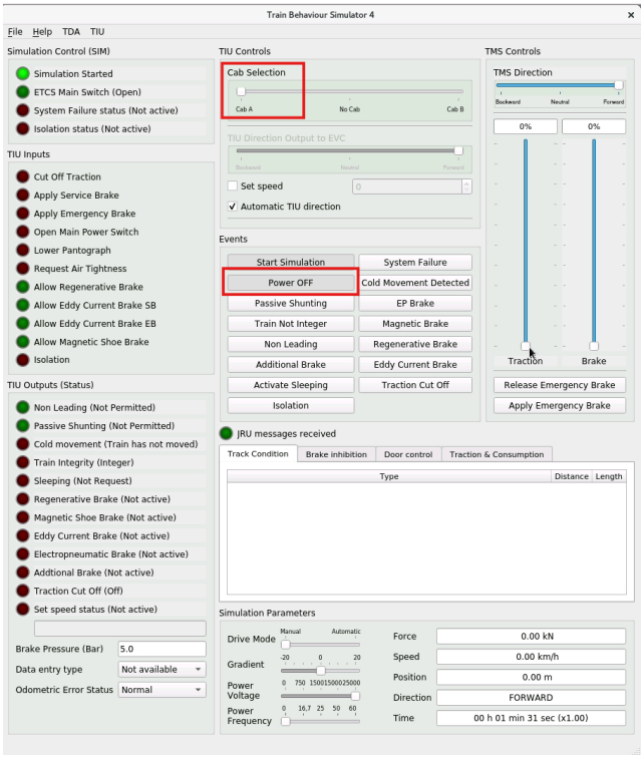
c) The Test Testbench is running with all required applications



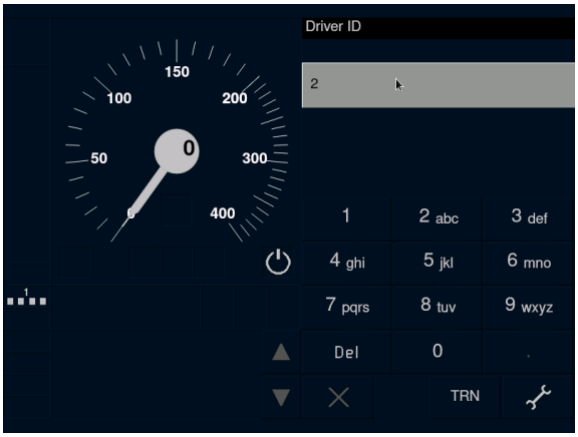
- The Route Map Controller, showing the balise positions on the track and the current position of the train.
- The Balise Transition Module, sending balise telegrams to the ETCS application, when the train reaches the position of a balise
- The Radio Transmission Simulator simulates the RBC. It is triggered by messages received from the onboard unit and provides the corresponding answers to the ETCS application running on the modular platform.

The Train Behaviour Simulator, is used to manage the active cabin, to control the traction and brakes as well as other train related signals and indications

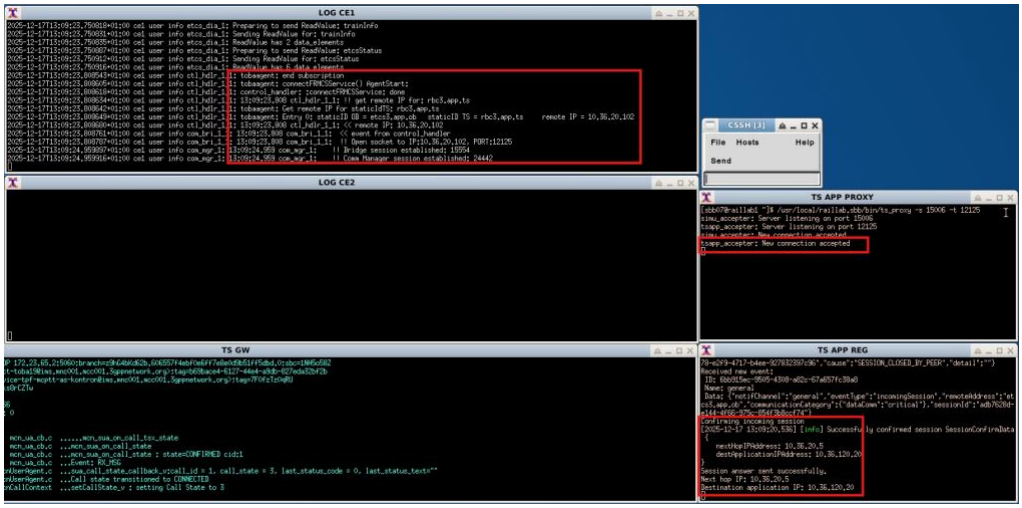
3	Open the train driver CAB



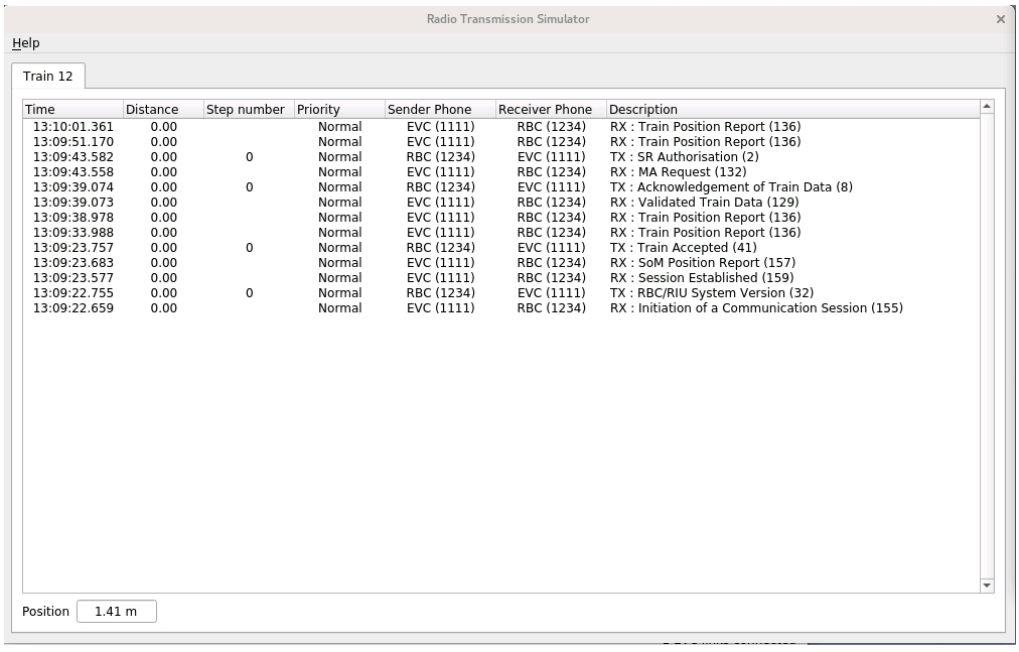
The DMI is activated, prompting for driver ID and Train Data.



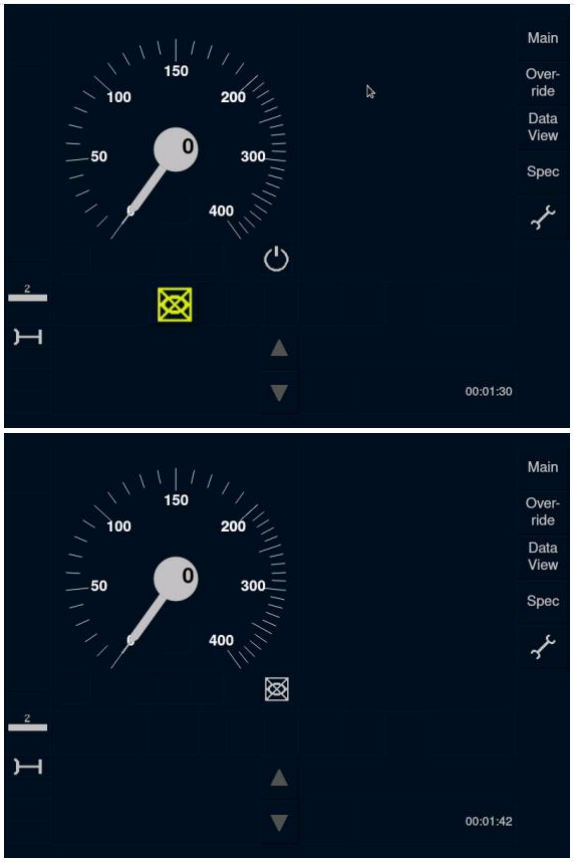
4	Start mission on DMI After activating the cabin, the ETCS Application prompts for required train data. • Driver ID • RBC data – after entering the RBC data, the ETCS Application establishes a connection to the RBC using FRMCS.

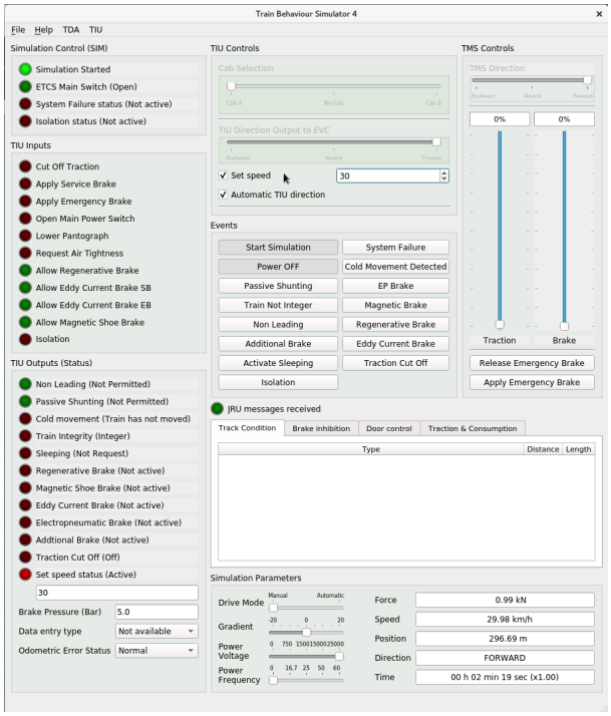


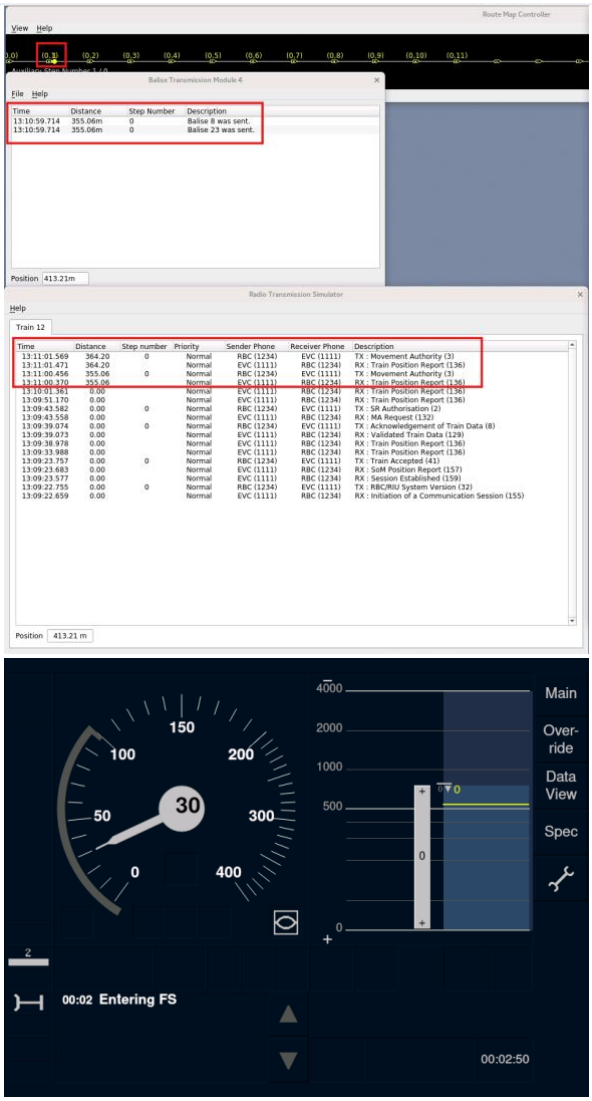
- The connection to the RBC is established and the ETCS application requests the driver to acknowledge SR (Staff Responsible) mode.

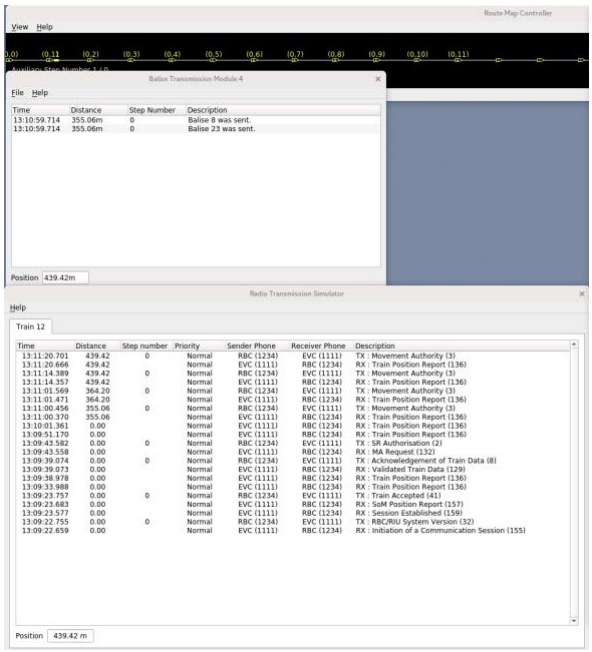
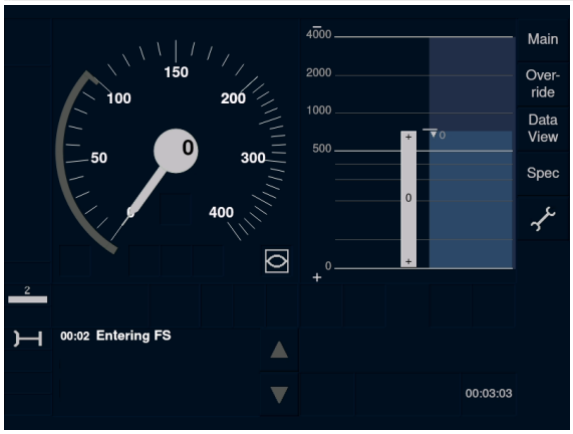


Time	Distance	Step number	Priority	Sender Phone	Receiver Phone	Description
13:10:01.361	0.00		Normal	EVC (1111)	RBC (1234)	RX : Train Position Report (136)
13:09:51.170	0.00		Normal	EVC (1111)	RBC (1234)	RX : Train Position Report (136)
13:09:43.582	0.00	0	Normal	RBC (1234)	EVC (1111)	TX : SR Authorisation (2)
13:09:43.558	0.00		Normal	EVC (1111)	RBC (1234)	RX : MA Request (132)
13:09:39.074	0.00	0	Normal	RBC (1234)	EVC (1111)	TX : Acknowledgement of Train Data (8)
13:09:39.073	0.00		Normal	EVC (1111)	RBC (1234)	RX : Validated Train Data (129)
13:09:38.978	0.00		Normal	EVC (1111)	RBC (1234)	RX : Train Position Report (136)
13:09:33.988	0.00		Normal	EVC (1111)	RBC (1234)	RX : Train Position Report (136)
13:09:23.757	0.00	0	Normal	RBC (1234)	EVC (1111)	TX : Train Accepted (41)
13:09:23.683	0.00		Normal	EVC (1111)	RBC (1234)	RX : SoM Position Report (157)
13:09:23.577	0.00		Normal	EVC (1111)	RBC (1234)	RX : Session Established (159)
13:09:22.755	0.00	0	Normal	RBC (1234)	EVC (1111)	TX : RBC/RIU System Version (32)
13:09:22.659	0.00		Normal	EVC (1111)	RBC (1234)	RX : Initiation of a Communication Session (155)


Main
Over-
ride
Data
View
Spec
150
100
200
50
0
300
400
00:01:30
Main
Over-
ride
Data
View
Spec
150
100
200
50
0
300
400
00:01:42

5	Drive train • Release emergency brake, release brake and apply traction   The train passes the 1st Balise and switches to FS (Full Supervision) mode

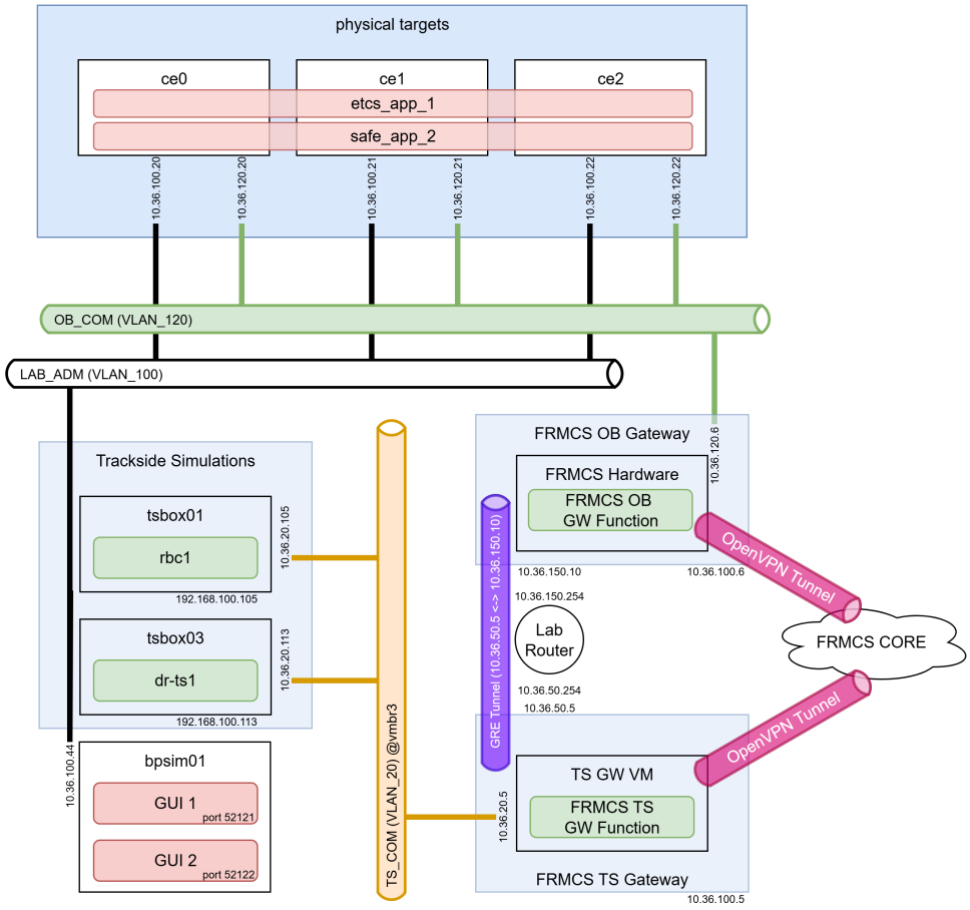

Route Map Controller
View Help
Balise Transmission Module 4
File Help
Time Distance Step Number Description
13:10:59.714 355.06m 0 Balise 8 was sent.
13:10:59.714 355.06m 0 Balise 23 was sent.
Position 413.21m
Radio Transmission Simulator
Help
Train 12
Time Distance Step number Priority Sender Phone Receiver Phone Description
Position 413.21 m
150
100
200
50
30
300
0
400
4000
2000
1000
500
0
Main
Over-ride
Data View
Spec
00:02 Entering FS
00:02:50

6	Stop train   After reaching FS mode, the train is stopped
Status	Pass
Comments	
Attachments	

3.13 REUSE FRMCS PLATFORM FUNCTIONS [2.4.2]

Description User Story	As R2DATO WP36, we want multiple applications on the modular computing platform (and applications on the safety layer / runtime environment) to communicate over FRMCS to a trackside entity, so that we can demonstrate the reuse of FRMCS platform functions.

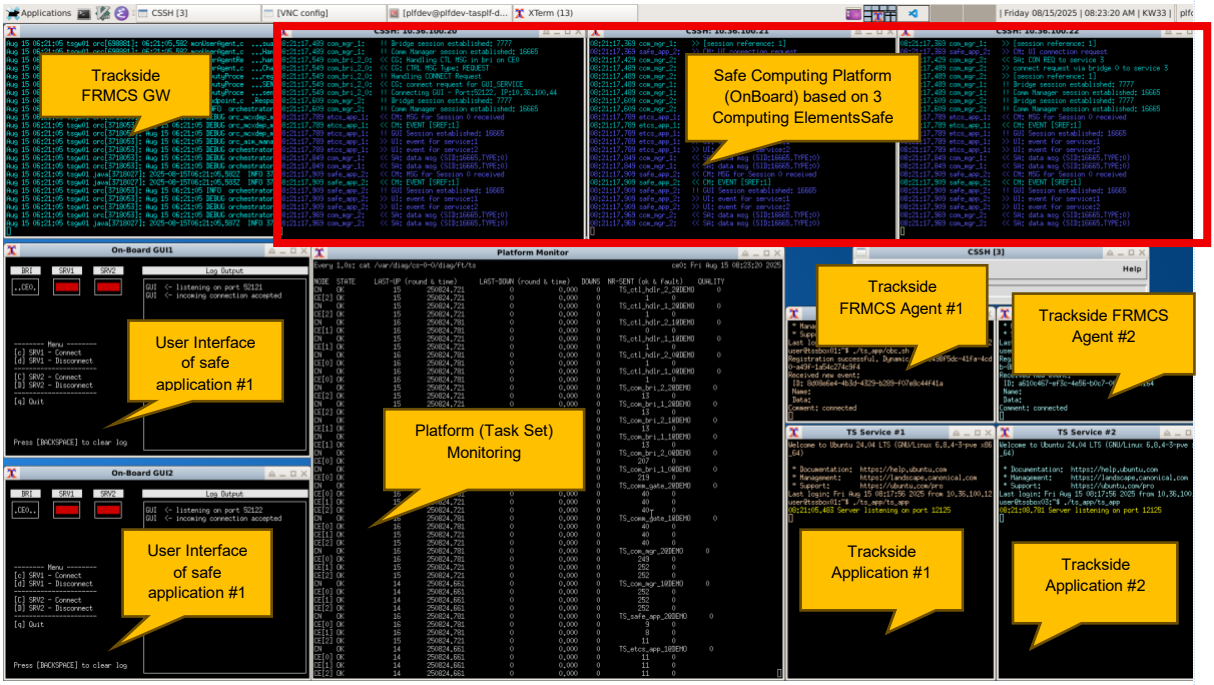
Test Case Title	**Run two safe applications in parallel using FRMCS**

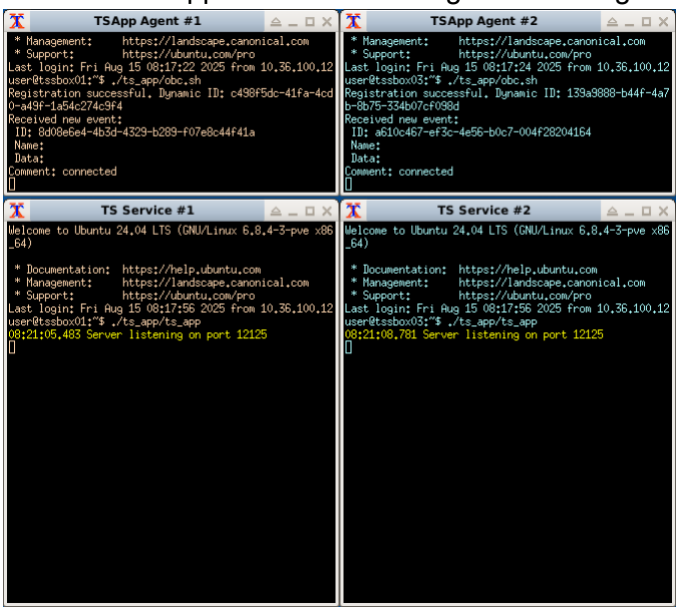
Test Case ID	2.4.2-A
Description	Run two safe applications in parallel on the TAS Platform with each using their own communication over FRMCS.
Test Environment	• Modular Platform (TAS Platform) • Physical FRMCS Onboard Gateway • FRMCS trackside Gateway on a VM • CCS Communication Network (CCN) • Two trackside application simulations running as VMs on the lab server • User interfaces for onboard applications • Two onboard applications (etcs_app_1 & safe_app_2)  For this user story, two instances of the architecture blueprint will be executed on the same Modular Platform. This was achieved without any code change, just by duplicating the Taskset configuration of the blueprint and adapting the startup script to start two instances of the safe application.
Preconditions	Set up Test environment in a manner that runs long enough for this Test Case.

Test Steps	1. Prepare testbench 2. Start testbench components and monitoring a. Start FMCS trackside GW logging b. Register trackside applications to trackside GW c. Start trackside applications d. Start onboard user interfaces e. Start the two onboard safe applications f. Start platform monitor 3. Initiate connections from each onboard application over FRMCS to its trackside counterpart. 4. Verify message exchange between onboard applications and trackside applications 5. Disconnect from trackside applications
Test Data	None.
Expected Result	During the full test, the system must continue to perform its intended functions, without any detectable issues or degradation.

Actual Result

1	All terminals ready
2	Testbench started

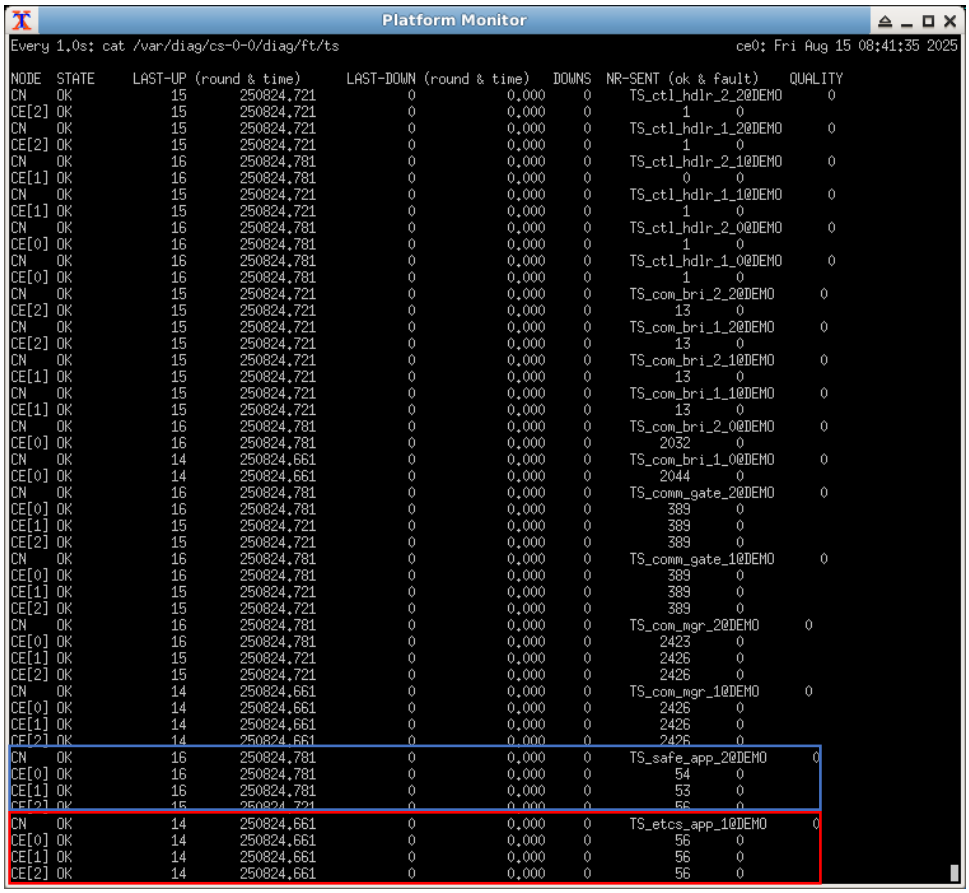


2a	Trackside FRMCS GW started.
2b	Trackside applications running and registered to trackside FRMCS GW.
2c	
2d	Onboard user interfaces ready 

2e Onboard applications ready and connected to their user interfaces (Safe Application #1, Safe Application #2)

```
ce0:~# /usr/local/erju/blueprint/bin/start242.sh
cleanup app 1
cleanup app 2
waiting up to 30 seconds for cs/ft to come up ...
Starting etcs_app_1 on all CEs
Starting com_mgr on all CEs
Starting comm_gate on all CEs
Starting com_bri_1_0 on ce0
Starting ctl_hdlr_1_0 on ce0
Starting safe_app_2 on all CEs
Starting com_mgr on all CEs
Starting comm_gate on all CEs
Starting com_bri_2_0 on ce0
Starting ctl_hdlr_2_0 on ce0
08:21:17.130 com_bri_1_0:  !! This is bridge com_bri_1_0 with number 0
08:21:17.192 com_bri_2_0:  !! This is bridge com_bri_2_0 with number 0
08:21:17.250 etcs_app_1:   >> CM: UI connection request
08:21:17.310 com_mgr_1:    << SA: CON REQ to service 3
08:21:17.369 com_mgr_1:    >> connect request via bridge 0 to service 3
08:21:17.369 com_mgr_1:    >> [session reference: 1]
08:21:17.369 safe_app_2:   >> CM: UI connection request
08:21:17.429 com_bri_1_0:  << CG: Handling CTL MSG in bri on CE0
08:21:17.429 com_bri_1_0:  << CG: CTRL MSG Type: REQUEST
08:21:17.429 com_bri_1_0:  !! Handling CONNECT Request
08:21:17.429 com_bri_1_0:  << CG: connect request for GUI_SERVICE
08:21:17.429 com_bri_1_0:  !! Connecting GUI - Port:52121, IP:10.36.100.44
08:21:17.429 com_mgr_2:    << SA: CON REQ to service 3
08:21:17.489 com_mgr_2:    >> connect request via bridge 0 to service 3
08:21:17.489 com_mgr_2:    >> [session reference: 1]
08:21:17.489 com_mgr_1:    !! Bridge session established: 7777
08:21:17.489 com_mgr_1:    !! Comm Manager session established: 16665
08:21:17.549 com_bri_2_0:  << CG: Handling CTL MSG in bri on CE0
08:21:17.549 com_bri_2_0:  << CG: CTRL MSG Type: REQUEST
08:21:17.549 com_bri_2_0:  !! Handling CONNECT Request
08:21:17.549 com_bri_2_0:  << CG: connect request for GUI_SERVICE
08:21:17.549 com_bri_2_0:  !! Connecting GUI - Port:52122, IP:10.36.100.44
08:21:17.609 com_mgr_2:    !! Bridge session established: 7777
08:21:17.609 com_mgr_2:    !! Comm Manager session established: 16665
08:21:17.789 etcs_app_1:   << CM: MSG for Session 0 received
08:21:17.789 etcs_app_1:   << CM: EVENT [SREF:1]
08:21:17.789 etcs_app_1:   !! GUI Session established: 16665
08:21:17.789 etcs_app_1:   >> UI: event for service:1
08:21:17.789 etcs_app_1:   >> UI: event for service:2
08:21:17.849 com_mgr_1:    << SA: data msg (SID:16665,TYPE:0)
08:21:17.849 com_mgr_1:    << SA: data msg (SID:16665,TYPE:0)
08:21:17.909 safe_app_2:   << CM: MSG for Session 0 received
08:21:17.909 safe_app_2:   << CM: EVENT [SREF:1]
08:21:17.909 safe_app_2:   !! GUI Session established: 16665
08:21:17.909 safe_app_2:   >> UI: event for service:1
08:21:17.909 safe_app_2:   >> UI: event for service:2
08:21:17.969 com_mgr_2:    << SA: data msg (SID:16665,TYPE:0)
08:21:17.969 com_mgr_2:    << SA: data msg (SID:16665,TYPE:0)
```

2f Platform monitor ready and all Tasksets are running (STATE -> OK)



Connection to trackside application established. Since the communication bridge is currently running on CE0, only the log (Safe Application #1, Safe Application #2) from CE0 is shown below.

```
09:46:45.748 safe_app_2:   << CM: MSG for Session 16665 received
09:46:45.748 safe_app_2:   << UI: Received message: 1
09:46:45.748 safe_app_2:   >> CM: TS connection request                          connection request to CM
09:46:45.808 com_mgr_2:    << SA: CON REQ to service 1 09:46:45.928 com_mgr_2:    >> connect request via
bridge 0 to service 1              request to bridge 09:46:45.928 com_mgr_2:    >> [session reference: 2]
09:46:45.988 com_bri_2_0:  << CG: Handling CTL MSG in bri on CE0
09:46:45.988 com_bri_2_0:  << CG: CTRL MSG Type: REQUEST
09:46:45.988 com_bri_2_0:  !! Handling CONNECT Request
09:46:45.988 com_bri_2_0:  << CG: connect request for TS_SERVICE_1
09:46:45.988 com_bri_2_0:  >> CH: request to Control Handler                  request to FRMCS agent
09:46:45.988 ctl_hdlr_2_0: << CONNECT request from comm_bridge
09:46:45.988 ctl_hdlr_2_0: >> CONNECT response SUCCESS to comm_bridge
09:46:45.988 ctl_hdlr_2_0: !! Doing Agent connect with timeout = 5s            FRMCS session setup
09:46:45.988 com_bri_2_0:  >> CM: SUCCESS Response from CH
09:46:45.988 com_bri_2_0:  >> CM: SUCCESS message
09:46:46.108 safe_app_2:   << CM: MSG for Session 0 received
09:46:46.108 safe_app_2:   !! Connecting TS Service session
09:46:59.078 ctl_hdlr_2_0: !! get remote IP for: dr-ts1.app.ts
09:46:59.078 ctl_hdlr_2_0: << remote IP: 10.36.20.113                IP for trackside service received
09:46:59.078 com_bri_2_0:  << CH: CONNECTED Event from CH
09:46:59.078 com_bri_2_0:  !! CB: Open socket to IP:10.36.20.113, PORT:12125          open socket
09:47:02.188 com_mgr_2:    !! Bridge session established: 7778
09:47:02.188 com_mgr_2:    !! Comm Manager session established: 16666
09:47:02.428 safe_app_2:   << CM: MSG for Session 0 received
09:47:02.428 safe_app_2:   << CM: EVENT [SREF:2]
09:47:02.428 safe_app_2:   !! Trackside Session established: 16666
09:47:02.428 safe_app_2:   >> UI: event for service:1
09:47:02.488 com_mgr_2:    << SA: data msg (SID:16665,TYPE:0)
09:47:03.028 safe_app_2:   << CM: MSG for Session 16665 received
09:47:03.028 safe_app_2:   << UI: Received message: 1
09:47:03.208 safe_app_2:   >> CM: 'TST SEQ 1' to ts (SID:16666)        send message to trackside app
09:47:03.268 com_mgr_2:    << SA: data msg (SID:16666,TYPE:0)
09:47:03.448 safe_app_2:   << CM: MSG for Session 16666 received
09:47:03.448 safe_app_2:   << CM: Received reply 'TST SEQ 1' [240ms]        reply from trackside app
09:47:08.248 safe_app_2:   >> CM: 'TST SEQ 2' to ts (SID:16666)        send message to trackside app
09:47:08.308 com_mgr_2:    << SA: data msg (SID:16666,TYPE:0)
09:47:08.488 safe_app_2:   << CM: MSG for Session 16666 received
09:47:08.488 safe_app_2:   << CM: Received reply 'TST SEQ 2' [240ms]        reply from trackside app
09:47:13.288 safe_app_2:   >> CM: 'TST SEQ 3' to ts (SID:16666)        send message to trackside app
09:47:13.348 com_mgr_2:    << SA: data msg (SID:16666,TYPE:0)
09:47:13.528 safe_app_2:   << CM: MSG for Session 16666 received
09:47:13.528 safe_app_2:   << CM: Received reply 'TST SEQ 3' [239ms]        reply from trackside app
09:47:15.748 etcs_app_1:   << CM: MSG for Session 16665 received
09:47:15.748 etcs_app_1:   << UI: Received message: 1
09:47:15.748 etcs_app_1:   >> CM: TS connection request                          connection request to CM
09:47:15.808 com_mgr_1:    << SA: CON REQ to service 1
09:47:15.928 com_mgr_1:    >> connect request via bridge 0 to service 1              request to bridge
09:47:15.928 com_mgr_1:    >> [session reference: 2]
09:47:15.988 com_bri_1_0:  << CG: Handling CTL MSG in bri on CE0
09:47:15.988 com_bri_1_0:  << CG: CTRL MSG Type: REQUEST
09:47:15.988 com_bri_1_0:  !! Handling CONNECT Request
09:47:15.988 com_bri_1_0:  << CG: connect request for TS_SERVICE_1
09:47:15.988 com_bri_1_0:  >> CH: request to Control Handler                  request to FRMCS agent
09:47:15.988 ctl_hdlr_1_0: << CONNECT request from comm_bridge
09:47:15.988 ctl_hdlr_1_0: >> CONNECT response SUCCESS to comm_bridge
09:47:15.989 ctl_hdlr_1_0: !! Doing Agent connect with timeout = 5s            FRMCS session setup
09:47:15.989 com_bri_1_0:  >> CM: SUCCESS Response from CH
09:47:15.989 com_bri_1_0:  >> CM: SUCCESS message
09:47:16.108 etcs_app_1:   << CM: MSG for Session 0 received
09:47:16.108 etcs_app_1:   !! Connecting TS Service session
09:47:18.328 safe_app_2:   >> CM: 'TST SEQ 4' to ts (SID:16666)        send message to trackside app
09:47:18.388 com_mgr_2:    << SA: data msg (SID:16666,TYPE:0)
09:47:18.568 safe_app_2:   << CM: MSG for Session 16666 received
09:47:18.568 safe_app_2:   << CM: Received reply 'TST SEQ 4' [240ms]        reply from trackside app
09:47:23.368 safe_app_2:   >> CM: 'TST SEQ 5' to ts (SID:16666)        send message to trackside app
09:47:23.428 com_mgr_2:    << SA: data msg (SID:16666,TYPE:0)
09:47:23.608 safe_app_2:   << CM: MSG for Session 16666 received
09:47:23.608 safe_app_2:   << CM: Received reply 'TST SEQ 5' [240ms]        reply from trackside app
```

```
09:47:28.408 safe_app_2:   >> CM: 'TST SEQ 6' to ts (SID:16666)            send message to trackside app
09:47:28.468 com_mgr_2:    << SA: data msg (SID:16666,TYPE:0)
09:47:28.648 safe_app_2:   << CM: MSG for Session 16666 received
09:47:28.648 safe_app_2:   << CM: Received reply 'TST SEQ 6' [240ms]          reply from trackside app
09:47:29.264 ctl_hdlr_1_0: !! get remote IP for: rbc1.app.ts
09:47:29.264 ctl_hdlr_1_0: << remote IP: 10.36.20.105                IP for trackside service received
09:47:29.264 com_bri_1_0:  << CH: CONNECTED Event from CH
09:47:29.264 com_bri_1_0:  !! CB: Open socket to IP:10.36.20.105, PORT:12125                open socket
09:47:29.308 com_mgr_1:    !! Bridge session established: 7778
09:47:29.308 com_mgr_1:    !! Comm Manager session established: 16666
09:47:29.548 etcs_app_1:   << CM: MSG for Session 0 received
09:47:29.548 etcs_app_1:   << CM: EVENT [SREF:2]
09:47:29.548 etcs_app_1:   !! Trackside Session established: 16666
09:47:29.548 etcs_app_1:   >> UI: event for service:1
09:47:29.608 com_mgr_1:    << SA: data msg (SID:16665,TYPE:0)
09:47:33.328 etcs_app_1:   >> CM: 'TST SEQ 1' to ts (SID:16666)            send message to trackside app
09:47:33.388 com_mgr_1:    << SA: data msg (SID:16666,TYPE:0)
09:47:33.448 safe_app_2:   >> CM: 'TST SEQ 7' to ts (SID:16666)            send message to trackside app
09:47:33.508 com_mgr_2:    << SA: data msg (SID:16666,TYPE:0)
09:47:33.568 etcs_app_1:   << CM: MSG for Session 16666 received
09:47:33.568 etcs_app_1:   << CM: Received reply 'TST SEQ 1' [239ms]          reply from trackside app
09:47:33.688 safe_app_2:   << CM: MSG for Session 16666 received
09:47:33.688 safe_app_2:   << CM: Received reply 'TST SEQ 7' [239ms]          reply from trackside app
09:47:38.368 etcs_app_1:   >> CM: 'TST SEQ 2' to ts (SID:16666)            send message to trackside app
09:47:38.428 com_mgr_1:    << SA: data msg (SID:16666,TYPE:0)
09:47:38.488 safe_app_2:   >> CM: 'TST SEQ 8' to ts (SID:16666)            send message to trackside app
09:47:38.548 com_mgr_2:    << SA: data msg (SID:16666,TYPE:0)
09:47:38.608 etcs_app_1:   << CM: MSG for Session 16666 received
09:47:38.608 etcs_app_1:   << CM: Received reply 'TST SEQ 2' [240ms]          reply from trackside app
09:47:38.728 safe_app_2:   << CM: MSG for Session 16666 received
09:47:38.728 safe_app_2:   << CM: Received reply 'TST SEQ 8' [240ms]          reply from trackside app
09:47:39.988 safe_app_2:   << CM: MSG for Session 16665 received
09:47:39.988 safe_app_2:   << UI: Received message: 1
09:47:39.988 safe_app_2:   >> CM: TS disconnect request                                disconnect request
09:47:40.048 com_mgr_2:    << SA: DIS REQ for session 16666
09:47:40.168 com_mgr_2:    >> CG: DIS REQ via CB 0 to SVC 1 [SREF: 2]
09:47:40.228 com_bri_2_0:  << CG: Handling CTL MSG in bri on CE0
09:47:40.228 com_bri_2_0:  << CG: CTRL MSG Type: REQUEST
09:47:40.228 com_bri_2_0:  !! Handling DISCONNECT Request
09:47:40.228 com_bri_2_0:  !! CB: Disconnecting Session:7778
09:47:40.228 com_bri_2_0:  >> CH: Disconnecting service: TS_SERVICE_1
09:47:40.228 ctl_hdlr_2_0: << DISCONNECT request from comm_bridge
09:47:40.228 com_bri_2_0:  >> CM: SUCCESS Response from CH
09:47:40.229 com_bri_2_0:  >> CM: SUCCESS message
09:47:40.348 safe_app_2:   << CM: MSG for Session 0 received
09:47:40.348 safe_app_2:   !! Connecting TS Service session
09:47:41.229 com_bri_2_0:  << CH: DISCONNECTED Event from CH
09:47:41.229 com_bri_2_0:  !! Bridge disconnected from Service
09:47:41.248 com_mgr_2:    !! Bridge session disconnected: 7778
09:47:41.248 com_mgr_2:    !! Comm Manager session disconnected: 16666
09:47:41.308 safe_app_2:   << CM: MSG for Session 0 received
09:47:41.308 safe_app_2:   << CM: EVENT [SREF:2]
09:47:41.308 safe_app_2:   !! Trackside Session disconnected: 16666          disconnected from trackside
09:47:41.308 safe_app_2:   >> UI: event for service:1
09:47:41.368 com_mgr_2:    << SA: data msg (SID:16665,TYPE:0)
09:47:43.408 etcs_app_1:   >> CM: 'TST SEQ 3' to ts (SID:16666)            send message to trackside app
09:47:43.468 com_mgr_1:    << SA: data msg (SID:16666,TYPE:0)
09:47:43.648 etcs_app_1:   << CM: MSG for Session 16666 received
09:47:43.648 etcs_app_1:   << CM: Received reply 'TST SEQ 3' [240ms]          reply from trackside app
09:47:46.588 etcs_app_1:   << CM: MSG for Session 16665 received
09:47:46.588 etcs_app_1:   << UI: Received message: 1
09:47:46.588 etcs_app_1:   >> CM: TS disconnect request                                disconnect request
09:47:46.648 com_mgr_1:    << SA: DIS REQ for session 16666
09:47:46.768 com_mgr_1:    >> CG: DIS REQ via CB 0 to SVC 1 [SREF: 2]
09:47:46.828 com_bri_1_0:  << CG: Handling CTL MSG in bri on CE0
09:47:46.828 com_bri_1_0:  << CG: CTRL MSG Type: REQUEST
09:47:46.828 com_bri_1_0:  !! Handling DISCONNECT Request
09:47:46.828 com_bri_1_0:  !! CB: Disconnecting Session:7778
09:47:46.828 com_bri_1_0:  >> CH: Disconnecting service: TS_SERVICE_1
09:47:46.828 ctl_hdlr_1_0: << DISCONNECT request from comm_bridge
09:47:46.828 com_bri_1_0:  >> CM: SUCCESS Response from CH
09:47:46.829 com_bri_1_0:  >> CM: SUCCESS message
09:47:46.948 etcs_app_1:   << CM: MSG for Session 0 received
09:47:46.948 etcs_app_1:   !! Connecting TS Service session
09:47:47.829 com_bri_1_0:  << CH: DISCONNECTED Event from CH
```

```
09:47:47.829 com_bri_1_0:  !! Bridge disconnected from Service
09:47:47.848 com_mgr_1:    !! Bridge session disconnected: 7778
09:47:47.848 com_mgr_1:    !! Comm Manager session disconnected: 16666
09:47:47.908 etcs_app_1:   << CM: MSG for Session 0 received
09:47:47.908 etcs_app_1:   << CM: EVENT [SREF:2]
09:47:47.908 etcs_app_1:   !! Trackside Session disconnected: 16666          disconnected from trackside
09:47:47.908 etcs_app_1:   >> UI: event for service:1
09:47:47.968 com_mgr_1:    << SA: data msg (SID:16665,TYPE:0)
```

Log of trackside application #1, interacting with Safe Application #1

```
user@tssbox01:~$ ./ts_app/ts_app
08:21:05.483 Server listening on port 12125
09:47:29.269 New connection accepted                                    connection established
09:47:29.433 << HBT REQ from com_mgr_1@CE0                                   heartbeat request
09:47:29.433 >> HBT ACK to com_mgr_1@CE0                                      heartbeat ack
09:47:30.031 << HBT REQ from com_mgr_1@CE0
09:47:30.031 >> HBT ACK to com_mgr_1@CE0
09:47:30.630 << HBT REQ from com_mgr_1@CE0
09:47:30.630 >> HBT ACK to com_mgr_1@CE0
09:47:31.231 << HBT REQ from com_mgr_1@CE0
09:47:31.231 >> HBT ACK to com_mgr_1@CE0
09:47:31.831 << HBT REQ from com_mgr_1@CE0
09:47:31.831 >> HBT ACK to com_mgr_1@CE0
09:47:32.431 << HBT REQ from com_mgr_1@CE0
09:47:32.431 >> HBT ACK to com_mgr_1@CE0
09:47:33.033 << HBT REQ from com_mgr_1@CE0
09:47:33.033 >> HBT ACK to com_mgr_1@CE0
09:47:33.454 << MSG RCV 'TST SEQ 1 [etcs_app_1]'                  application message received
09:47:33.454 >> MSG ACK 'TST SEQ 1 [etcs_app_1]'                     application message ack
09:47:33.630 << HBT REQ from com_mgr_1@CE0
09:47:33.630 >> HBT ACK to com_mgr_1@CE0
09:47:34.231 << HBT REQ from com_mgr_1@CE0
09:47:34.231 >> HBT ACK to com_mgr_1@CE0
.
.
.
09:47:38.490 << MSG RCV 'TST SEQ 2 [etcs_app_1]'
09:47:38.490 >> MSG ACK 'TST SEQ 2 [etcs_app_1]'
09:47:39.031 << HBT REQ from com_mgr_1@CE0
09:47:39.031 >> HBT ACK to com_mgr_1@CE0
09:47:39.631 << HBT REQ from com_mgr_1@CE0
09:47:39.631 >> HBT ACK to com_mgr_1@CE0
09:47:40.231 << HBT REQ from com_mgr_1@CE0
09:47:40.231 >> HBT ACK to com_mgr_1@CE0
09:47:40.841 << HBT REQ from com_mgr_1@CE0
09:47:40.841 >> HBT ACK to com_mgr_1@CE0
09:47:41.434 << HBT REQ from com_mgr_1@CE0
09:47:41.434 >> HBT ACK to com_mgr_1@CE0
09:47:42.044 << HBT REQ from com_mgr_1@CE0
09:47:42.044 >> HBT ACK to com_mgr_1@CE0
09:47:42.634 << HBT REQ from com_mgr_1@CE0
09:47:42.634 >> HBT ACK to com_mgr_1@CE0
09:47:43.234 << HBT REQ from com_mgr_1@CE0
09:47:43.235 >> HBT ACK to com_mgr_1@CE0
09:47:43.533 << MSG RCV 'TST SEQ 3 [etcs_app_1]'
09:47:43.533 >> MSG ACK 'TST SEQ 3 [etcs_app_1]'
09:47:43.833 << HBT REQ from com_mgr_1@CE0
09:47:43.833 >> HBT ACK to com_mgr_1@CE0
09:47:44.433 << HBT REQ from com_mgr_1@CE0
09:47:44.434 >> HBT ACK to com_mgr_1@CE0
09:47:45.040 << HBT REQ from com_mgr_1@CE0
09:47:45.040 >> HBT ACK to com_mgr_1@CE0
09:47:45.634 << HBT REQ from com_mgr_1@CE0
09:47:45.634 >> HBT ACK to com_mgr_1@CE0
09:47:46.231 << HBT REQ from com_mgr_1@CE0
09:47:46.231 >> HBT ACK to com_mgr_1@CE0
```

Log of Trackside Application #2, interacting with Safe Application #2.

```
08:21:08.781 Server listening on port 12125
09:47:02.177 New connection accepted                                    connection established
09:47:02.313 << HBT REQ from com_mgr_2@CE0                                   heartbeat request
```

```
09:47:02.313 >> HBT ACK to com_mgr_2@CE0                                heartbeat ack
09:47:02.914 << HBT REQ from com_mgr_2@CE0
09:47:02.914 >> HBT ACK to com_mgr_2@CE0
09:47:03.337 << MSG RCV 'TST SEQ 1 [safe_app_2]'                  application message received
09:47:03.337 >> MSG ACK 'TST SEQ 1 [safe_app_2]'                      application message ack
09:47:03.513 << HBT REQ from com_mgr_2@CE0
09:47:03.513 >> HBT ACK to com_mgr_2@CE0
.
.
.
09:47:08.314 << HBT REQ from com_mgr_2@CE0
09:47:08.314 >> HBT ACK to com_mgr_2@CE0
09:47:08.370 << MSG RCV 'TST SEQ 2 [safe_app_2]'
09:47:08.370 >> MSG ACK 'TST SEQ 2 [safe_app_2]'
.
.
.
09:47:38.611 << MSG RCV 'TST SEQ 8 [safe_app_2]'
09:47:38.611 >> MSG ACK 'TST SEQ 8 [safe_app_2]'
09:47:38.915 << HBT REQ from com_mgr_2@CE0
09:47:38.915 >> HBT ACK to com_mgr_2@CE0
09:47:39.514 << HBT REQ from com_mgr_2@CE0
09:47:39.514 >> HBT ACK to com_mgr_2@CE0
09:47:40.115 << HBT REQ from com_mgr_2@CE0
09:47:40.115 >> HBT ACK to com_mgr_2@CE0
```

Status	Pass
Comments	
Attachments	

3.14 INTEGRATE AN APPLICATION WITH FRMCS OVER LOOSE COUPLING [2.4.3]

Description User Story	As R2DATO WP36, we want to implement an OB_{App} client for an application on the modular computing platform (and an application on the safety layer / runtime environment) to enable communication over FRMCS to a trackside entity, so that we can demonstrate a “loose coupling” communication model where the MCX client is integrated in the FRMCS gateway.
Annotation	This User Story has already been demonstrated in WP36.3 (User Story [2.4.1] already used loose coupling) and here in WP36.4 with User Story [2.4.2].

3.15 COLLECT DIAGNOSTICS DATA FROM EXTERNAL BASIC INTEGRITY APP [2.5.1]

Description User Story	As R2DATO WP36, we want to collect diagnostics data from an external basic integrity application through a standardized interface, so that we can demonstrate a harmonised diagnostics service on the modular computing platform.
Annotation	MCP-DIA provides standardised interfaces (HTTP, REST) to clients and a unified API (on top of TCP sockets) for applications. From the perspective of the SOVD-Server it doesn't make a difference if diagnostics data is sent to the SOVD-Server locally or from external system. Client applications only need to have access to the provided TCP ports. As a result, the fulfilment of this user story is technically covered by User Story [2.5.2] and Test Case 2.5.2-A. The only change for setting up SystemHealthStatusApp as an external basic integrity app is its deployment on a different hardware than the SOVD-Server with the need to establish a TCP connection via a real network interface instead of localhost. To access diagnostics data, clients need to know the explicit Entity Path to the data. The server supports a query mechanism to obtain data points and applications need to execute the provisioning procedure as described in chapter 2.1.1.4 prior to external queries from MDCM client.

Test Case Title	**Collect diagnostics data from an external basic integrity application deployed on a different computing element**
Test Case ID	2.5.1-A
Description	It shall be demonstrated how diagnostics data gets collected from a basic integrity application on the safety layer / runtime environment (Model-3 Taskset) deployed on a different computing element to be provided through a standardized interface as a harmonised diagnostics service.
Test Environment	Two Computing Elements (running TAS Platform) in the laboratory deployed with SOVD-Server at CE0 and SystemHealthStatusApp on CE1. Both CEs need access to each other's network interface to establish the TCP connection between the Server and the App. The data gets queried over the laboratory network from an external computer integrated in the laboratory. Any host machine that has access to the network and endpoint (target HW, running TAS Platform) can reach the MCP-DIA service over TCP/IP.
Preconditions	CEs need to be booted, and SOVD-Server and SystemHealthStatusApp get started A configured network connection is needed between the host and the target system
Test Steps	1. Start the SOVD-Server with appropriate configuration on CE0 2. Start SystemHealthStatusApp on CE1 (the App executes the provisioning procedure towards the SOVD-Server on startup as explained in chapter 2.1.1.4) 3. Query the data from MCP-DIA via its REST SOVD interface (HTTP GET)
Test Data	Example query for faults stored in SystemHealthStatusApp to the SOVD-Server curl -s http://10.36.130.20:7690/v1/apps/SystemHealthStatusApp/faults
Expected Result	Received data gets evaluated by a regular expression: grep -E -q '\{"items":\[.*\{"code":"345435","scope":"","display_code":"345435","fault_name":"cpu_load_high","fault_translation_id":"translation_345435","severity":1,"status":\{"aggregated_status":"Active","testFailed":true,"testFailedThisOperationCycle":true,"testNotCompletedSinceLastClear":false,"testFailedSinceLastClear":true,"testNotCompletedThisOperationCycle":false\},"symptom":"","symptom_transaction_id":""\}.*\]\}'
Actual Result	{"items":[{"code":"345435","scope":"","display_code":"345435","fault_name":"cpu_load_high","fault_translation_id":"translation_345435","severity":1,"status":{"aggregated_status":"Active","testFailed":true,"testFailedThisOperationCycle":true,"testNotCompletedSinceLastClear":false,"testFailedSinceLastClear":true,"testNotCompletedThisOperationCycle":false},"symptom":"","symptom_transaction_id":""},{"code":"345436","scope":"","display_code":"345436","fault_name":"free_memory_percent_low","fault_translation_id":"translation_345436","severity":1,"status":{"aggregated_status":"Active","testFailed":true,"testFailedThisOperationCycle":true,"testNotCompletedSinceLastClear":false,"testFailed

	SinceLastClear":true,"testNotCompletedThisOperationCycle":false},"symptom":"","symptom_transaction_id":""}]}
Status	Test succeeds
Comments	Test Result content available as Azure DevOps Pipeline artefact (the pipeline writes the received data into a file)
Attachments	N/A – online available for WP36 partners in Azure Pipeline runs

3.16 COLLECT DIAGNOSTICS DATA FROM BASIC INTEGRITY APP ON RTE [2.5.2]

Description User Story	As R2DATO WP36, we want to collect diagnostics data from a basic integrity application on the safety layer / runtime environment through a standardized interface, so that we can demonstrate a harmonised diagnostics service on the modular computing platform.
Annotation	The fulfilment of this user story has already been presented in the former deliverable D36.3 [3] chapter 4.2.1 but we present here the detailed test case for completeness.

Test Case Title	**Collect diagnostics data from a basic integrity application on the safety layer / runtime environment deployed on the same computing element**
Test Case ID	2.5.2-A
Description	It shall be demonstrated how diagnostics data gets collected from a basic integrity application on the safety layer / runtime environment (Model-3 Taskset) deployed on the same computing element to be provided through a standardized interface as a harmonised diagnostics service.
Test Environment	One Computing Element (running TAS Platform) in the laboratory deployed with SOVD-Server and at least one Application realised as Model-3 Taskset. The data gets queried over the laboratory network from an external computer integrated in the laboratory. Any host machine that has access to the network and endpoint (target HW, running TAS Platform) can in principle reach the MCP-DIA service over TCP/IP.
Preconditions	• CE needs to be booted, and SOVD-Server and SystemHealthStatusApp get started • A configured network connection is needed between the host and the target system
Test Steps	1. Start the SOVD-Server with appropriate configuration 2. Start SystemHealthStatusApp (the App executes the provisioning procedure towards the SOVD-Server on startup as explained in chapter 2.1.1.4) 3. Query the data from MCP-DIA via its REST SOVD interface (HTTP GET)
Test Data	Example query for faults stored in SystemHealthStatusApp to the SOVD-Server curl -s http://10.36.130.20:7690/v1/apps/SystemHealthStatusApp/faults
Expected Result	Received data gets evaluated by a regular expression: grep -E -q '\{"items":\[.*\{"code":"345435","scope":"","display_code":"345435","fault_name":"cpu_load_high","fault_translation_id":"translation_345435","severity":1,"status":\{"aggregated_status":"Active","testFailed":true,"testFailedThisOperationCycle":true,"testNotCompletedSinceLastClear":false,"testFailedSinceLastClear":true,"testNotCompletedThisOperationCycle":false\},"symptom":"","symptom_transaction_id":""\}.*\]\}'
Actual Result	{"items":[{"code":"345435","scope":"","display_code":"345435","fault_name":"cpu_load_high","fault_translation_id":"translation_345435","severity":1,"status":{"aggregated_status":"Active","testFailed":true,"testFailedThisOperationCycle":true,"testNotCompletedSinceLastClear":false,"testFailedSinceLastClear":true,"testNotCompletedThisOperationCycle":false},"symptom":"","symptom_transaction_id":""},{"code":"345436","scope":"","display_code":"345436","fault_name":"free_memory_percent_low","fault_translation_id":"translation_345436","severity":1,"status":{"aggregated_status":"Active","testFailed":true,"testFailedThisOperationCycle":true,"testNotCompletedSinceLastClear":false,"testFailed

	SinceLastClear":true,"testNotCompletedThisOperationCycle":false},"symptom":"","symptom_transaction_id":""}]}
Status	Test succeeds
Comments	Test Result content available as Azure DevOps Pipeline artefact (the pipeline writes the received data into a file)
Attachments	N/A – online available for WP36 partners in Azure Pipeline runs

3.17 COLLECT DIAGNOSTICS DATA FROM SAFE APPLICATION ON RTE [2.5.3]

Description User Story	As R2DATO WP36, we want to collect diagnostics data from a safe application (up to SIL 4) on the safety layer / runtime environment through a standardized interface, so that we can demonstrate a harmonised diagnostics service on the modular computing platform.
Annotation	This User Story has been realised as described in the Diagnostics Blueprint Chapter 2.1.2. Diagnostics data from a safe application (up to SIL 4) running on the safety layer / runtime environment, gets forwarded to a basic integrity application (Model-3 Taskset) via message queues, that can directly access the network ports of the SOVD-Server. The following test case utilises the Diagnostics Blueprint, providing data from etcs_dia that has been configured to run as a Model-1 Taskset. As explained in chapter 2.2.1.2, MCP-DIA has been integrated into the final demonstration setup, including a safe ETCS onboard application. For details, please refer to chapter 3.50, test case “2.10.2-C Drive simulated Train in Lab – including Diagnostics”.

Test Case Title	**Collect diagnostics data from a safe application on the safety layer / runtime environment deployed on the same computing element**
Test Case ID	2.5.3-A
Description	It shall be demonstrated how diagnostics data gets collected from a safe application (up to SIL 4) on the safety layer / runtime environment (Model-1 Taskset) deployed on the same computing element to be provided through a standardized interface as a harmonised diagnostics service.
Test Environment	One Computing Element (running TAS Platform) in the laboratory deployed with SOVD-Server and at least one application realised as Model-1 Taskset. The data gets queried over the laboratory network from an external computer integrated in the laboratory. Any host machine that has access to the network and endpoint (target HW, running TAS Platform) can in principle reach the MCP-DIA service over TCP/IP.
Preconditions	• CE needs to be booted, and SOVD-Server and etcs_dia get started • A configured network connection is needed between the host and the target system
Test Steps	1. Start the Sovd-Server with appropriate configuration 2. Start etcs_dia (the App executes the provisioning procedure towards the SOVD-Server on startup as explained in chapter 2.1.1.4) 3. Query the data from MCP-DIA via its REST SOVD interface (HTTP GET)
Test Data	Example query for faults stored in etcs_dia to the SOVD-Server curl -s http://10.36.130.20:7690/v1/apps/Model1Taskset/faults/P0138
Expected Result	Received data gets evaluated by a regular expression: grep -E -q '{"item":{"code":"P0138","scope":"","display_code":"P0138","fault_name":"ATO Brakes Ctrl","fault_translation_id":"translation_P0138","severity":1,"status":{"aggregated_status":"NotSet","testFailed":false,"testFailedThisOperationCycle":false,"testNotCompletedSinceLastClear":false,"testFailedSinceLastClear":false,"testNotCompletedThisOperationCycle":false},"symptom":"","symptom_transaction_id":""},"environment_data":{"OccurrenceCounter":[0-9]+,"LastOccurrence":"[0-9]{13}","FirstOccurrence":"[0-9]{13}"}}'
Actual Result	fault details response from Model1Taskset: {"item":{"code":"P0138","scope":"","display_code":"P0138","fault_name":"ATO Brakes Ctrl","fault_translation_id":"translation_P0138","severity":1,"status":{"aggregated_status":"NotSet","testFailed":false,"testFailedThisOperationCycle":false,"testNotCompletedSinceLastClear":false,"testFailedSinceLastClear":false,"testNotCompletedThisOperationCycle":false},"symptom":"","symptom_transaction_id":""},"environment_data":{"OccurrenceCounter":1,"LastOccurrence":"1764847133286","FirstOccurrence":"1764847133286"}}
Status	Test succeeds

Comments	Test Result content available as azure artefact (the pipeline writes the received data into a file)
Attachments	N/A – online available for WP36 partners in Azure Pipeline runs

3.18 COLLECT DIAGNOSTICS DATA FROM SAFETY LAYER / RTE [2.5.4]

Description User Story	As R2DATO WP36, we want to collect diagnostics data from the safety layer / runtime environment through a standardized interface, so that we can demonstrate a harmonised diagnostics service on the modular computing platform.
Annotation	The collection of data from the Safety Layer is feasible via accessing the `/var/diag` filesystem as described in the TAS Platform manual. The implementation of reading a file system from an application and providing the acquired data to MCP-DIA has already been realised in SystemHealthStatusApp, where `/proc/stat` gets accessed to acquire the CPU load. Accessing `/var/diag` would not bring added value due to its similarity to the current implementation but would fulfil the User Story. Since the safety layer writes as well into Syslog, we have decided to cover this user story by utilising the newly realised feature of reading out (syslog-ng) logging information and providing it to SOVD-clients.

Test Case Title	**Collect diagnostics data from external basic integrity application deployed on the same computing element**
Test Case ID	2.5.4-A
Description	It shall be demonstrated how diagnostics data gets collected from other applications or services deployed on the same computing element to be provided through a standardized interface as a harmonised diagnostics service.
Test Environment	One Computing Element (running TAS Platform) in the laboratory deployed with SOVD-Server (and other applications to demonstrate the ability to integrate MCP-DIA in typical TAS Platform setups that could be much more sophisticated).

Preconditions	CE needs to be booted, and relevant configurations need to be applied. For this test case syslog-ng gets utilised as external service, therefore an appropriate sylog-ng.conf (that declares the SOVD-Server as “destination”) must be applied.
	`destination d_sovd_server {`
	`  network("127.0.0.1"`
	`  port(515)`
	`  transport(tcp)`
	`  template("Date=${ISODATE};Host=${HOST};Program=${PROGRAM};PID=${PID};Level=${LEVEL};Message=${MESSAGE}\`
	`};`
	`log {`
	`  source(aeos_msgs);`
	`  destination(messages_program);`
	`  destination(d_sovd_server);`
	`cp /usr/local/erju/mcp-dia/conf/syslog-ng.conf /etc/syslog-ng/`
	`/etc/init.d/syslog-ng restart`

Test Steps	1. Start the Sovd-Server 2. Copy adapted sylog-ng.conf to /etc/syslog-ng/ 3. Restart the syslog-ng daemon 4. Wait until data from syslog has been received 5. Query the data from MCP-DIA via its SOVD interface (http GET)
Test Data	Query syslog data of interest, example: Syslogs from a running application: curl -s http://10.36.130.20:7690/v1/apps/SystemHealthStatusApp/logs/entries

Expected Result	Any syslog entries logged, here an example for SystemHealthStatusApp, to be tested if they are existing in the received data by a regular expression: grep -E -q '{"items":\[.+\]}'
	`case sovd::LibSovdServerMessage::kReadValueRequest: {`
	`  syslog(LOG_INFO, "Received ReadValueRequest");`

Actual Result	{"items":[{"timestamp":"2025-12-04T11:19:00.637Z","context":{"type":"RFC5424","host":"ce0","process":"system_health_status_dia","pid":1418},"severity":"info","msg":" Received ReadValueRequest "}]}
Status	Test succeeds

Comments	Test Result content available on Azure DevOps as artefact (the pipeline writes to a file)
Attachments	N/A – online available for WP36 partners in Azure Pipeline runs

3.19 PROVIDE DIAGNOSTICS EVENTS TO EXTERNAL BASIC INTEGRITY APP [2.5.5]

Description User Story	As R2DATO WP36, we want to provide diagnostics events to an external basic integrity application through a standardized interface, so that we can demonstrate a harmonised diagnostics service on the modular computing platform.
Annotation	The provision of Diagnostics data to the MDCM Client already fulfils this User Story. Another way to realise this User Story is similar to [2.5.1] and [2.5.2]. As pointed out in User Story [2.5.1], it doesn't make any difference from the perspective of the SOVD-Server if diagnostics data gets read from or written to its TCP Server ports from external or RTE hosted applications, as long as they can reach the port over the network and follow the needed provisioning procedure as described in chapters 2.1.1.4 and 2.1.1.5.

3.20 PROVIDE DIAGNOSTICS EVENTS TO BASIC INTEGRITY APP ON RTE [2.5.6]

Description User Story	As R2DATO WP36, we want to provide diagnostics events to a basic integrity application on the safety layer / runtime environment through a standardized interface, so that we can demonstrate a harmonised diagnostics service on the modular computing platform.
Annotation	The fulfilment of this user story has already been presented in the former Deliverable D36.3 [3] chapter 4.2.2. As stated there, the successful provision of this diagnostics event to the application can be proven by the reception of the respective diagnostics data as described in the general end-to-end test scenario, more explicitly following the User Story [2.5.2] applying the REQUEST_RESPONSE communication pattern towards the application. Further details on the REQUEST_RESPONSE communication pattern are available in chapter 2.1.1.5 of this deliverable. The Test Case with ID 2.5.2-A is applicable here, it has been successfully tested with both communication patterns in the laboratory and will not be repeated here again.

3.21 PROVIDE DIAGNOSTICS EVENTS TO SAFE APPLICATION ON RTE [2.5.7]

Description User Story	As R2DATO WP36, we want to provide diagnostics events to a safe application (up to SIL 4) on the safety layer / runtime environment through a standardized interface, so that we can demonstrate a harmonised diagnostics service on the modular computing platform.
Annotation	Omitted: This User Story is not demonstrated because of an initially wrong understanding of "events". In general, it is not allowed to send diagnostics data

	towards safe applications (up to SIL 4) impacting their behaviour, branching & execution flow or safety logic. The provision of events to a Model-1 Taskset (instead to a Model-3 Taskset) can be considered as writing to a Message Queue that gets read by a Model-1 Taskset. This has been demonstrated already along multiple User Stories, e.g., [2.3.3], [2.3.4], [2.3.5] but the processing of such unsafe diagnostics events within the Model-1Taskset (changing its behaviour) is not possible. Therefore, the User Story has been evaluated as unfeasible.

3.22 PROVIDE DIAGNOSTICS EVENTS TO SAFETY LAYER / RTE [2.5.8]

Description User Story	As R2DATO WP36, we want to provide diagnostics events to the safety layer / runtime environment through a standardized interface, so that we can demonstrate a harmonised diagnostics service on the modular computing platform.
Annotation	Omitted: This User Story is not applicable, no data/events can be provided to the Safety Layer of the TAS Platform. Accessing the filesystem `/var/diag` would be feasible by implementing a specific application but this would provide no real value during operations. SOVD Bulk Data writes or Script execution are not supported by the current implementation of the SOVD-Server.

3.23 EXCHANGE DIAGNOSTICS DATA WITH TRACKSIDE DIAGNOSTICS SERVICE [2.5.9]

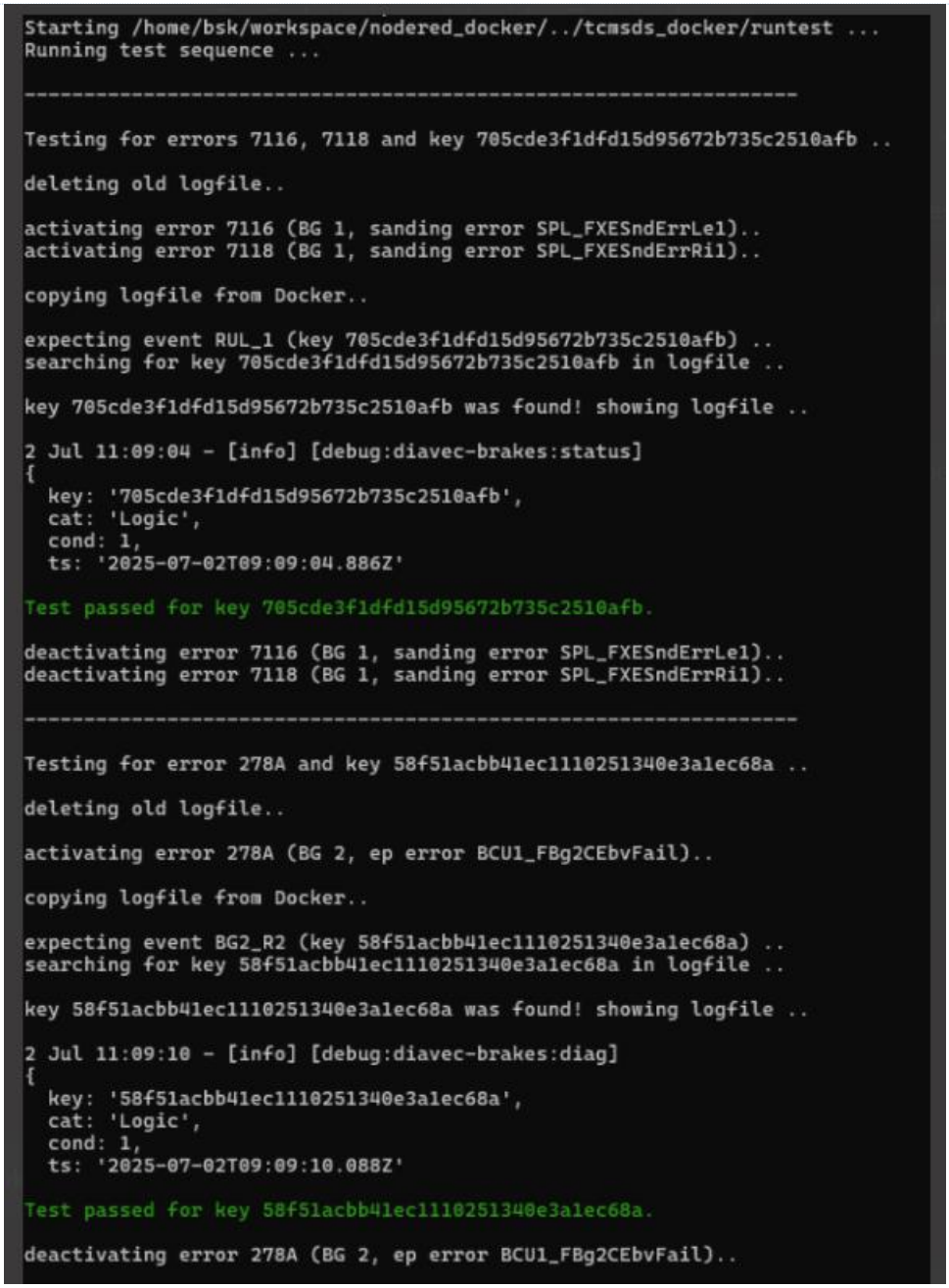
Description User Story	As R2DATO WP36, we want to exchange diagnostics data with a trackside diagnostics service over FRMCS, so that we can demonstrate a harmonised diagnostics service on the modular computing platform.
Annotation	Omitted: A trackside diagnostics service has not been implemented in the project. In case the MDCM Client would be deployed as a wayside component connected over FRMCS we may argue that the User Story could be considered as fulfilled. In that case the TCP/IP communication would need to be routed via FRMCS. Anyhow, neither the MCP-DIA nor MDCM Client have implemented any specific FRMCS connectivity.

3.24 COLLECT AND AGGREGATE DIAGNOSTICS DATA FROM VEHICLE [2.5.10]

Description User Story	As R2DATO WP36, we want to collect and aggregate diagnostics data from the vehicle via the TCMS Data Service, so that we can demonstrate the vehicle diagnostics agent (DIA-VEC) as basic integrity application on the modular computing platform.
Annotation	The user story is fulfilled and tested driven by an automated DevOps integration and pipeline test run, according to test case 2.5.10-A below.

Test Case Title	**Integration test for DIA-VEC**
Test Case ID	2.5.10-A
Description	This integration test checks the correct data transmission and handling of the TCMS Data Service and the DIA-VEC software. It is an automated test routine of the integration pipeline (db_dia-integration) executing the following tasks: 1. Start of components 2. Injection of data 3. Check of results
Test Environment	Integration Test of WP36 DIA-VEC Brakes in Azure DevOps
Preconditions	Artifacts are available in Azure (get checked out from the repositories): • TCMS_DS • DIA-VEC Brakes • Node Red Test Client The integration pipeline (db_dia-integratio": {is available and can connect to the Azure DevOps Pipeline agent deployed in the laboratory
Test Steps	Execute the integration pipeline (db_dia-integration: link) in Azure. 1. Start of components: The following components are uploaded to the respective targets and executed: • Data Playback • TCMS_DS • DIA-VEC Brakes • Node Red (Diagnostic Client) 2. Injection of data: • The integration test injects data to TCMS_DS. The data is a playback of recorded status messages and failure events, identical to the data sent by the Brakes system. 3. Check of results • The integration test compares the data, stored by the Node Red client as output, with the injected data. When the output data meets the expectation, the test is passed. For details on this client, please refer to chapter 2.5.6.
Test Data	The test data is included in the 'runtest' file, located in the Azure repo db_dia-tcms-dataservice. The data source provides messages for Brakes states and diagnostics: # BrakeState definitions

	S01='{"base":{"fs":"inop","hm":"noerr"},"stat":{"sb":"unknown","pb":"released"}, perfm":{"brkp":"0","perf":"0"}}' S02='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"selfTest","pb":"engaged"},"perfm":{"brkp":"0","perf":"0"}}' S03='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"holdingBrake","pb":"engaged"},"perfm":{"brkp":"140","perf":"100"}}' S04='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"holdingBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}' S05='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"noServiceBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}' S06='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"serviceBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}' S07='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"noServiceBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}' S08='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"serviceBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}' S09='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"noServiceBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}' S10='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"slowingBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}' S11='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"stoppingBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}' S12='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"holdingBrake","pb":"released"},"perfm":{"brkp":"140","perf":"100"}}' S13='{"base":{"fs":"op","hm":"noerr"},"stat":{"sb":"holdingBrake","pb":"engaged"},"perfm":{"brkp":"140","perf":"100"}}' S14='{"base":{"fs":"inop","hm":"noerr"},"stat":{"sb":"unknown","pb":"engaged"},"perfm":{"brkp":"0","perf":"0"}}' # Test sequence: /update_state/brakes/bcu[id=GU1]/ echo "Running test sequence ... " tcmsdsReset ; sleep 1 brakesUpdateState "${GU1}" "${S01}" ; sleep 1 brakesUpdateState "${GU1}" "${S02}" ; sleep 1 brakesUpdateState "${GU1}" "${S03}" ; sleep 1 … brakesUpdateState "${GU1}" "${S14}" ; sleep 1
Expected Result	The integration pipeline compares the output of Node Red and checks if expected output matches the input. In case the output is as expected, the following message appears in the log file: "*Test passed for key xxx*" In case the output is NOT as expected, the following message appears in the log file: *"Test failed for key xxx"* The following screenshot shows an example of the output:



Actual Result	Tests passed for key…
Status	Test succeeds
Comments	Test Result content available as Azure DevOps Pipeline artefact (the pipeline writes to a file)
Attachments	N/A – online available for WP36 partners in Azure Pipeline runs

3.25 UPDATE A BASIC INTEGRITY APPLICATION ON PLATFORM [2.6.1]

Description User Story	As R2DATO WP36, we want to update a basic integrity application on the modular computing platform, so that we can demonstrate the technical workflow for the software update.
Annotation	Theoretical analysis within chapter 2.4 Software Update.

3.26 UPDATE A BASIC INTEGRITY APPLICATION ON SAFETY LAYER / RTE [2.6.2]

Description User Story	As R2DATO WP36, we want to update a basic integrity application on the safety layer / runtime environment, so that we can demonstrate the technical workflow for the software update.
Annotation	Theoretical analysis within chapter 2.4 Software Update.

3.27 UPDATE A SAFE APPLICATION ON SAFETY LAYER / RTE [2.6.3]

Description User Story	As R2DATO WP36, we want to update a safe application (up to SIL 4) on the safety layer / runtime environment, so that we can demonstrate the technical workflow for the software update.
Annotation	Theoretical analysis within chapter 2.4 Software Update.

3.28 UPDATE THE SAFETY LAYER / RTE [2.6.4]

Description User Story	As R2DATO WP36, we want to update the safety layer / runtime environment on the modular computing platform, so that we can demonstrate the technical workflow for the software update of the safety and runtime layer.
Annotation	Theoretical analysis within chapter 2.4 Software Update.

3.29 UPDATE THE PLATFORM [2.6.5]

Description User Story	As R2DATO WP36, we want to update the modular computing platform, so that we can demonstrate the technical workflow for the software update of the virtualization layer.
Annotation	Theoretical analysis within chapter 2.4 Software Update.

3.30 Rollback a Software / Configuration Update [2.6.6]

Description User Story	As R2DATO WP36, we want to rollback a software / configuration update, so that we can demonstrate the failsafe scenario for a corrupt / incomplete update process.
Annotation	Theoretical analysis within chapter 2.4 Software Update.

3.31 Perform a Software / Configuration Update over the Air (FRMCS) [2.6.7]

Description User Story	As R2DATO WP36, we want to perform a software / configuration update over the air using FRMCS, so that we can demonstrate possibility of a remote update.
Annotation	Theoretical analysis within chapter 2.4 Software Update.

3.32 Allow Exchange of Data over the Network [2.7.1]

Description User Story	As R2DATO WP36, we want to exchange data between onboard components via a standardized onboard communication network, so that we can demonstrate the standardized integration of the modular computing platform.
Annotation	Covered by 3.38 Prioritise Specific Network Traffic [2.7.7]

3.33 Provision of Uninterrupted Connectivity [2.7.2]

Description User Story	As R2DATO WP36, we want to provide uninterrupted connectivity between onboard components even if up to a defined number of onboard components fail, so that we can demonstrate reliable communication paths.

Test Case Title	**Simulate faulty network cable**
Test Case ID	2.7.2-A
Description	Run communication session over FRMCS between onboard and trackside applications and demonstrate uninterrupted connectivity due to a connection failure (cable failure) on the CCS Communication Network (CCN).
Test Environment	• Modular Platform (TAS Platform) • Physical FRMCS onboard Gateway • FRMCS trackside Gateway on a VM • CCS Communication Network (CCN) • Trackside application simulations running as VM on the lab server

- User interfaces (OB GUI) for onboard application
- onboard application (etcs_app_1) running on the Modular Platform

The CCN consists of four switches connected in a ring and have the Rapid Spanning Tree Protocol (RSTP) enabled. During normal operation, port X1 of switch 3 blocks the traffic to avoid network loops.

In case the ring is broken due to a failing switch (e.g. nor power) or a faulty network cable, the traffic is rerouted over the redundant link on the ring.

This testcase demonstrates that such rerouting of the traffic has no impact to the onboard application and the message exchange between onboard and trackside applications.

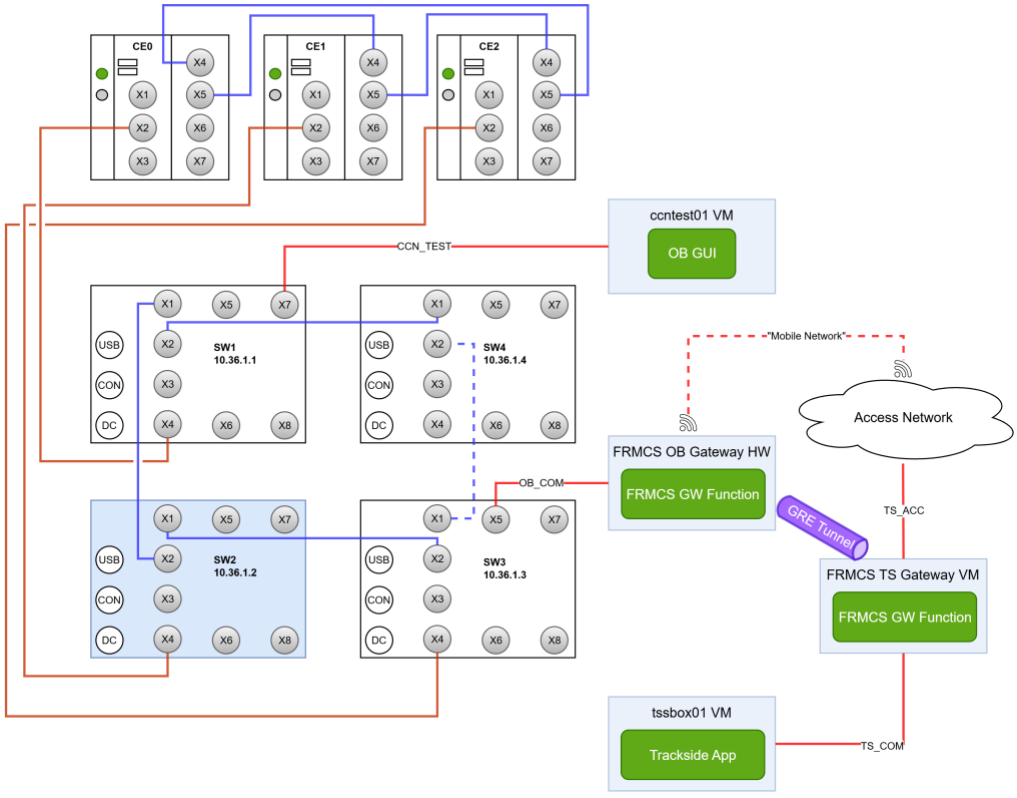


Figure 24 - Setup

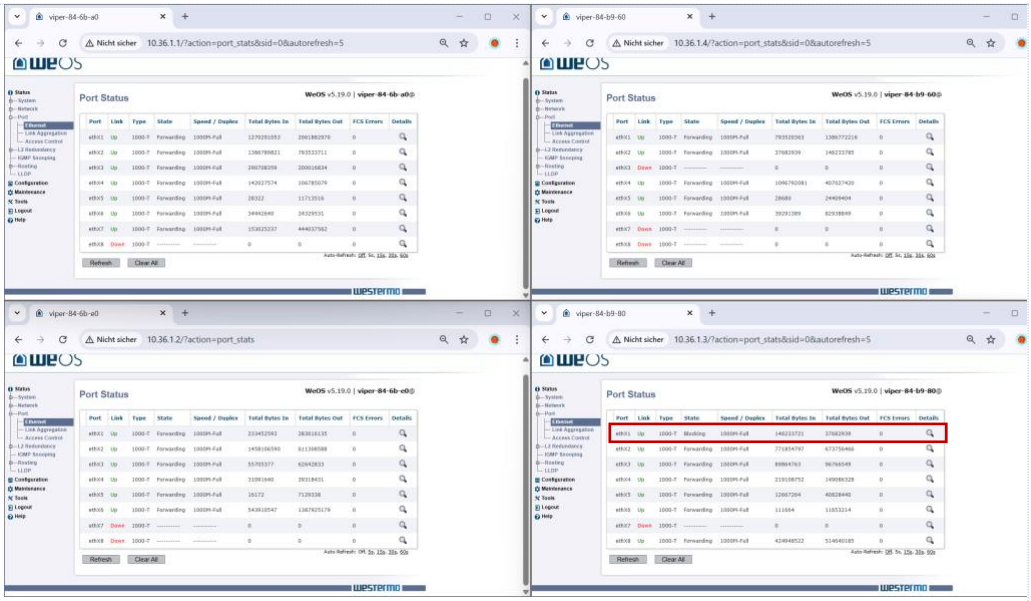
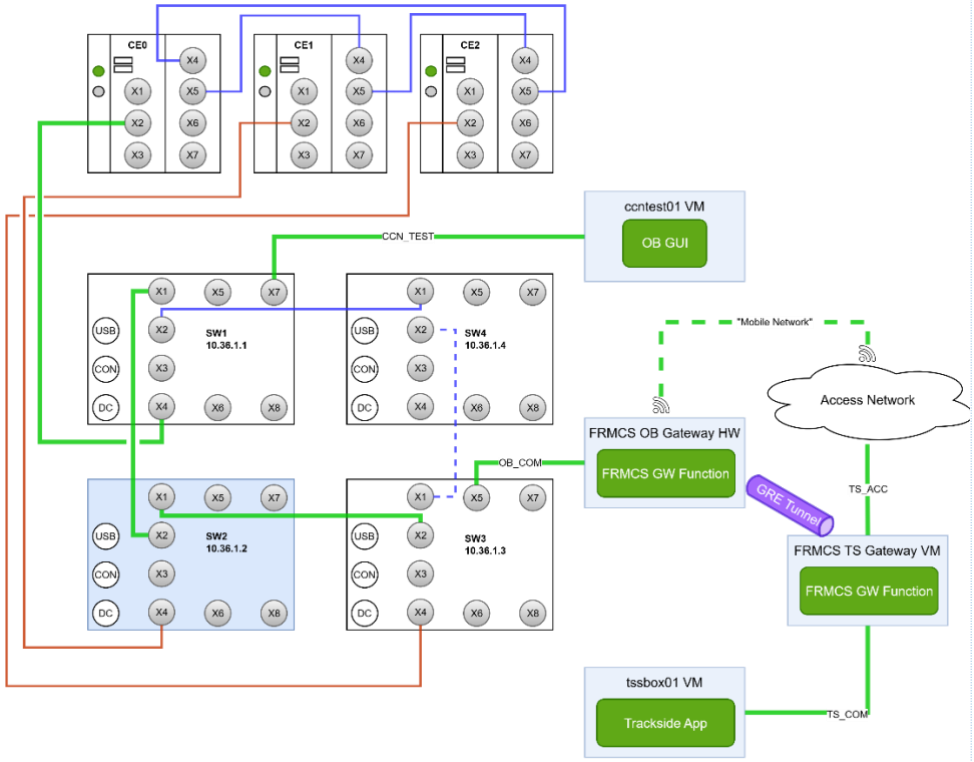


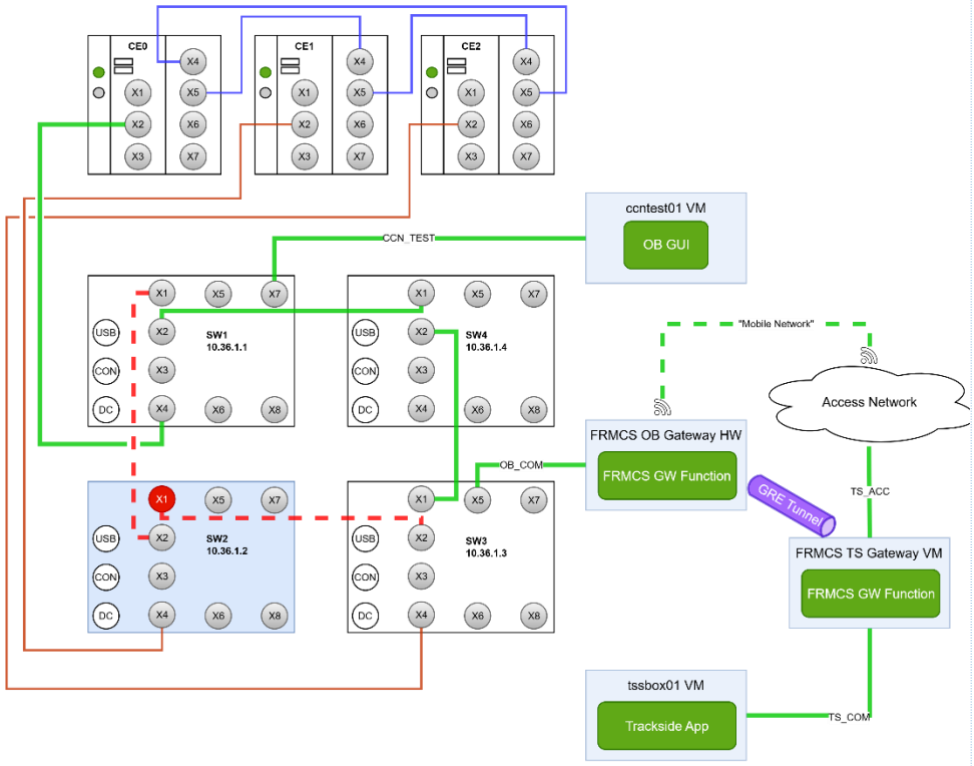
Figure 25 CCN with blocking port X1 on switch 3

Preconditions	Set up Test environment in a manner that runs long enough for this Test Case.
Test Steps	6. Prepare testbench 7. Start testbench components and monitoring a. Start FMCS trackside GW logging b. Register trackside application to trackside GW c. Start trackside application d. Start onboard user interface (OB GUI) e. Start the onboard safe application (etcs_app_1) f. Start platform monitor 8. Initiate the connection from the onboard application to the trackside counterpart. 9. Verify message exchange between onboard applications and trackside applications



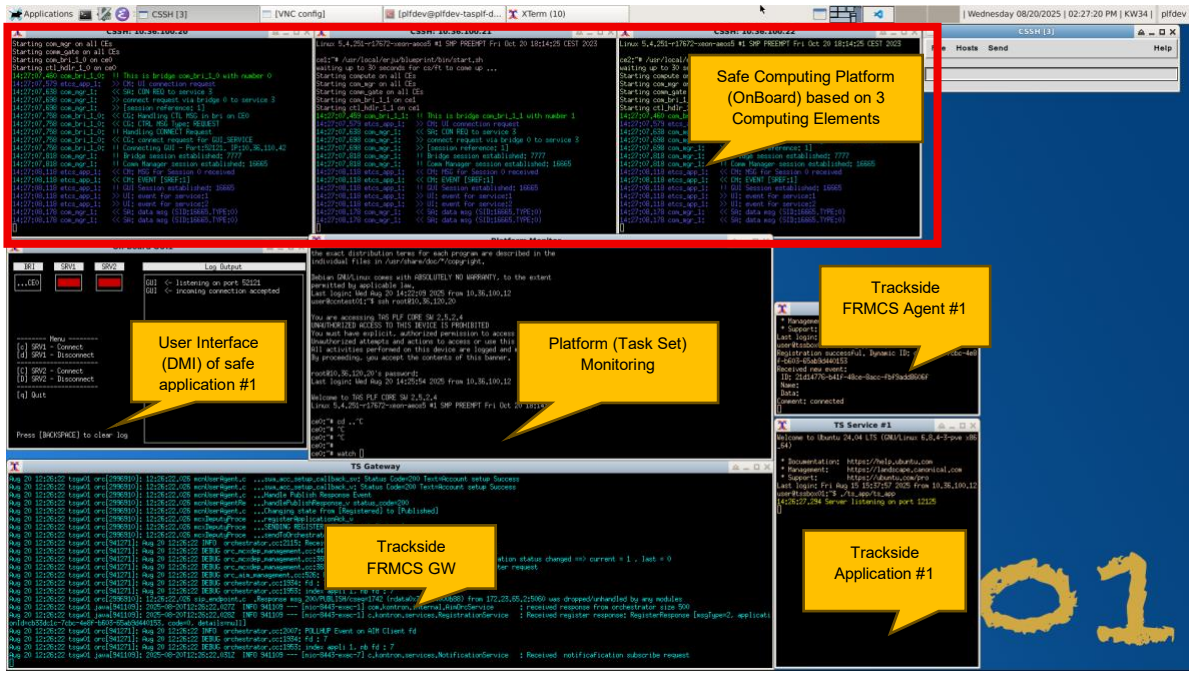
During normal operation, network traffic for the OB GUI and the communication between onboard and trackside applications will use the “green” route as indicated in the figure above.

10. Disable port X1 on switch 2
11. Verify, that message exchange is not interrupted and application operates as expected



	During the degraded mode, the network traffic on the CCN is rerouted due to the reconfiguration of the switches by unblocking the link between switch 3 and switch 4. In this scenario, the traffic for the OB GUI is not affected and therefore not rerouted. 12. Re-enable port X1 on switch 2 13. Verify, that message exchange is not interrupted and application operates as expected
Test Data	None.
Expected Result	During the full test, the system must continue to perform its intended functions, without any detectable issues or degradation.

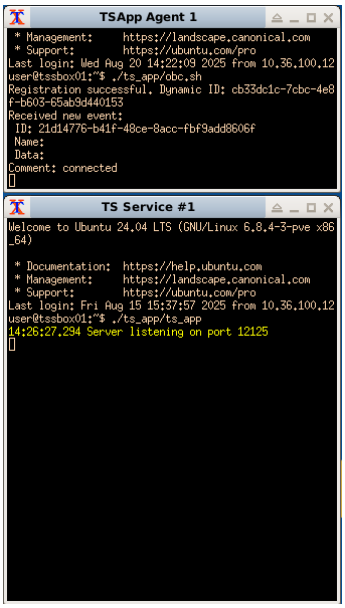
Actual Result	
1	All terminals ready



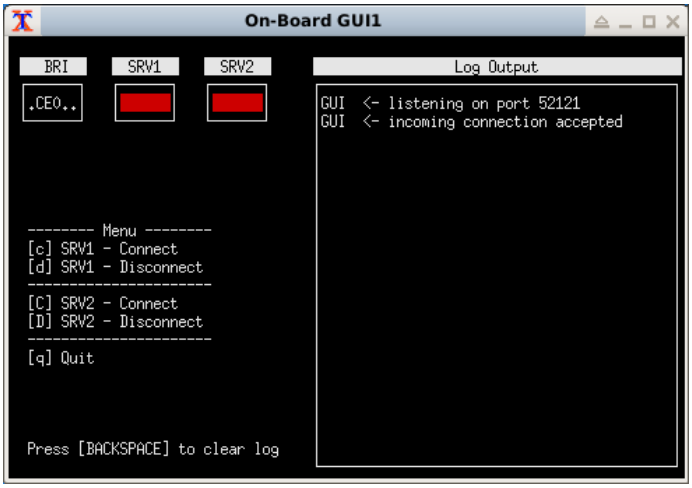
2	Testbench started
2a	Trackside FRMCS GW started.

2b Trackside application running and registered to trackside FRMCS GW.

2c



2d onboard user interfaces ready

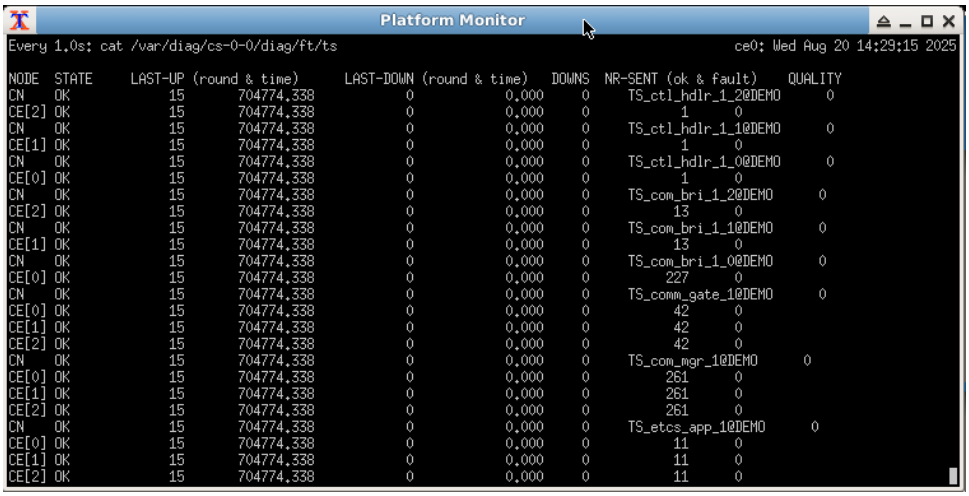


2e onboard application ready and connected to the user interface

```
waiting up to 30 seconds for cs/ft to come up ...
Starting compute on all CEs
Starting com_mgr on all CEs
Starting comm_gate on all CEs
Starting com_bri_1_0 on ce0
Starting ctl_hdlr_1_0 on ce0
14:27:07.460 com_bri_1_0:  !! This is bridge com_bri_1_0 with number 0
14:27:07.579 etcs_app_1:   >> CM: UI connection requestc14:27:07.638 com_mgr_1:    << SA: CON
REQ to service 3c14:27:07.698 com_mgr_1:    >> connect request via bridge 0 to service 3
14:27:07.698 com_mgr_1:    >> [session reference: 1]
14:27:07.758 com_bri_1_0:  << CG: Handling CTL MSG in bri on CE0
14:27:07.758 com_bri_1_0:  << CG: CTRL MSG Type: REQUEST
14:27:07.758 com_bri_1_0:  !! Handling CONNECT Request
14:27:07.758 com_bri_1_0:  << CG: connect request for GUI_SERVICE
14:27:07.758 com_bri_1_0:  !! Connecting GUI - Port:52121, IP:10.36.110.42
14:27:07.818 com_mgr_1:    !! Bridge session established: 7777
14:27:07.818 com_mgr_1:    !! Comm Manager session established: 16665
14:27:08.118 etcs_app_1:   << CM: MSG for Session 0 received
14:27:08.118 etcs_app_1:   << CM: EVENT [SREF:1]
14:27:08.118 etcs_app_1:   !! GUI Session established: 16665
```

```
14:27:08.118 etcs_app_1:    >> UI: event for service:1
14:27:08.118 etcs_app_1:    >> UI: event for service:2
14:27:08.178 com_mgr_1:     << SA: data msg (SID:16665,TYPE:0)
14:27:08.178 com_mgr_1:     << SA: data msg (SID:16665,TYPE:0)
```

2f Platform monitor ready and all Tasksets are running (STATE -> OK)



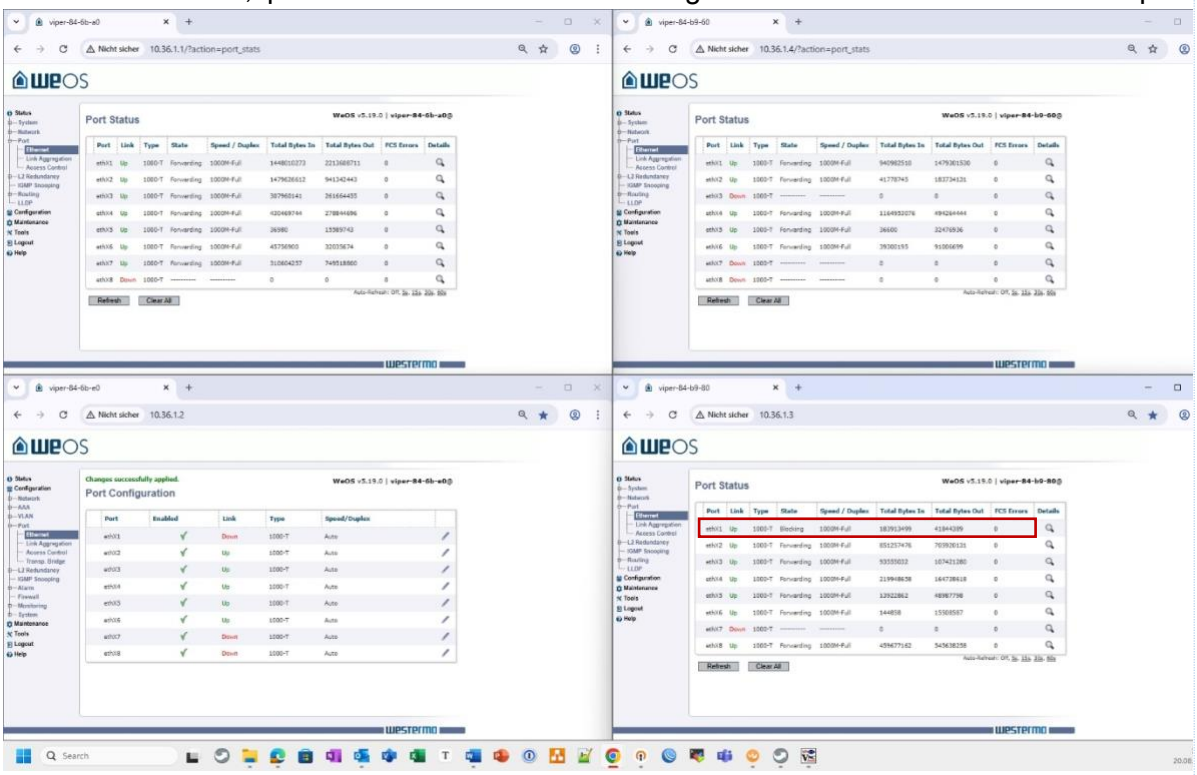
3 4 5 6 Connection to trackside application established. Since the communication bridge is currently running on CE0, only the log from CE0 is shown below.

While sending message 3, the ethernet port was disabled to simulate a broken cable. Thus, the reply for message 3 required more time (i.e. ~1000ms v.s. ~240ms) due to the reconfiguration of the layer 2 “routing”.

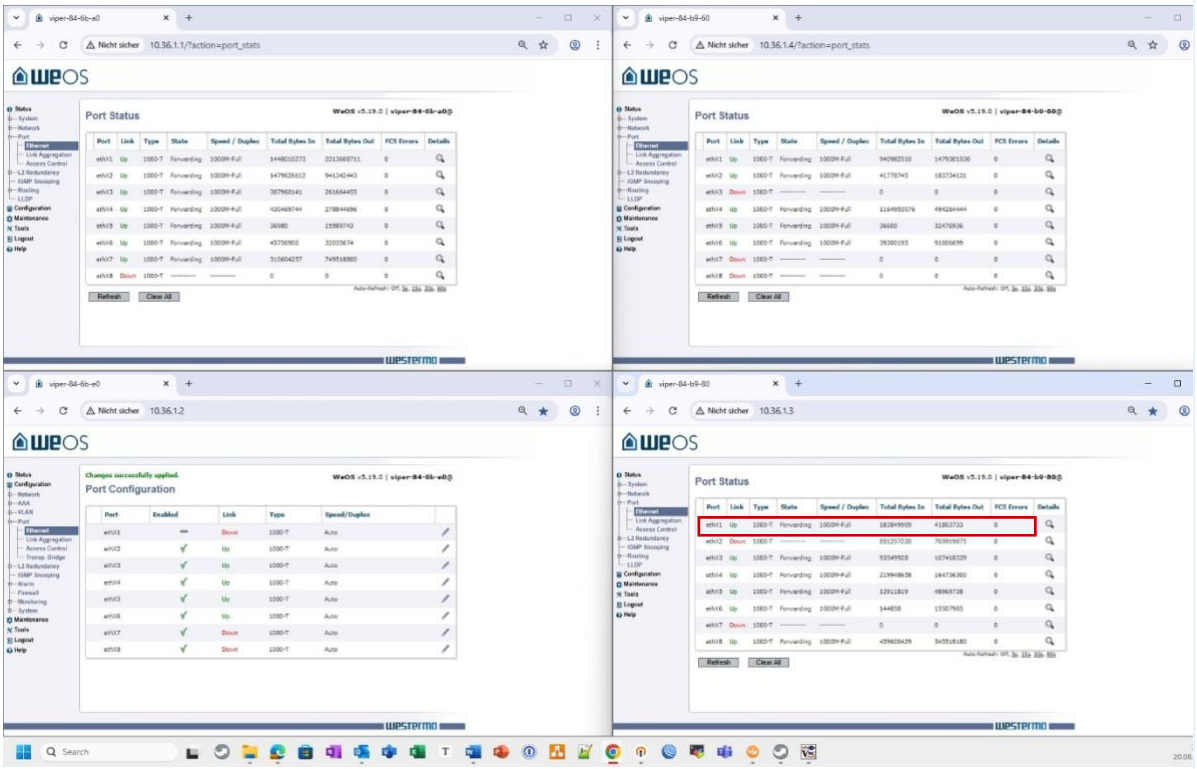
```
15:27:26.484 etcs_app_1:    << CM: MSG for Session 16665 received
15:27:26.484 etcs_app_1:    << UI: Received message: 1
15:27:26.484 etcs_app_1:    >> CM: TS connection request      connect to TS Application
15:27:26.544 com_mgr_1:     << SA: CON REQ to service 1
15:27:26.664 com_mgr_1:     >> connect request via bridge 0 to service 1
15:27:26.664 com_mgr_1:     >> [session reference: 3]
15:27:26.724 com_bri_1_0:   << CG: Handling CTL MSG in bri on CE0
15:27:26.724 com_bri_1_0:   << CG: CTRL MSG Type: REQUEST
15:27:26.724 com_bri_1_0:   !! Handling CONNECT Request
15:27:26.724 com_bri_1_0:   << CG: connect request for TS_SERVICE_1
15:27:26.724 com_bri_1_0:   >> CH: request to Control Handler
15:27:26.724 ctl_hdlr_1_0: << CONNECT request from comm_bridge
15:27:26.724 ctl_hdlr_1_0: >> CONNECT response SUCCESS to comm_bridge
15:27:26.724 ctl_hdlr_1_0: !! Doing Agent connect with timeout = 5s
15:27:26.724 com_bri_1_0:   >> CM: SUCCESS Response from CH
15:27:26.724 com_bri_1_0:   >> CM: SUCCESS message
15:27:26.844 etcs_app_1:    << CM: MSG for Session 0 received
15:27:26.844 etcs_app_1:    !! Connecting TS Service session
15:27:40.258 ctl_hdlr_1_0: !! get remote IP for: rbc1.app.ts
15:27:40.258 ctl_hdlr_1_0: << remote IP: 10.36.20.105
15:27:40.258 com_bri_1_0:   << CH: CONNECTED Event from CH
15:27:40.258 com_bri_1_0:   !! CB: Open socket to IP:10.36.20.105, PORT:12125
15:27:40.284 com_mgr_1:     !! Bridge session established: 7778
15:27:40.284 com_mgr_1:     !! Comm Manager session established: 16666
15:27:40.524 etcs_app_1:    << CM: MSG for Session 0 received
15:27:40.524 etcs_app_1:    << CM: EVENT [SREF:3]
15:27:40.524 etcs_app_1:    !! Trackside Session established: 16666
15:27:40.524 etcs_app_1:    >> UI: event for service:1
15:27:40.584 com_mgr_1:     << SA: data msg (SID:16665,TYPE:0)
15:27:40.764 etcs_app_1:    >> CM: 'TST SEQ 1' to ts (SID:16666)
15:27:40.824 com_mgr_1:     << SA: data msg (SID:16666,TYPE:0)
15:27:41.004 etcs_app_1:    << CM: MSG for Session 16666 received
15:27:41.004 etcs_app_1:    << CM: Received reply 'TST SEQ 1' [240ms]
```

```
15:27:45.804 etcs_app_1:    >> CM: 'TST SEQ 2' to ts (SID:16666)
15:27:45.864 com_mgr_1:     << SA: data msg (SID:16666,TYPE:0)
15:27:46.044 etcs_app_1:    << CM: MSG for Session 16666 received
15:27:46.044 etcs_app_1:    << CM: Received reply 'TST SEQ 2' [239ms]
15:27:50.844 etcs_app_1:    >> CM: 'TST SEQ 3' to ts (SID:16666)     message    3    to    TS
Application
15:27:50.904 com_mgr_1:     << SA: data msg (SID:16666,TYPE:0)
15:27:51.924 etcs_app_1:    << CM: MSG for Session 16666 received
15:27:51.924 etcs_app_1:    << CM: Received reply 'TST SEQ 3' [1080ms]       reply  3  from  TS
Appl.
15:27:55.884 etcs_app_1:    >> CM: 'TST SEQ 4' to ts (SID:16666)
15:27:55.944 com_mgr_1:     << SA: data msg (SID:16666,TYPE:0)
15:27:56.124 etcs_app_1:    << CM: MSG for Session 16666 received
15:27:56.124 etcs_app_1:    << CM: Received reply 'TST SEQ 4' [239ms]
15:28:00.924 etcs_app_1:    >> CM: 'TST SEQ 5' to ts (SID:16666)
15:28:00.984 com_mgr_1:     << SA: data msg (SID:16666,TYPE:0)
15:28:01.164 etcs_app_1:    << CM: MSG for Session 16666 received
15:28:01.164 etcs_app_1:    << CM: Received reply 'TST SEQ 5' [240ms]
15:28:05.964 etcs_app_1:    >> CM: 'TST SEQ 6' to ts (SID:16666)
15:28:06.024 com_mgr_1:     << SA: data msg (SID:16666,TYPE:0)
15:28:06.204 etcs_app_1:    << CM: MSG for Session 16666 received
15:28:06.204 etcs_app_1:    << CM: Received reply 'TST SEQ 6' [240ms]
15:28:11.004 etcs_app_1:    >> CM: 'TST SEQ 7' to ts (SID:16666)
15:28:11.064 com_mgr_1:     << SA: data msg (SID:16666,TYPE:0)
15:28:11.244 etcs_app_1:    << CM: MSG for Session 16666 received
15:28:11.244 etcs_app_1:    << CM: Received reply 'TST SEQ 7' [240ms]
15:28:16.044 etcs_app_1:    >> CM: 'TST SEQ 8' to ts (SID:16666)
15:28:16.104 com_mgr_1:     << SA: data msg (SID:16666,TYPE:0)
15:28:16.284 etcs_app_1:    << CM: MSG for Session 16666 received
15:28:16.284 etcs_app_1:    << CM: Received reply 'TST SEQ 8' [240ms]
15:28:20.364 etcs_app_1:    << CM: MSG for Session 16665 received
15:28:20.364 etcs_app_1:    << UI: Received message: 1
15:28:20.364 etcs_app_1:    >> CM: TS disconnect request
15:28:20.424 com_mgr_1:     << SA: DIS REQ for session 16666
15:28:20.544 com_mgr_1:     >> CG: DIS REQ via CB 0 to SVC 1 [SREF: 3]
15:28:20.604 com_bri_1_0:   << CG: Handling CTL MSG in bri on CE0
15:28:20.604 com_bri_1_0:   << CG: CTRL MSG Type: REQUEST
15:28:20.604 com_bri_1_0:   !! Handling DISCONNECT Request
15:28:20.604 com_bri_1_0:   !! CB: Disconnecting Session:7778
15:28:20.604 com_bri_1_0:   >> CH: Disconnecting service: TS_SERVICE_1
15:28:20.604 ctl_hdlr_1_0: << DISCONNECT request from comm_bridge
15:28:20.604 com_bri_1_0:   >> CM: SUCCESS Response from CH
15:28:20.604 com_bri_1_0:   >> CM: SUCCESS message
15:28:20.724 etcs_app_1:    << CM: MSG for Session 0 received
15:28:20.724 etcs_app_1:    !! Connecting TS Service session
15:28:21.084 etcs_app_1:    >> CM: 'TST SEQ 9' to ts (SID:16666)
15:28:21.144 com_mgr_1:     << SA: data msg (SID:16666,TYPE:0)
15:28:21.204 com_bri_1_0:   !! CB: Dropping MSG as connection not established
15:28:21.604 com_bri_1_0:   << CH: DISCONNECTED Event from CH
15:28:21.604 com_bri_1_0:   !! Bridge disconnected from Service
15:28:21.624 com_mgr_1:     !! Bridge session disconnected: 7778
15:28:21.624 com_mgr_1:     !! Comm Manager session disconnected: 16666
15:28:21.684 etcs_app_1:    << CM: MSG for Session 0 received
15:28:21.684 etcs_app_1:    << CM: EVENT [SREF:3]
15:28:21.684 etcs_app_1:    !! Trackside Session disconnected: 16666
15:28:21.684 etcs_app_1:    >> UI: event for service:1
15:28:21.744 com_mgr_1:     << SA: data msg (SID:16665,TYPE:0)
```

In the initial state, port X1 on switch 3 is blocking the traffic to avoid network loops.



A soon, as port X1 on switch 2 is down, the port X1 of switch 3 is reconfigured to now forward the traffic to switch 4 and to assure the network is still operational.



Log of trackside application

```
15:27:40.272 New connection accepted
15:27:40.409 << HBT REQ from com_mgr_1@CE0
15:27:40.409 >> HBT ACK to com_mgr_1@CE0
15:27:40.896 << MSG RCV 'TST SEQ 1 [etcs_app_1]'
15:27:40.896 >> MSG ACK 'TST SEQ 1 [etcs_app_1]'
15:27:41.006 << HBT REQ from com_mgr_1@CE0
15:27:41.006 >> HBT ACK to com_mgr_1@CE0
15:27:41.616 << HBT REQ from com_mgr_1@CE0
15:27:41.616 >> HBT ACK to com_mgr_1@CE0
.
.
.
15:27:45.809 << HBT REQ from com_mgr_1@CE0
15:27:45.809 >> HBT ACK to com_mgr_1@CE0
15:27:45.927 << MSG RCV 'TST SEQ 2 [etcs_app_1]'
15:27:45.927 >> MSG ACK 'TST SEQ 2 [etcs_app_1]'
15:27:46.410 << HBT REQ from com_mgr_1@CE0
15:27:46.410 >> HBT ACK to com_mgr_1@CE0
15:27:47.019 << HBT REQ from com_mgr_1@CE0
15:27:47.019 >> HBT ACK to com_mgr_1@CE0
.
.
.
15:27:50.609 << HBT REQ from com_mgr_1@CE0
15:27:50.610 >> HBT ACK to com_mgr_1@CE0
15:27:51.838 << MSG RCV 'TST SEQ 3 [etcs_app_1]'     Test message 3 from OB Application
15:27:51.838 >> MSG ACK 'TST SEQ 3 [etcs_app_1]'
15:27:51.841 << HBT REQ from com_mgr_1@CE0
15:27:51.841 >> HBT ACK to com_mgr_1@CE0
.
.
.
15:27:56.013 << HBT REQ from com_mgr_1@CE0
15:27:56.013 >> HBT ACK to com_mgr_1@CE0
15:27:56.015 << MSG RCV 'TST SEQ 4 [etcs_app_1]'
15:27:56.015 >> MSG ACK 'TST SEQ 4 [etcs_app_1]'
15:27:56.607 << HBT REQ from com_mgr_1@CE0
15:27:56.607 >> HBT ACK to com_mgr_1@CE0
15:27:57.209 << HBT REQ from com_mgr_1@CE0
15:27:57.209 >> HBT ACK to com_mgr_1@CE0
.
.
.
15:28:00.812 << HBT REQ from com_mgr_1@CE0
15:28:00.812 >> HBT ACK to com_mgr_1@CE0
15:28:01.053 << MSG RCV 'TST SEQ 5 [etcs_app_1]'
15:28:01.053 >> MSG ACK 'TST SEQ 5 [etcs_app_1]'
15:28:01.409 << HBT REQ from com_mgr_1@CE0
15:28:01.409 >> HBT ACK to com_mgr_1@CE0
15:28:02.009 << HBT REQ from com_mgr_1@CE0
.
.
```

Status	Pass
Comments	The reconfiguration of the switches through the RSTP can introduce some delay into the message exchange and needs to be considered accordingly in the application protocols.
Attachments	

3.34 Allow Management Access to Onboard Components [2.7.3]

Description User Story	As R2DATO WP36, we want to have management access to all onboard components remotely or physically, so that we can demonstrate the update of configurations or software.
Annotation	Omitted: This User Story became obsolete because in the daily work and during development and deployment the team accessed the OBU permanently (VPN access, LAN setup as described in WP36.3).

3.35 Allow Ease of Access to the Network [2.7.4]

Description User Story	As R2DATO WP36, we want to allow plug-and-playability of components after an initial identity and access management (IAM) configuration, so that we can demonstrate an easy reconfiguration when a component is added or removed.
Annotation	Omitted: This User Story is not demonstrated due to cost and proprietary platform dependency.

3.36 Supply List of Onboard Network Components [2.7.5]

Description User Story	As R2DATO WP36, we want to automatically supply a list of all configured onboard components, so that we can demonstrate maintainable configurations.
Annotation	Omitted: This User Story is not demonstrated due to cost and proprietary platform dependency.

3.37 Provision of Date Time Reference [2.7.6]

Description User Story	As R2DATO WP36, we want to have a common date time reference for all onboard components, so that we can demonstrate a shared service crucial for reliable information sharing.
Annotation	Fulfilled in the Lab using the NTP (see architecture).

3.38 Prioritise Specific Network Traffic [2.7.7]

Description User Story	As R2DATO WP36, we want to configure different quality of service classes in the network, so that we can demonstrate the prioritisation of specific network traffic.

Annotation	The demonstration of this User Story has been delayed and the test result will be reported in the next deliverable 36.5.

3.39 MONITOR THE NETWORK HEALTH STATUS / GUARANTEED CHARACTERISTICS [2.7.8]

Description User Story	As R2DATO WP36, we want to monitor the network health status, so that we can demonstrate how the guaranteed network characteristics can be measured.
Annotation	Covered by User Story [2.7.7]

3.40 REALIZE AUTHENTICATION AND AUTHORIZATION MECHANISMS [2.8.1]

Description User Story	As R2DATO WP36, we want to realize authorization and authentication mechanisms for the nodes in the network, so that we can demonstrate a measure to ensure the integrity of the sent data.
Annotation	Theoretical analysis within chapter 2.3 IT/OT Security.

3.41 CHANGE IDENTITY AND ACCESS MANAGEMENT SETTINGS [2.8.2]

Description User Story	As R2DATO WP36, we want to change Identity and Access Management (IAM) settings of the modular computing platform, so that we can demonstrate the possibility of reacting to security threats.
Annotation	Theoretical analysis within chapter 2.3 IT/OT Security.

3.42 ENCRYPT ALL DATA ON THE NETWORK [2.8.3]

Description User Story	As R2DATO WP36, we want to encrypt all data that is exchanged between the components on the onboard network, so that we can demonstrate the feasibility of secure communication over grey channels.
Annotation	Theoretical analysis within chapter 2.3 IT/OT Security.

3.43 REALIZE SECURE COMMUNICATION OVER FRMCS [2.8.4]

Description User Story	As R2DATO WP36, we want to realize a secure communication over FRMCS, so that we can demonstrate the feasibility of safe communication over FRMCS as grey channel.
Annotation	Theoretical analysis within chapter 2.3 IT/OT Security.

3.44 Separation of Network Zones [2.8.5]

Description User Story	As R2DATO WP36, we want to separate network zones and conduits according to the security levels of the applications, so that we can demonstrate the virtual separation of applications of different security levels.
Annotation	Theoretical analysis within chapter 2.3 IT/OT Security

3.45 Detect Unusual Network Traffic [2.8.6]

Description User Story	As R2DATO WP36, we want to receive notifications when unusual traffic goes through the network, so that we can demonstrate the detection of unusual traffic.
Annotation	Theoretical analysis within chapter 2.3 IT/OT Security.

3.46 Detect Unauthorised Network Device [2.8.7]

Description User Story	As R2DATO WP36, we want to receive alarms when unauthorised devices are added to the network, so that we can demonstrate the detection of unauthorized devices.
Annotation	Theoretical analysis within chapter 2.3 IT/OT Security.

3.47 Provide Safe Access Train Hardware via a Functional Vehicle Adapter [2.9.1]

Description User Story	As R2DATO WP36, we want the safety layer / runtime environment to run on diverse hardware, so that we can demonstrate future upgrade / migration scenarios.
Annotation	Omitted: This User Story is not demonstrated because FVA is out of scope for WP36.

3.48 Provide Access to Train Hardware via a TCMS Data Service [2.9.2]

Description User Story	As R2DATO WP36, we want to integrate an application on the modular computing platform with a TCMS Data Service (as defined in https://eurospec.eu/download/tcms-dataservices-1-0), so that we can demonstrate the abstraction of train hardware for basic integrity applications.
Annotation	This User Story is fulfilled and demonstrated with User Story [2.5.10]. The TCMS-DS is explained in chapter 2.5.4. It has to be noted that in the lab setup the TCMS_DS itself is not deployed on the CEs of the Modular Platform, but DIA-VEC is (please refer to chapter 2.5.1).

3.49 EXECUTE A SIMPLIFIED ATO CONTROL LOOP [2.10.1]

Description User Story	As R2DATO WP36, we want to host a simplified ATO onboard application on the safety layer / runtime environment executing an ATO control loop, so that we can demonstrate a basic integrity application communicating over FRMCS and interacting with the TCMS.
Annotation	Omitted: This User Story is not demonstrated but the intended behaviour is shown in diagnostics User Stories.

3.50 EXECUTE A SIMPLIFIED ETCS CONTROL LOOP [2.10.2]

Description User Story	As R2DATO WP36, we want to host a simplified ETCS onboard application on the safety layer / runtime environment executing an ETCS control loop, so that we can demonstrate a safe application communicating over FRMCS and interacting with the TCMS.

Test Case Title	**Drive simulated Train in Lab**
Test Case ID	2.10.2-A
Description	Run a train on an ETCS L2 track communicating over FMRCS with a simulated RBC.
Test Environment	• DMI Application running on a VM • Modular Platform (TAS Platform) • CCS Communication Network (CCN) • Testbench VMs running on the Lab Server • FRMCS trackside Gateway on a VM • Physical FRMCS onboard Gateway • ETCS Application running on the Modular Platform For this user story, an existing ETCS Application was ported to the TAS Platform. This ETCS application is normally used in the validation of ERTMS implementations. The ETCS Application is running in a redundant 2oo3 setup on all 3 computing elements (3 separate computers). For redundancy, the communication bridge for the FRMCS communication with the trackside application (RBC) is running on the two computing elements ce1 and ce2.

All remaining peripheral communication with the Testbench (e.g. odometry, TUI interface, Balise simulator), is running on the first computing element (ce0). This approach was chosen to reduce the implementation effort.

As a result of this setup, ce0 is essential for the overall onboard system and cannot be used to demonstrate the recovery process of the onboard system. Therefore, only ce1 and ce2 can be used to demonstrate such recovery scenarios of the computing platform.

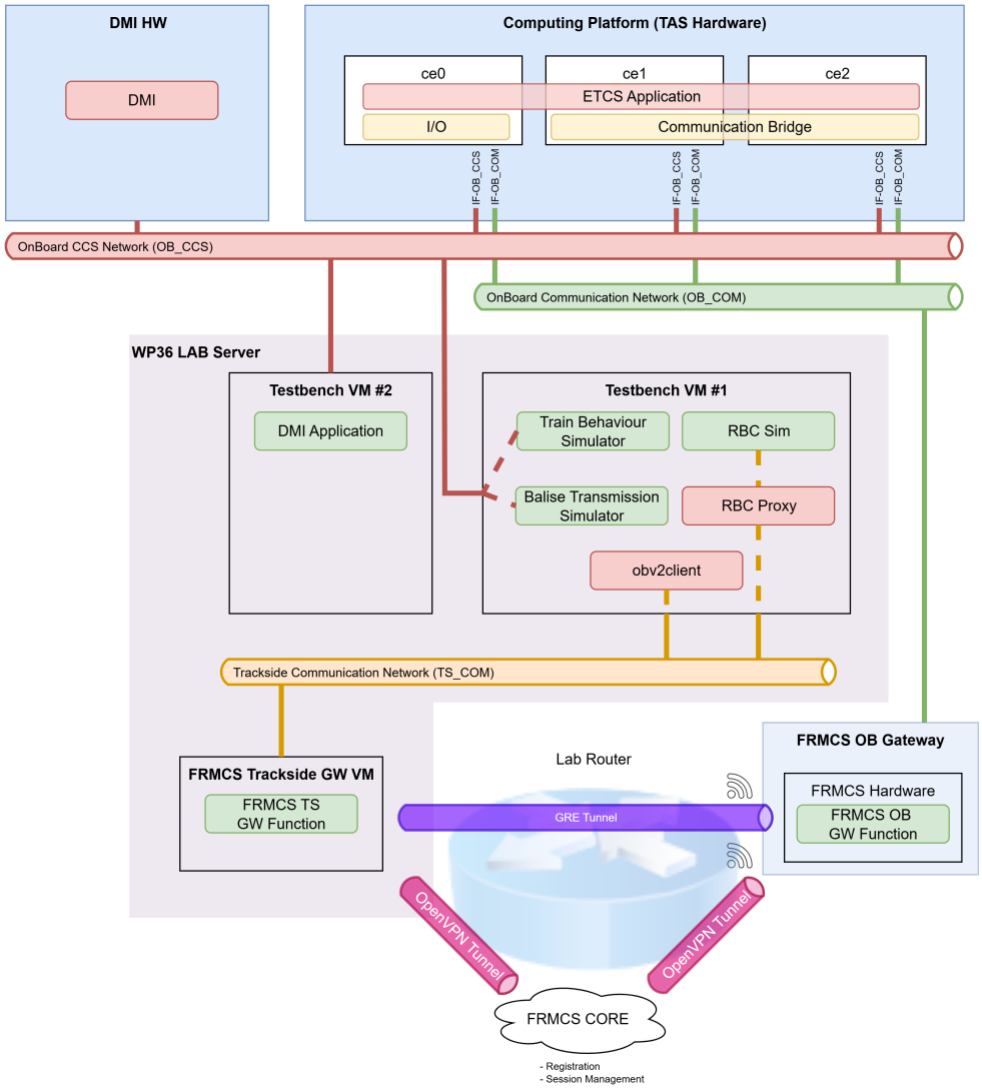


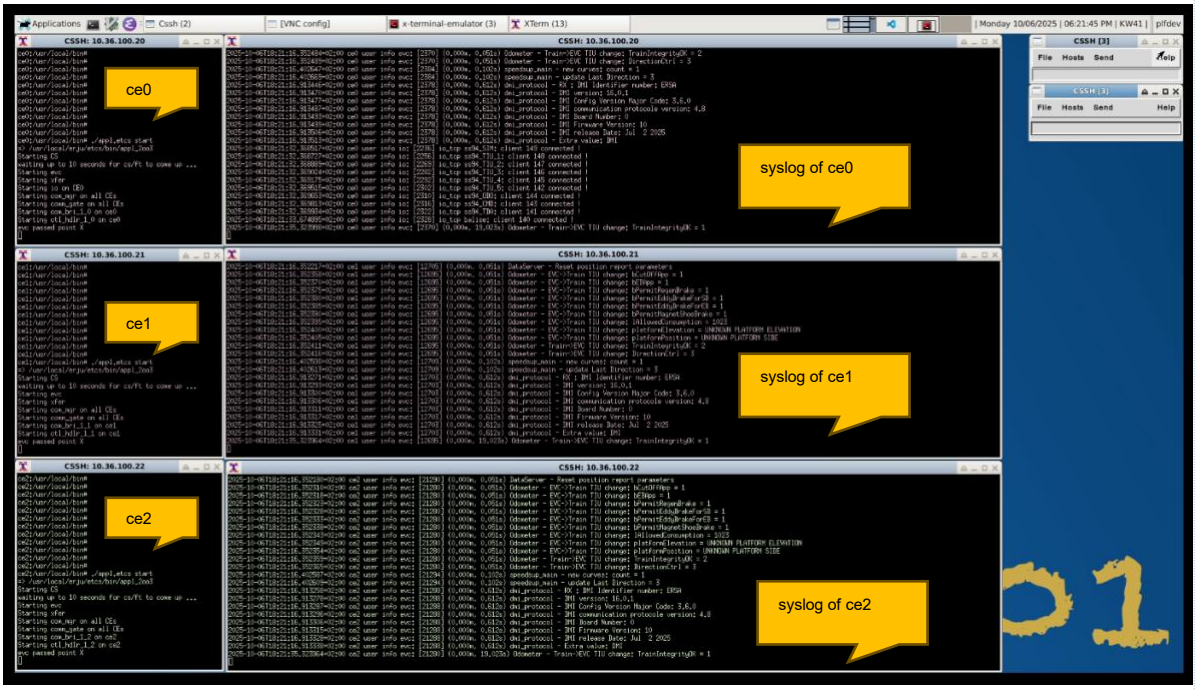
Figure 26 Setup

Preconditions	Set up Test environment in a manner that runs long enough for this Test Case.
Test Steps	14. Prepare testbench (FRMCS and DMI) a. Start FRMCS onboard gateway

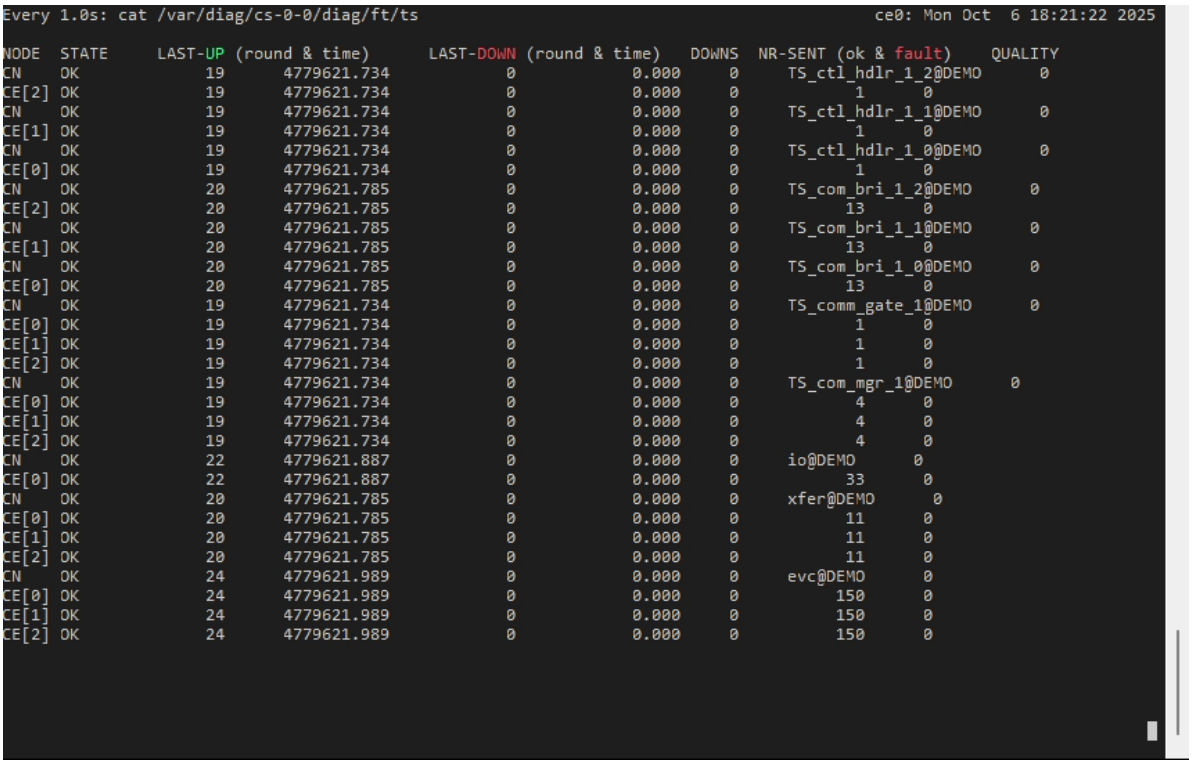
	b. Start FRMCS trackside gateway c. Register trackside application to trackside GW d. Start DMI Application 15. Start testbench components and monitoring a. Start the onboard safe application (ETCS) b. Start platform monitor c. Start the testbench simulators using the scenario controller 16. Open the train driver cab 17. Start mission on DMI 18. Drive train 19. Stop train 20. Close cabin
Test Data	None.
Expected Result	During the full test, the system must continue to perform its intended functions, without any detectable issues or degradation.

Actual Result	
1	e) FRMCS onboard gateway is running and connected to the FRMCS Core f) FRMCS trackside gateway is running and connected to the FRMCS Core g) The trackside application “rbc3.app.ts” is registered to the FRMCS Core h) The DMI Applications is running and ready to accept connections from the ETCS application

2	d) The ETCS application is started on the modular computing platform consisting of 3 discrete computing elements (ce0, ce1 and ce2). Next to each console of the individual computing elements (ce), the syslog output is shown.



e) The platform monitor provides the status of all Tasksets running on the different computing elements.

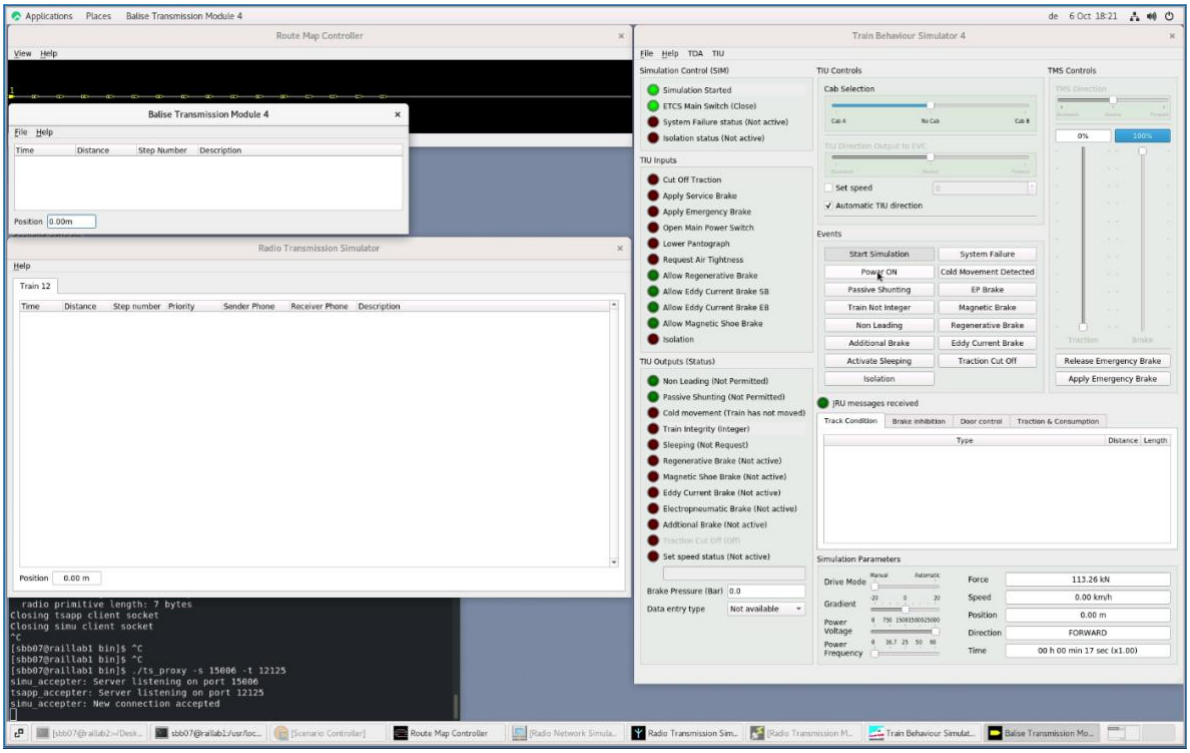


```
Every 1.0s: cat /var/diag/cs-0-0/diag/ft/ts                                  ce0: Mon Oct  6 18:21:22 2025

NODE  STATE     LAST-UP (round & time)      LAST-DOWN (round & time)   DOWNS  NR-SENT (ok & fault)     QUALITY
CN    OK             19      4779621.734           0           0.000     0    TS_ctl_hdlr_1_2@DEMO      0
CE[2] OK             19      4779621.734           0           0.000     0          1      0
CN    OK             19      4779621.734           0           0.000     0    TS_ctl_hdlr_1_1@DEMO      0
CE[1] OK             19      4779621.734           0           0.000     0          1      0
CN    OK             19      4779621.734           0           0.000     0    TS_ctl_hdlr_1_0@DEMO      0
CE[0] OK             19      4779621.734           0           0.000     0          1      0
CN    OK             20      4779621.785           0           0.000     0    TS_com_bri_1_2@DEMO       0
CE[2] OK             20      4779621.785           0           0.000     0         13      0
CN    OK             20      4779621.785           0           0.000     0    TS_com_bri_1_1@DEMO       0
CE[1] OK             20      4779621.785           0           0.000     0         13      0
CN    OK             20      4779621.785           0           0.000     0    TS_com_bri_1_0@DEMO       0
CE[0] OK             20      4779621.785           0           0.000     0         13      0
CN    OK             19      4779621.734           0           0.000     0    TS_comm_gate_1@DEMO       0
CE[0] OK             19      4779621.734           0           0.000     0          1      0
CE[1] OK             19      4779621.734           0           0.000     0          1      0
CE[2] OK             19      4779621.734           0           0.000     0          1      0
CN    OK             19      4779621.734           0           0.000     0    TS_com_mgr_1@DEMO       0
CE[0] OK             19      4779621.734           0           0.000     0          4      0
CE[1] OK             19      4779621.734           0           0.000     0          4      0
CE[2] OK             19      4779621.734           0           0.000     0          4      0
CN    OK             22      4779621.887           0           0.000     0    io@DEMO       0
CE[0] OK             22      4779621.887           0           0.000     0         33      0
CN    OK             20      4779621.785           0           0.000     0    xfer@DEMO       0
CE[0] OK             20      4779621.785           0           0.000     0         11      0
CE[1] OK             20      4779621.785           0           0.000     0         11      0
CE[2] OK             20      4779621.785           0           0.000     0         11      0
CN    OK             24      4779621.989           0           0.000     0    evc@DEMO       0
CE[0] OK             24      4779621.989           0           0.000     0        150      0
CE[1] OK             24      4779621.989           0           0.000     0        150      0
CE[2] OK             24      4779621.989           0           0.000     0        150      0
```

Taskset	Description
TS_ctl_hdlr_1_n@DEMO	The control handler controls the communication setup ove FRMCS using the OB_{App} Interface. On each ce, there is a separate instance running, but onl one instance is actively controlled by the communicatio manager (TS_com_mgr).
TS_com_bri_1_n@DEMO	The communication bridge opens the socket connection t the trackside application (RBC) after the control handle successfully initiates the communication session. On each ce, there is a separate instance running, but onl one instance is actively controlled by the communicatio manager (TS_com_mgr).
TS_comm_gate_1@DEMO	Taskset is required to transfer messages between the saf and non-safe parts of the platform.
TS_com_mgr_1@DEMO	The communication manager is responsible for managin and monitor the communication sessions. The communication manager monitors the communicatio bridges and switches to the next bridge, if the current one i not available anymore.

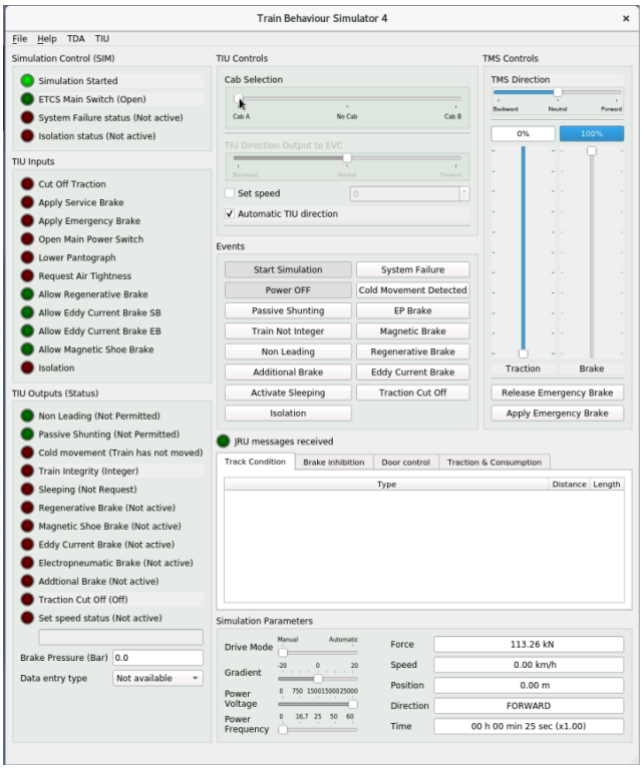
io@DEMO	The io Taskset implements all peripheral communication with the testbench. It is running only on ce0 and therefore ce0 cannot be used to demonstrate recovery scenarios.
xfer@DEMO	Taskset is required to transfer messages between the safe and non-safe parts of the platform.
evc@DEMO	The ETCS application runs in a 2oo3 setup.

f) The Test Testbench is running with all required applications

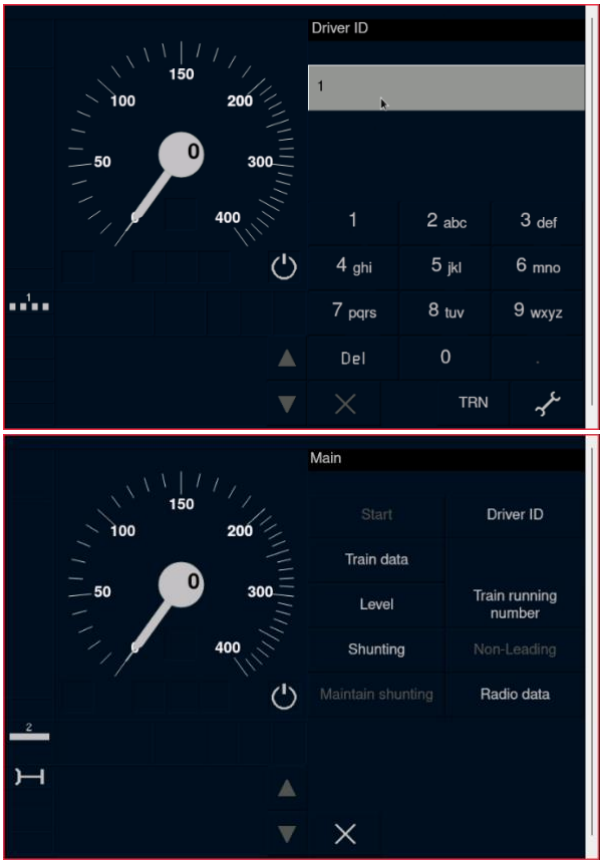


- The Route Map Controller, showing the balise positions on the track and the current position of the train.
- The Balise Transition Module, sending balise telegrams to the ETCS application, when the train reaches the position of a balise
- The Radio Transmission Simulator simulates the RBC. It is triggered by messages received from the onboard unit and provides the corresponding answers to the ETCS application running on the modular computing platform.
- The Train Behaviour Simulator, is used to manage the active cabin, to control the traction and brakes as well as other train related signals and indications

3	Open the train driver CAB



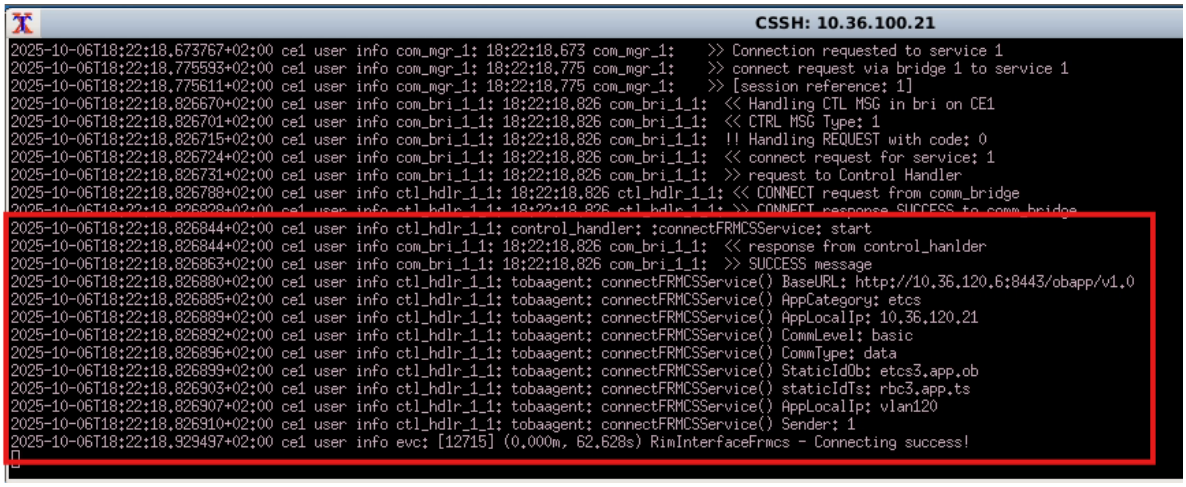
The DMI is activated, prompting for driver ID and Train Data.



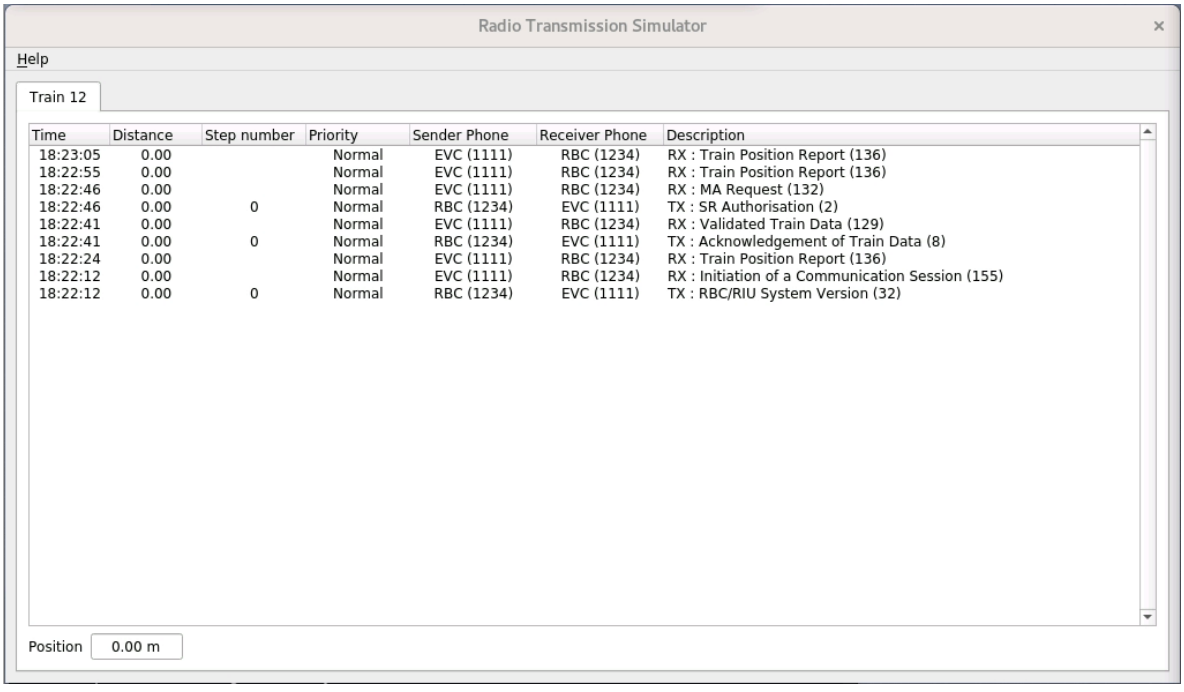
4	Start mission on DMI

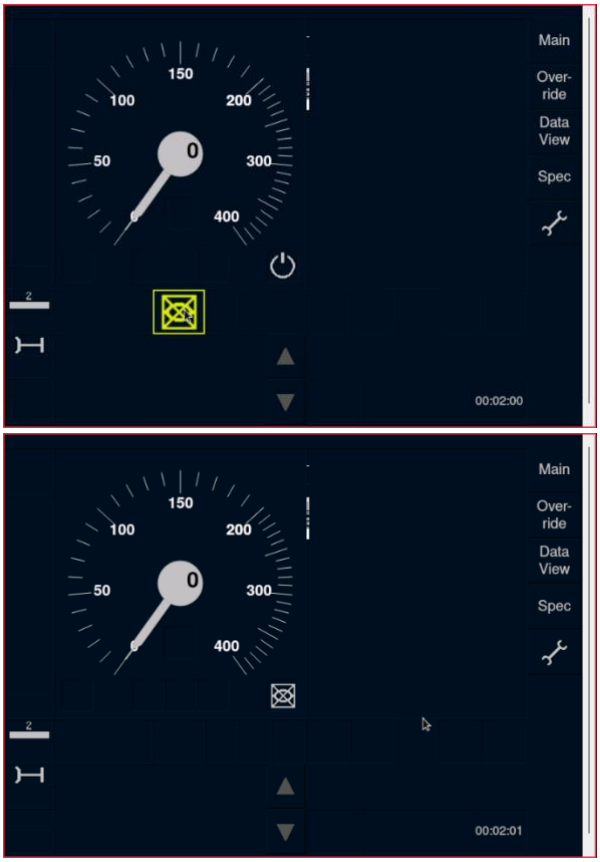
After activating the cabin, the ETCS Application prompts for required train data.

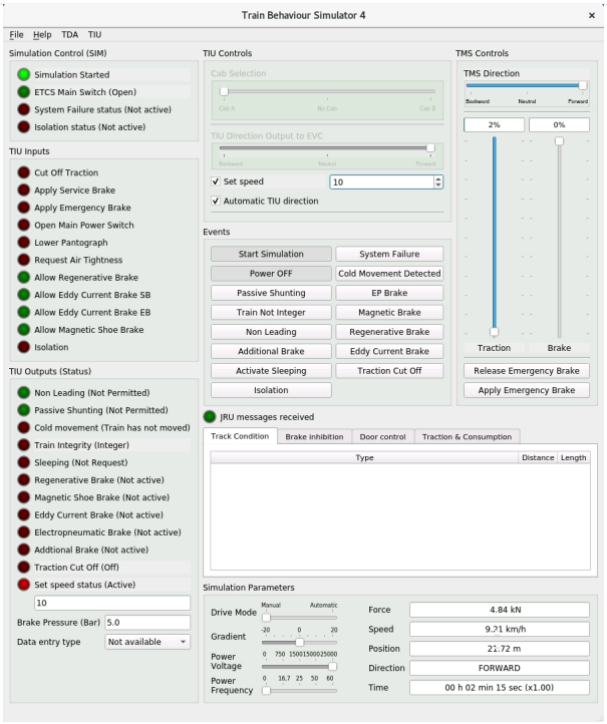
- Driver ID
- RBC data – after entering the RBC data, the ETCS Application establishes a connection to the RBC using FRMCS.



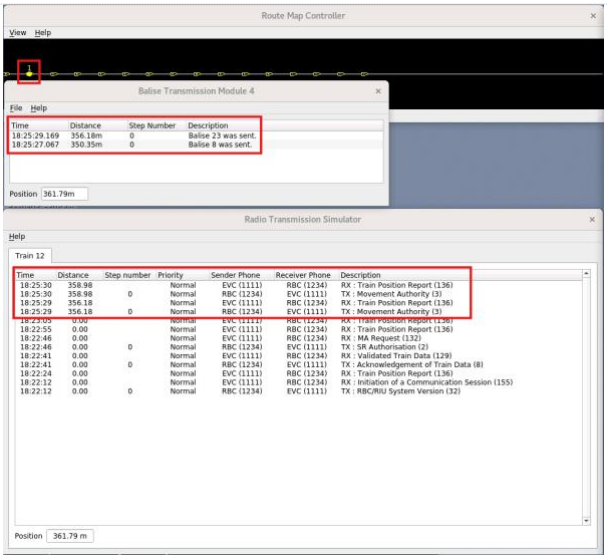
- The connection to the RBC is established and the ETCS application requests the driver to acknowledge SR (Staff Responsible) mode.

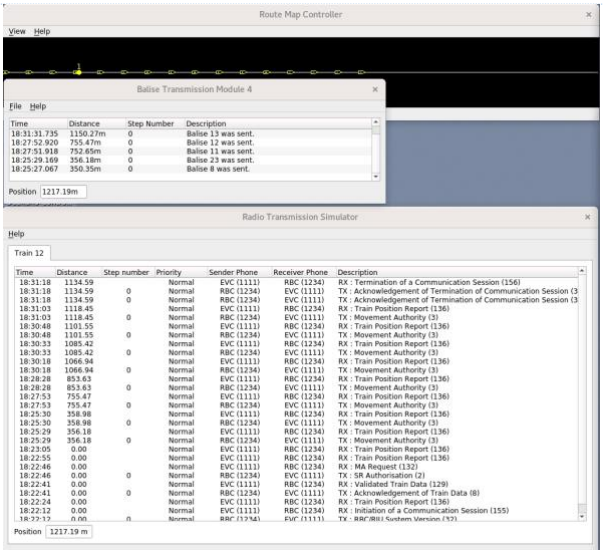


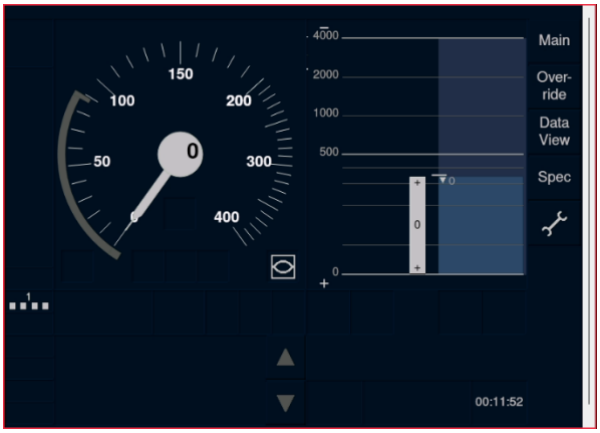
	
5	Drive train • Release emergency brake, release brake and apply traction



The train passes the 1st Balise and switches to FS (Full Supervision) mode

	 Route Map Controller View Help Balise Transmission Module 4 File Help Time Distance Step Number Description 18:25:29.169 356.18m 0 Balise 23 was sent. 18:25:27.067 350.35m 0 Balise 8 was sent. Position 361.79m Radio Transmission Simulator Help Train 12 Time Distance Step number Priority Sender Phone Receiver Phone Description 18:25:30 358.98 Normal EVC (1111) RBC (1234) RX : Train Position Report (136) 18:25:30 358.98 0 Normal RBC (1234) EVC (1111) TX : Movement Authority (3) 18:25:29 356.18 Normal EVC (1111) RBC (1234) RX : Train Position Report (136) 18:25:29 356.18 0 Normal RBC (1234) EVC (1111) TX : Movement Authority (3) 18:22:55 0.00 Normal EVC (1111) RBC (1234) RX : Train Position Report (136) 18:22:46 0.00 Normal EVC (1111) RBC (1234) RX : MA Request (132) 18:22:46 0.00 0 Normal RBC (1234) EVC (1111) TX : SR Authorisation (2) 18:22:41 0.00 Normal EVC (1111) RBC (1234) RX : Validated Train Data (129) 18:22:41 0.00 0 Normal RBC (1234) EVC (1111) TX : Acknowledgement of Train Data (8) 18:22:24 0.00 Normal EVC (1111) RBC (1234) RX : Train Position Report (136) 18:22:12 0.00 Normal EVC (1111) RBC (1234) RX : Initiation of a Communication Session (155) 18:22:12 0.00 0 Normal RBC (1234) EVC (1111) TX : RBC/RIU System Version (32) Position 361.79 m 150 100 200 50 10 300 0 400 4000 2000 1000 500 0 Main Over-ride Data View Spec 00:04 Entering FS 00:04:35
6	Stop train





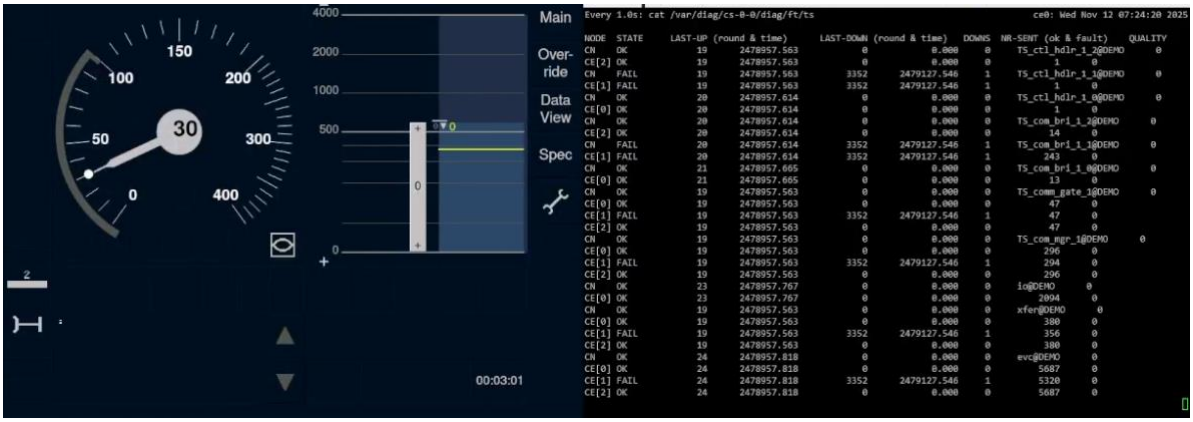
After driving 1.2km the train is stopped

Status	Pass
Comments	…
Attachments	

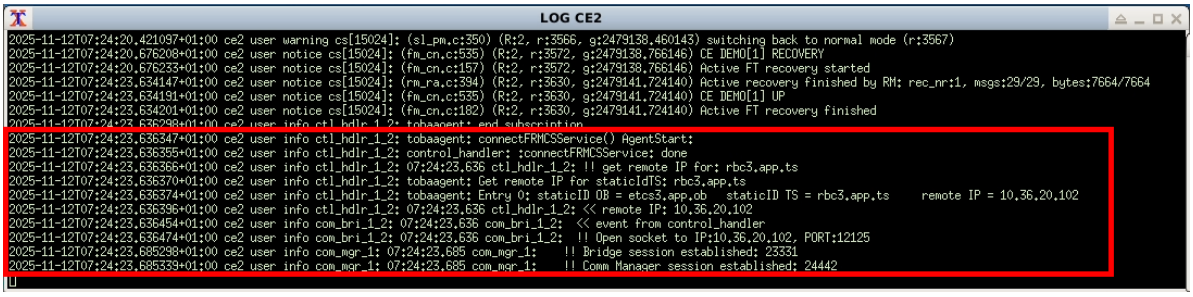
Test Case Title	**Drive simulated Train in Lab – with failing replica**
Test Case ID	2.10.2-B
Description	Run a train on an ETCS L2 track communicating over FMRCS with a simulated RBC.
Test Environment	• DMI Application running on a VM • Modular Platform (TAS Platform) • CCS Communication Network (CCN) • Testbench VMs running on the Lab Server

	• FRMCS trackside Gateway on a VM • Physical FRMCS onboard Gateway • ETCS Application running on the Modular Platform For this user story, we use the same setup as for Test Case ID 2.10.2-A and will demonstrate that the safe application continues to work while a replica of the modular computing platform fails.
Preconditions	Set up Test environment in a manner that runs long enough for this Test Case.
Test Steps	Remark: Steps *1-5* and *8-10* are the same as in Test Case ID 2.10.2-A and will therefore not be documented. 1. *Prepare testbench (FRMCS and DMI)* a. *Start FRMCS onboard gateway* b. *Start FRMCS trackside gateway* c. *Register trackside application to trackside GW* d. *Start DMI Application* 2. *Start testbench components and monitoring* a. *Start the onboard safe application (ETCS)* b. *Start platform monitor* c. *Start the testbench simulators using the scenario controller* 3. *Open the train driver cab* 4. *Start mission on DMI* 5. *Drive train* 6. Stop ETCS Application on the replica, ce1 (computing element 1 of the platform) 7. Demonstrate recovery and re-integration of ce1. 8. *Stop train* 9. *Close cabin*
Test Data	None.
Expected Result	During the full test, the system must continue to perform its intended functions, without any detectable issues or degradation.
Actual Result	

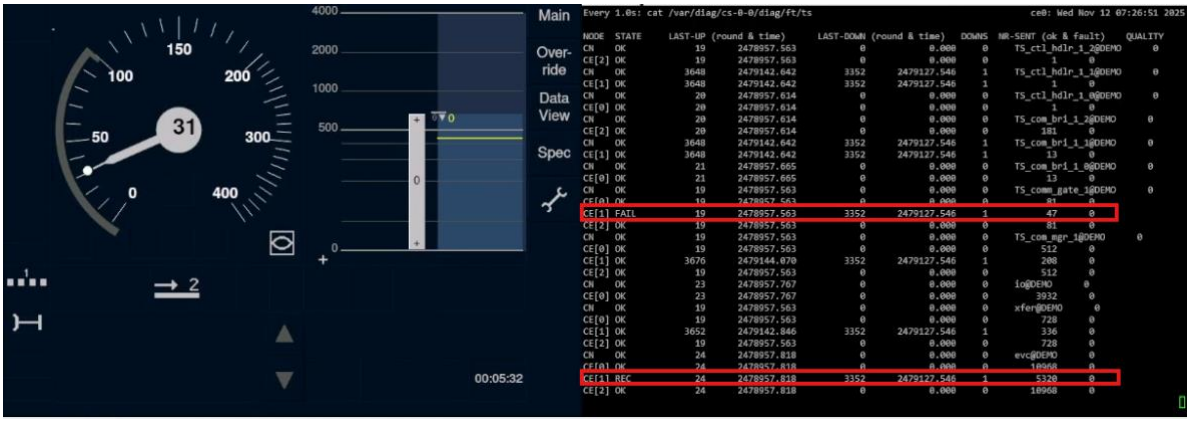
1-5	Refer to Test Case ID 2.10.2-A
6	a) The ETCS application on ce1 is manually stopped to simulate a failure on ce1. The task monitor (right window) is reporting the failing replica ce1. The ETCS application continues to work as a 2oo2 setup until the recovery is completed.



b) Since the FRMCS communication gateway was running on ce1, the communication is automatically re-established by ce2.



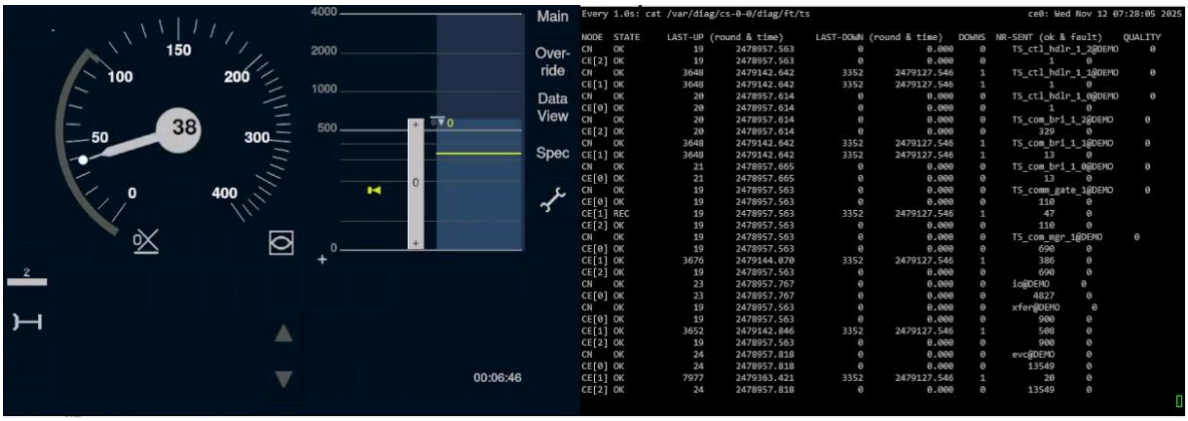
7	a) While the recovery of ce1 is in progress, the ETCS application is still working as expected. At the current stage, the ETCS application is handling an ETCS L1->L2 transition initiated by the trackside balises.



b) Recovery is completed after approx. 4 minutes. During recovery, the ETCS application successfully managed the following ETCS tasks:

- L2 -> L1 transition
- RBC connection release

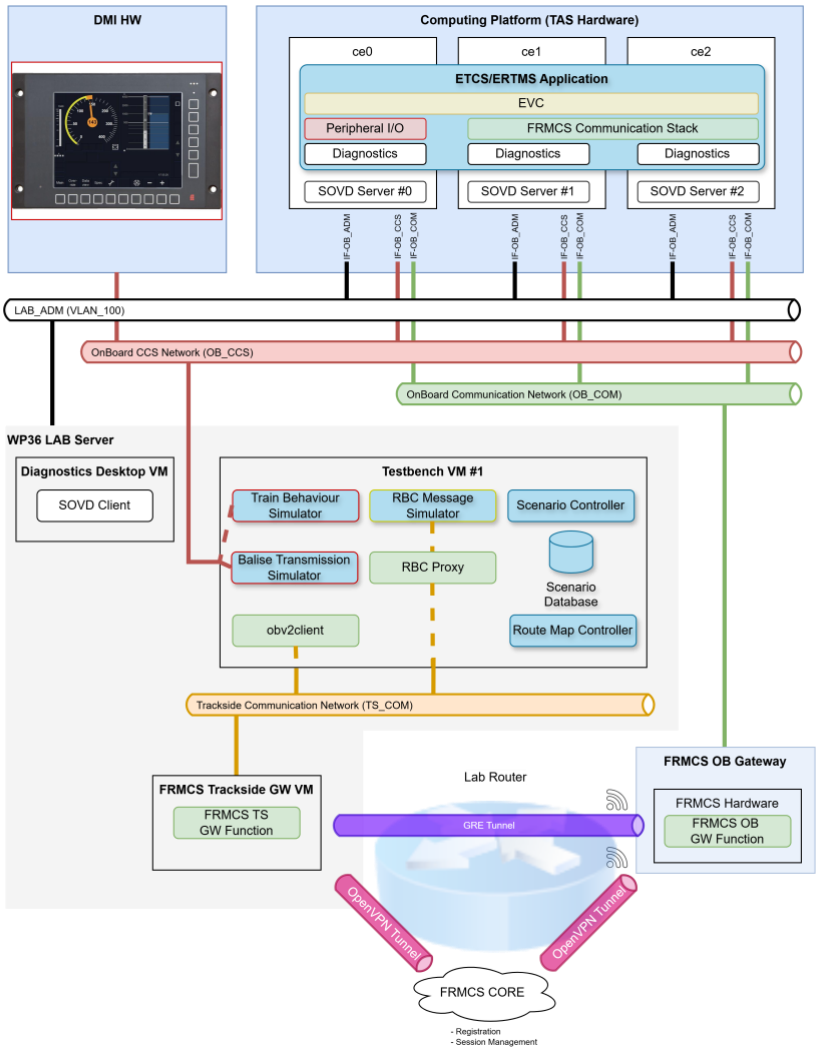
- L1 -> L2 transition (incl. RBC connection setup over FRMCS)



8-9	Refer to Test Case ID 2.10.2-A

Status	Pass
Comments	The current implementation of the ETCS application was not optimized for recovery and therefore, the recovery process in this scenario took approximately 4 minutes. This behaviour can be optimized by reducing the amount of memory to be recovered.
Attachments	

Test Case Title	Drive simulated Train in Lab – including Diagnostics
Test Case ID	2.10.2-C
Description	Run a train on an ETCS L2 track, communicating over FMRCS with a simulated RBC and provide diagnostics information through a basic integrity application to the diagnostics client running on a desktop.
Test Environment	• DMI Application running on a VM • Modular Platform (TAS Platform) • CCS Communication Network (CCN) • Testbench VMs running on the Lab Server • FRMCS trackside Gateway on a VM • Physical FRMCS onboard Gateway • ETCS Application running on the Modular Platform • Diagnostics client running on a desktop VM For this user story, we use the setup from Test Case ID 2.10.2-A extended with the diagnostics functionality running as basic integrity application on the computing elements of the safe modular computing platform. The diagnostics information is collected by the SOVD server and retrieved by the diagnostics client running on a desktop.



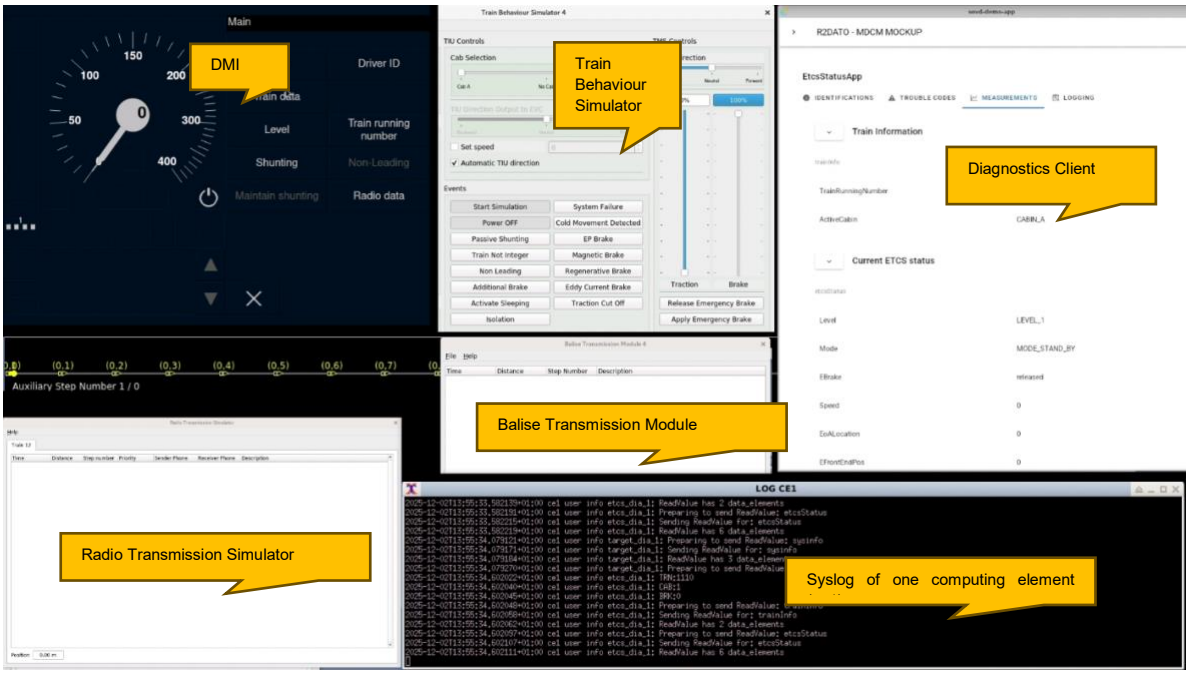
Preconditions	Set up Test environment in a manner that runs long enough for this Test Case.
Test Steps	Remark: Some steps are the same as in the previous Test Cases and will therefore not be documented. 1. Prepare testbench (FRMCS and DMI) *a. Start FRMCS onboard gateway* *b. Start FRMCS trackside gateway* *c. Register trackside application to trackside GW* *d. Start DMI Application* e. Start Diagnostics Client 2. Start testbench components and monitoring *a. Start the onboard safe application (ETCS)* *b. Start platform monitor*

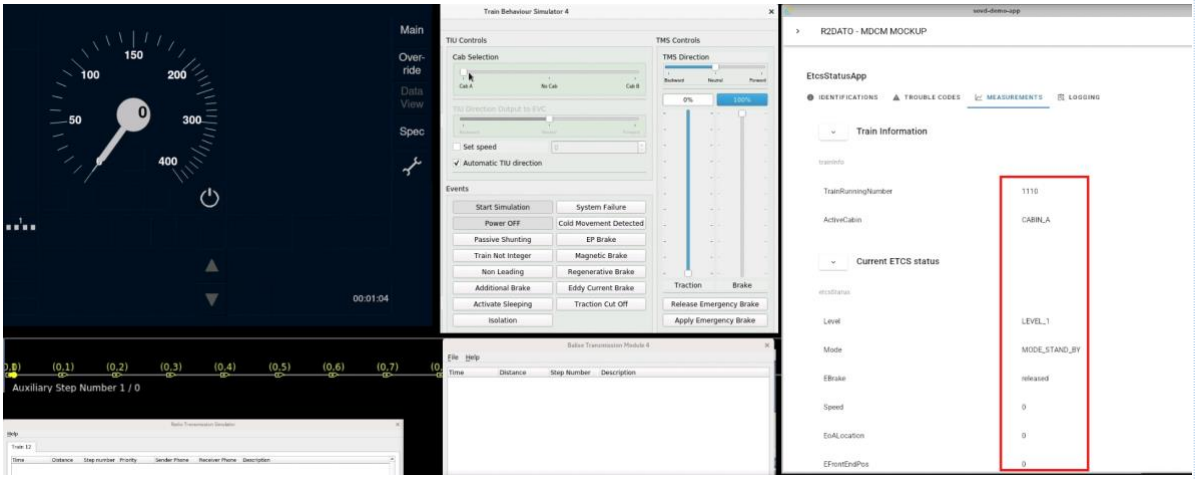
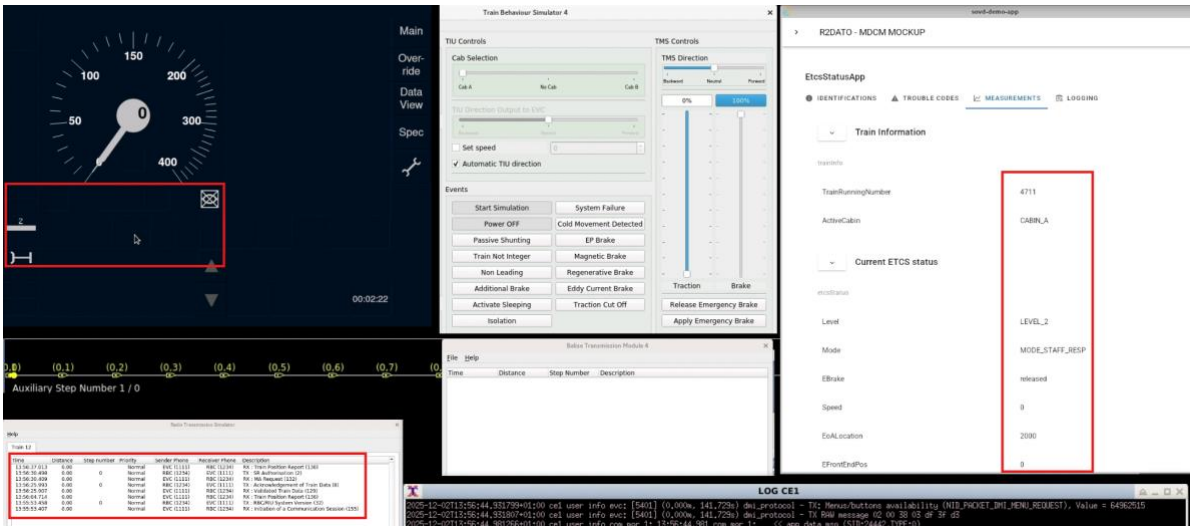
	c. *Start the testbench simulators using the scenario controller* 3. Open the train driver cab 4. Start mission on DMI 5. Drive train 6. *Stop train* 7. *Close cabin*
Test Data	None.
Expected Result	During the full test, the system must continue to perform its intended functions, without any detectable issues or degradation and provide diagnostics information to the remote diagnostics client.

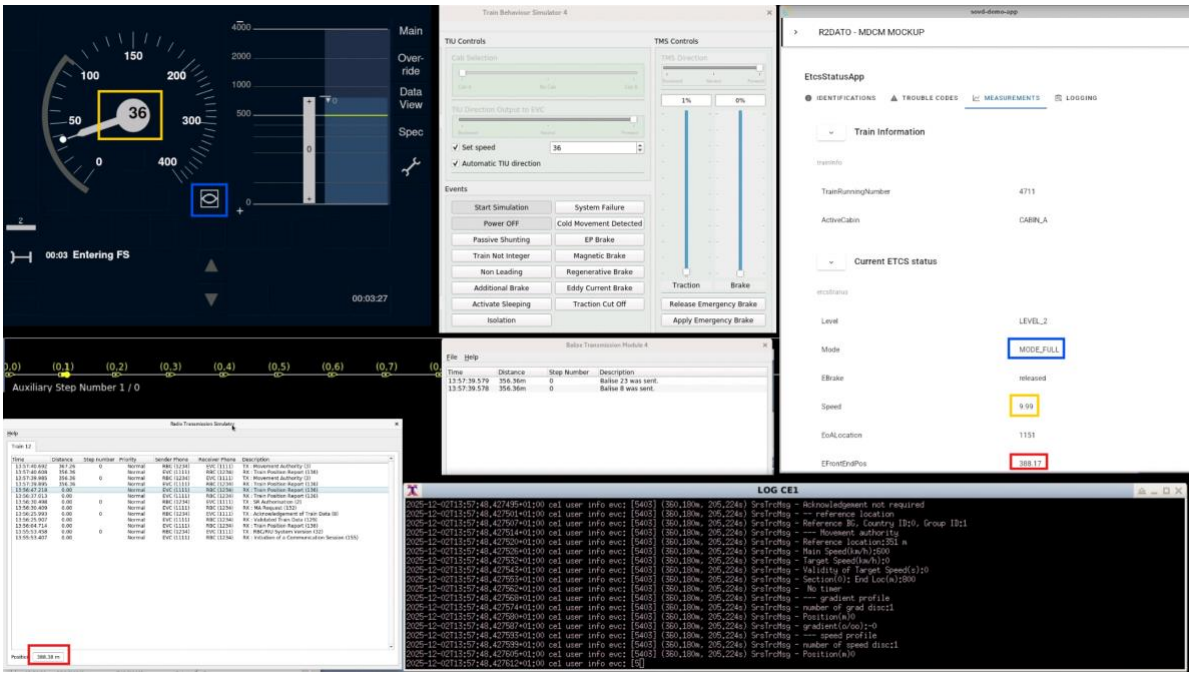
Actual Result

1-2 Refer to Test Case ID 2.10.2-A

For this test case, the output was reduced to the relevant components.



3	While the Cabin A is opened through the Train Behaviour Simulator, the DMI is activated and the status information is updated on the diagnostics client to show the active cabin (CABIN_A) and the ETCS mode (MODE_STAND_BY). 
4	During the Start of Mission process, the diagnostics information is continuously updated, transferred und visualized on the diagnostics client. e.g. Train Running Number, ETCS Level, ETCS Mode, EB status, Speed, 

5	While driving the train, the diagnostics information is updated on the client. e.g. current position (distance driven since start), ETCS Mode (MODE_FULL), current speed (36km/h ~ 10m/s) 
6-7	Refer to Test Case ID 2.10.2-A

Status	Pass
Comments	The sampling rate of the diagnostics information during this test case was set to one second. Thus, speed and distance information shown on the simulator and the diagnostics client showed (expected) discrepancies.
Attachments	

3.51 CACHE MAP DATA FROM DIGITAL REGISTER [2.10.3]

Description	As R2DATO WP36, we want to host an onboard map cache for the digital register on the safety layer / runtime environment, so that we can demonstrate a safe application communicating to a safe trackside entity over FRMCS and using safe persistent storage.
Annotation	Omitted: This User Story is not demonstrated because there is no digital register map available.

4 CONCLUSION

This deliverable documents the completion of the third implementation phase of WP36's on-board modular computing platform demonstrator. It describes implemented user stories in a laboratory environment, porting and running an ERTMS/ETCS Application on the platform in a fault-tolerant configuration, and consolidates test cases, results and lessons learned for use in follow-on work (notably D36.5 and further TRL advancement).

Main achievements and contribution to new knowledge

- **Porting and fault-tolerant operation of an onboard ERTMS/ETCS Application**

  Clearsy's ETCS Baseline 3 (System Version 2.1) was ported and adapted to operate in a 2-out-of-3 (2oo3) configuration on the modular platform, using the platform's native fault detection and recovery mechanisms. This is concrete demonstrator shows that applications of third-party suppliers can be integrated onto the modular platform and run in a safety architecture with redundancy primitives provided by the platform.

- **Demonstrated functional clusters and implemented user stories**

  The demonstrator executed and tested numerous user stories across Diagnostics, Deployment & Orchestration, Train Integration, Communication and FRMCS clusters. Implementations included the MCP-DIA diagnostics blueprint, TCMS Data Service, diagnostics data collection and aggregation flows, and several deployment/orchestration behaviours (e.g., handling failing replicas). The deliverable consolidates which user stories were implemented in WP36.4, which were demonstrated earlier in WP36.2/3, and which were skipped or left theoretical.

- **Integration of simulation and test toolchains**

  The project ran a comprehensive online toolchain (Route Map Controller, Train Behaviour Simulator, Balise & Radio simulators and a hardware DMI) to exercise the ERTMS/ETCS Application and test end-to-end interactions in realistic scenarios. This added rigor to the testing and provided reproducible scenarios for validation of networking and application behaviours.

- **Consolidated analysis of software update, IT/OT security and network aspects**

  The deliverable documents both practical tests and theoretical analyses (where lab constraints applied) covering software update workflows, IAM and encryption considerations, separation of zones, and FRMCS integration modes. These analyses provide actionable guidance for future prototyping, standardization and operator integration.

Evaluation versus objectives

- **Fulfilled objectives**

  The demonstrator successfully ported and ran an ERTMS/ETCS Application in a redundant configuration, implemented and tested numerous diagnostics and train-integration user stories and produced the respective test results, identifying the achieved outcomes and justified omissions.

- **Partially fulfilled or theoretical objectives**

  Mostly security, software update and migration user stories were treated theoretically, skipped or only partially implemented due to prioritisation, feasibility, or lack of resources and added value. Constraints in lab hardware, proprietary RTEs and the scope of the demonstrator played an additional role (e.g., full over-the-air FRMCS updates, full platform/RTE portability, and diverse hardware swaps were not fully demonstrated). These limitations were identified and documented in the deliverable, and it had been evident intentionally (please refer to D36.1 [2]), that not all user stories must be realised in the end.

Lessons learned

- **Hardware diversity and proprietary RTE limitations**

  The demonstrator relied on a single proprietary RTE and hardware module type. As a result, true hardware diversity, cross RTE portability, or migration between different vendor RTEs could not be demonstrated. Achieving this would require access to additional hardware variants and collaboration with multiple RTE suppliers.

- **Software update feasibility with TAS Platform**

  For all user stories possible implementations and demonstration paths were identified and have been described with respect to the functionality available in the demonstrator. Furthermore, using the TAS Platform add-on MNT, these user stories would be fulfillable.

- **Security architecture dependencies on operator infrastructure**

  IAM, key lifecycle management and monitoring must integrate with the operators' broader security environment. While architectural recommendations (e.g., TLS, Subset-146 based approaches) are clear, full end-to-end operational validation was not possible in the isolated lab environment.

- **Gap between realistic testing and certification requirements**

  The demonstrator provides a high-fidelity lab environment using simulators, hardware DMI and replicated ETCS behaviour. However, transitioning to field trials and certification introduces additional requirements (traceability, evidence for authorities, long-term operational robustness and resilience) that extend beyond laboratory capabilities.

Recommendations and proposals for follow-on activities

- **Conduct field trials to extend TRL**

  Collaborate with railway operators to move from laboratory testing to controlled field trials, progressing toward industrial readiness and higher TRLs.

- **Launch Hardware and RTE diversity pilots**

  Partner with additional hardware and RTE vendors to validate portability and migration workflows in pilots and prove application interoperability across heterogeneous hardware.

- **Pilot complete secure software update workflows for SIL applications**

  Implement and demonstrate full secure over the air update processes (including rollback, staging, IAM and key lifecycle management) in a pilot with operator involvement and clear certification preparation.

- **Specify and standardise application-platform safety interfaces**

  Develop concrete interface and responsibility specifications (e.g., message sequencing, replay protection APIs) that define clear responsibilities between SIL-4 application vendors and platform integrators. Feed results into Europe's Rail technical bodies, working groups and EuroSpec/ERA standardisation channels.

- **Integrate and harden security architecture with operator systems**

  Pilot and validate IAM integration, monitoring, and intrusion detection within actual operator environments to operationalise the theoretical security architecture.

- **Define certification pathways for modular SIL-4 hosting platforms**

  Establish a working group (operators, suppliers, certification bodies) to clarify evidence requirements and certification strategies for modular platforms hosting safety critical applications.

Why this matters — final synthesis

D36.4 provides tangible demonstrator evidence that modular on-board platforms can host and operate SIL-4 applications of third-party suppliers (such as ETCS) in fault-tolerant configurations while supporting diagnostics, TCMS data integration and modern deployment workflows. The work clarifies key architectural responsibilities — particularly that safety semantics must remain with the application — and delivers tested blueprints and lessons that materially advance the conversation about how to evolve on-board architectures for increased modularity, supplier diversity and lifecycle agility.

The work carried out has made efficient use of available project resources. In addition to the technical and architectural implementation, the team had to establish sophisticated laboratory and simulation environments, and the necessary testing and documentation added significant complementary effort. Considering this broader scope, the demonstrated proof of feasibility and the successful implementation of the defined user stories represent a strong achievement and confirm the value and effectiveness of the invested partner resources.

The remaining gaps are practical and well-scoped: hardware diversity, over-the-air certified update workflows, and operator-scale security and certification. Addressing these in follow-on pilots and through coordinated standardization will enable Europe's Rail to translate these demonstrator results into interoperable, certifiable solutions that support future GoA and digitalisation objectives.

This deliverable thus represents a meaningful step in the Europe's Rail System Approach: it consolidates prototype evidence, offers concrete architectural guidance, and identifies a focused set of next steps to push modular, SIL-capable on-board modular platforms from lab prototypes toward operational reality.

REFERENCES

[1] ASAM SOVD, “Service-Oriented Vehicle Diagnostics, API Specification,” 2022. [Online]. Available: https://www.asam.net/standards/detail/sovd.

[2] Europe's Rail R2DATO, “D36.1 – Demonstrator Specification, User Stories & Test Cases.,” 2023. [Online]. Available: https://rail-research.europa.eu/pages/fp2-r2dato/deliverables.

[3] Europe's Rail R2DATO, “D36.3 – Demonstrator implementation phase 2 concluded, and test results consolidated,” 2025. [Online]. Available: https://rail-research.europa.eu/pages/fp2-r2dato/deliverables.

[4] Europe's Rail R2DATO, “WP31 Validation of TCMS Data Service concept and formalization,” 2025. [Online]. Available: https://rail-research.europa.eu/pages/fp2-r2dato/deliverables.

[5] Europe's Rail R2DATO, “D36.2 – Demonstrator implementation phase 1 concluded, and test results consolidated,” 2024. [Online]. Available: https://rail-research.europa.eu/pages/fp2-r2dato/deliverables.