Rail to Digital Automated up to Autonomous Train Operation

D36.2 – Demonstrator implementation phase 1 concluded and test results consolidated

Due date of deliverable: 31/07/2024

Actual submission date: 08/01/2025

Leader/Responsible of this Deliverable: Hitachi / GTS

Reviewed: Y

Document status		
Revision	Date	Description
0.1	03/05/2024	Aligned Document Structure
0.2	19/06/2024	Technical content provision deadline
0.3	26/06/2024	Internal review
0.4	16/07/2024	Final review for TMT submission
0.5	19/07/2024	TMT submission
0.6	09/10/2024	Resolved minor TMT review comments
0.7	08/01/2025	Resolved JU review comments

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
Stefan Resch	Hitachi / GTS	Chapter 2.1, 3, 4, parts of Executive Summary, Introduction, Conclusion chapters, review
Wolfgang Wernhart	Hitachi / GTS	Chapter 2.1, 2.6, 4
Nikolaus König	Hitachi / GTS	Chapter 2.1, review
Reinhard Hametner	Hitachi / GTS	review
Benjamin Labonté	DB InfraGO	Chapter 3.2, 4.1, parts of Executive Summary, Introduction, Conclusion chapters, review
Oliver Mayer-Buschmann	DB InfraGO	Chapter 2.2, 2.3, 2.4, 2.7, parts of Executive Summary, Introduction, Conclusion chapters, review
Kai Schories	DB InfraGO	Chapter 2.3
Claus-Dieter Brei	DB InfraGO	Chapter 2.4
Maik Fox	DB InfraGO	Review
Zahoor Ahmed	DB InfraGO	Review
Thomas Buchmüller	SBB	Chapter 2.5, parts of Executive Summary, Introduction, Conclusion chapters, review
Thomas Martin	SBB	Review
Diallo Elhadj Thierno Sadou	Kontron	Review
Kevin Wriston	Kontron	Review
Harald Roelle	SMO	Review

EXECUTIVE SUMMARY

This document presents the first implementation report of WP36, encapsulating the outcomes from the initial phase of the Onboard Platform Demonstrator work package. This report progresses with a step-by-step approach, building on the specifications from the previous Demonstrator's Specification deliverable (D36.1), refining content, and presenting concrete results from the implementation and testing of proposed User Stories.

Key achievements include validating the feasibility to showcase deployments for mixed-SIL (Safety Integrity Level) functions based on the Modular Platform concepts of WP26. The implementation has proven many aspects of a complex architecture in a simplified demo application setup, emphasizing the role of automated testing in driving comprehensive test documentation from inception.

In a brief overview the report is presenting:

- An Architecture Update on those functional clusters where new insights have been gained
- A deeper introduction to the Implemented Modules and essential Modular Platform concepts and their concrete realisation in the demonstrator project on the Transportation Automation Systems (TAS) Platform provided by the task leader GTS/Hitachi as the focus topic of this implementation task
- Detailed reports on the Conducted Tests and Their Results based on a subset of realised User Stories (mainly utilising the demo application "Ping-Pong")
- A first Conclusion with a consolidation of the Key Learnings at this stage

The report delves into the architecture updates, providing insights on logical and physical components introduced in the Architecture document [7] of D36.1. It offers a deeper understanding of the Implemented Modules and essential Modular Platform concepts, realised in the project. The chapter Conducted Tests and Their Results also details key learnings gained by the realisation of a set of User Stories.

The Conclusion chapter consolidates these learnings and suggests further activities within Europe's Rail, ensuring the reader grasps the significance of the work and its alignment with the broader Europe's Rail System Approach.
In summary, Deliverable D36.2 marks a significant milestone in the Onboard Platform Demonstrator work package. It showcases the progress made in implementing proposed User Stories and conducted relevant tests, by setting a focus on software functionality, communication, and integration of a demo application. It lays the groundwork for future phases, setting the stage for more intricate implementations and tests, that will be continued in both, virtual and physical laboratory environments.

The report also highlights the importance of cybersecurity considerations and the need for continuous improvement and adaptation in project.

The findings and insights from this report will contribute to the further development and refinement of the Onboard Platform Demonstrator, paving the way for future advancements in rail automation, modularisation and digitalisation to enable further standardisation, evolvability and maintainability in the context of the FRMCS and ERTMS rollout in Europe.

ABBREVIATIONS AND ACRONYMS

Abbreviation	Definition
API	Application Programming Interface
BAV	Bundesamt für Verkehr (Switzerland)
CAN	Control Area Network
CE	Computing Element
CLIPS	C Language Integrated Production System
CN	Computing Node
COTS	Commercial Off-The-Shelf
DIA-VEC	Vehicle Diagnostics
EBA	Eisenbahn-Bundesamt (Germany)
FFFIS	Form Fit Function Interface Specification
FRMCS	Future Railway Mobile Communication System
FRS	Functional Requirements Specification
HTTP	Hypertext Transfer Protocol
I/O	Input / Output
IP	Internet Protocol
MCP-DIA	Modular Computing Platform Diagnostics
QEMU	Quick Emulator
RaSTA	Rail Safe Transport Application
REST	Representational State Transfer
RTE	Runtime Environment
SAHARA	Security-Aware Hazard And Risk Analysis
SRS	System Requirements Specification
SW	Software
TAS Platform	Transportation Automation Systems Platform
TCMS	Train Control Monitoring / Management System
TCMS_DS	TCMS Data Service
TLS	Transport Layer Security
TRL	Technology Readiness Level
YANG	Yet Another Next Generation (a modular language representing data structures in an XML tree format)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This deliverable (D36.2) is providing the first implementation report of WP36, summarising the outcome of the first of three implementation phases of the Onboard Platform Demonstrator work package. The objective of these published reports is to proceed with a step-by-step approach based on the Demonstrator Specification (D36.1), extending and refining relevant content continuously and presenting concrete results of the conducted implementation phase and proposed User Stories and corresponding testing to consolidate acquired findings and key learnings.

D36.2 summarises the outcome of the first implementation phase of the Onboard Platform Demonstrator work package. The main goal at this stage is to demonstrate the feasibility of a Modular Platform, supporting basic integrity and safety-critical applications as envisioned by WP26. This deliverable D36.2 provides a major milestone in showing that a concrete Modular Platform implementation indeed can realise those concepts, even if the developed and hosted demo application is rather simple in its first iteration.

The report consists of several key components. Firstly, an Architecture Update is provided, focusing on functional clusters where new insights have been gained. This update aims to increase the level of specification detail to provide a deeper understanding of the platform's structure and components.

The section Implemented Modules provides an overview of the first realised remits within the Onboard Platform Demonstrator. It covers the applied architecture, realised virtual test environment, platform, and demo applications.

The architecture section explains the overall structure and design principles of the implemented modules, setting the foundation for the subsequent discussions.

The virtual test environment section highlights the deployment of containers and the use of continuous integration, ensuring efficient, automated and reproducible testing by the definition and application of processes.

The platform section focuses on the TAS Platform, its programming model, and cybersecurity considerations. It showcases the platform's features and redundancy architectures, demonstrating its suitability for mixed-SIL function deployments (i.e. mixed-criticality).

The demo applications section showcases the current state of the demonstrator based on the TAS Platform. It highlights the specific developed demo applications and their integration into the platform, demonstrating their functionality and role in realizing the User Stories.

Detailed reports on the This demonstrator shows the feasibility of safe and non-safe input, output, as well as the integration of safe computation and basic integrity computation on top of the modular platform TAS Platform. Conducted Tests and Their Results set the primary focus on the demo application "Ping-Pong.", validating the functionality and performance of the implemented modules to provide valuable insights into the system's capabilities.

The Conclusion section consolidates the key learnings from the implementation phase. It highlights important achievements and acknowledges the challenges encountered during the process. The conclusion also emphasizes the importance of stakeholder engagement, knowledge sharing, and adaptability in project planning and execution.

Throughout the report, references are provided to relevant documents and specifications that support the implementation and findings. These references serve as evidence and provide further possibilities to follow up to interested readers.

2 ARCHITECTURE UPDATE

The following chapters aim to provide further content on logical and physical components that have been introduced in the Architecture document (D36.1 Architecture [7]). The intention of this update is to continuously develop the demonstrator specification and to increase the level of detail on specific functional clusters, where additional knowledge and insights have been gained and worked out during the implementation phase. For the sake of recap we represent here the Figure 1 of the chapter 2 “STARTING POINT FROM THE SYSTEM DEFINITION” of [7], though it might be a good idea to restudy that chapter to better comprehend the following content.

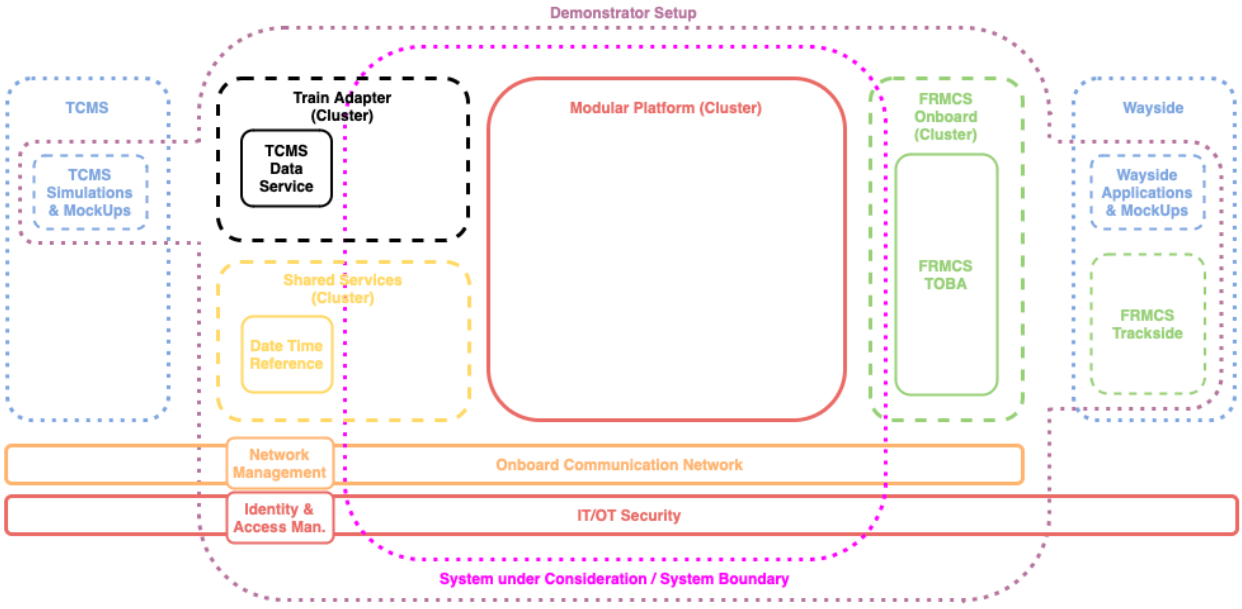


Figure 1: External Actors and Functional Cluster

2.1 FUNCTIONAL CLUSTER MODULAR PLATFORM

The R2DATO deliverables of WP26 encompass the comprehensive concepts and specifications of the overall modular platform. For the WP36 demonstrator, the TAS Platform serves as the foundation of the modular platform. In this section an overview of the TAS Platform is provided, followed by its programming model and cyber security approach.

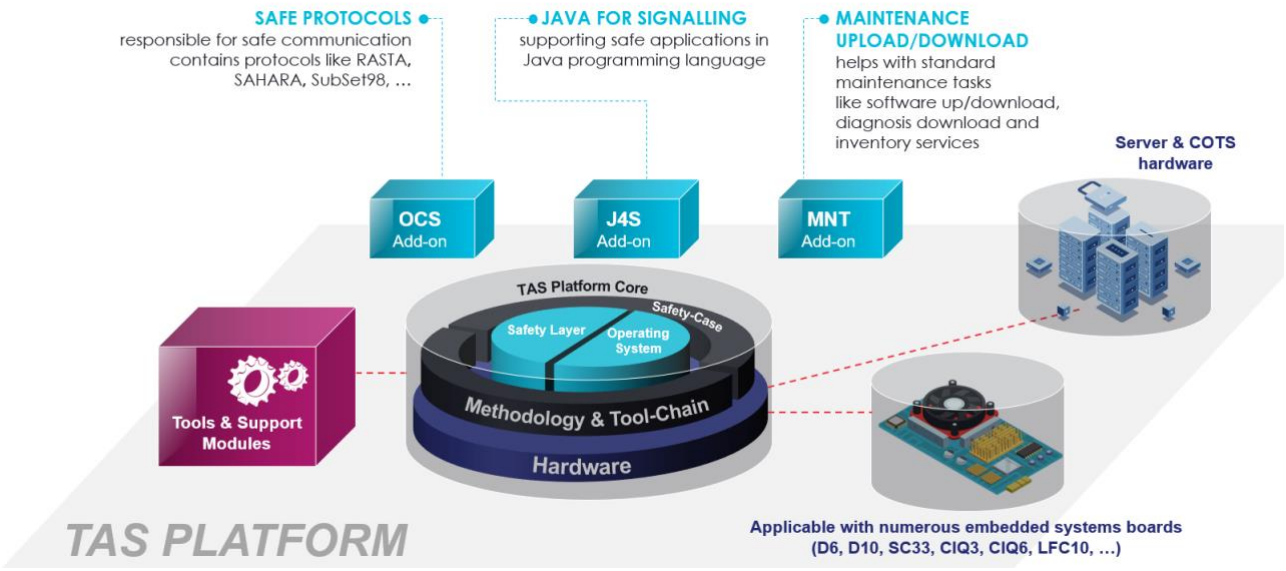
The architecture of the initial demo application, which is built upon the modular platform, is provided later in the document in Section 3.3

2.1.1 TAS Platform Overview

The TAS Platform is based on state-of-the-art software technologies. It uses an open, scalable software architecture based on well-established industrial computing standards and supports real-time multi-tasking applications in a computing environment. Depending on the applications’ needs, the platform runs on commercial-off-the-shelf (COTS) components supplemented by elements designed specifically for rail-way control systems, COTS hardware boards suitable for use in the railway environment or COTS server hardware. The core of TAS Platform contains a set of software components including the operating system, the communication system and the fault tolerance system. The TAS Platform’s communication system provides various standard services with extended semantics for safety applications, as well as safety relevant protocols, consistent with

CENELEC standards. The fault tolerance system offers several different redundancy configurations as well as fault management services. The platform meets stringent dependability requirements and provides application transparent redundancy handling and fault management services that enable safety-critical, real-time applications up to EN 50129 SIL4.

The TAS Platform has been assessed according to safety and security standards by independent assessors, end products are supervised by local national safety authorities. Its modular architecture, standard application programming interface and well-defined adaptation layers ensure that the platform will keep pace with technological advances in hardware components and system software components in a controlled manner. It has been used in the field successfully for over 20 years with safety responsibility.



Source: Hitachi / GTS

Figure 2: TAS Platform Overview

Figure 2 provides an overview of TAS Platform, consisting of TAS Platform Core software and hardware. It also includes support tools and modules as well as the add-on components OCS (One Channel Safe), J4S (Java for Signalling) and Maintenance (MNT). In detail, the TAS Platform Core provides:

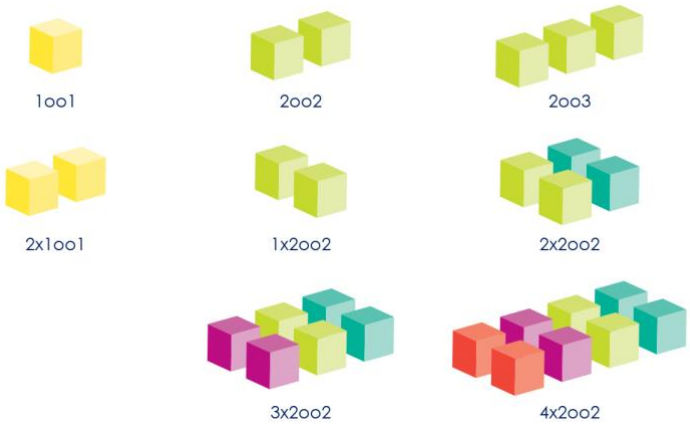
- Software:
    - Operating System: Open-source OS (e.g., Linux). The employment of such an OS is necessary to fulfil the needs of the safety critical applications and obtain major benefits such as:
        - introducing mechanisms for safety, reliability, availability requirements,
        - tailoring Linux to the needs for embedded safety-critical applications, one tailoring sufficient for all applications that run on TAS Platform,
        - targeting the real-time constraints of the applications,
        - providing drivers for (Hitachi Rail) developed devices (e.g., CAN driver),
        - providing additional packages used for customers,
        - dealing with the board specific devices for safety reasons,

        - managing operative/maintenance mode handling as well as safety reactions.
    - Safety Layer: provides the specific safety and availability functionality for safe applications. Here several redundancy architectures, such as 2oo3, 2x2oo2 or 4x2oo2 are supported as illustrated in Figure 3.
- Hardware: A set of COTS hardware boards, as well as the option to run on COTS servers.
- A corresponding safety case is developed for all supported redundancy architectures.
- Development Tool Chain: The TAS Platform provides validated and assessed compilers (C, C++). Online and offline debug and tracing support is given.

Concerning Tools & Support Modules, TAS Platform provides a separate tooling package for application development, which can help analysis during the development of applications in addition to development tooling like compiler. This tool chain is called POST (for Platform Offline Support and Tools). This provides debugger and associated terminal as well as tracing support. The capabilities to provide field tracing data (with limited tooling support) are available. POST is also a mechanism for the TAS Platform support to provide additional elements (like tools) to customers ensuring tracing and optimal support of customers.

The add-on OCS (One Channel Safe) provides safe communication between two safe TAS Platform applications and between a safe TAS Platform application and external non-Platform based components, according to CENELEC EN 50159 (class 1 and class 2, for class 3 additional measures must be considered). Various communication protocols and physical communication media are supported and in use. Methods and protocols such as SAHARA, RaSTA and others are supported in this communication framework.

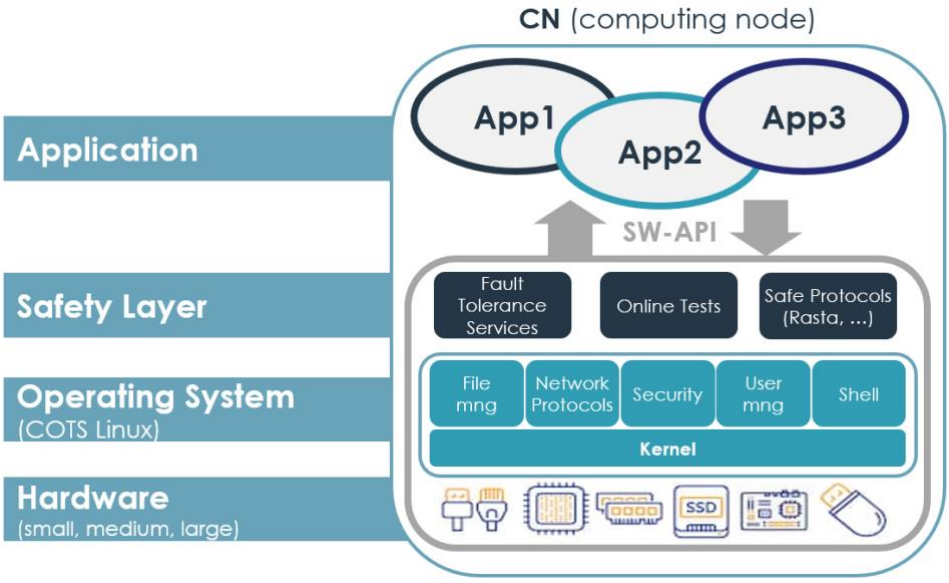
The add-on MNT is the Maintenance and Update framework of TAS Platform. The major focus of MNT is to enable secure up/download to the target system (embedded systems configuration and industrial server solution). Support of SNMP access techniques using a Management Information Base (MIB) for diagnostics in addition to secure download and upload service are provided. Download/update of the OS is also important and a major step for future maintenance activities, which is solved with MNT. Future applications will urgently need such technical support amongst others due to security updates. MNT can be used locally or in a remote environment.



Source: Hitachi / GTS

Figure 3: TAS Platform Redundancy Architectures

Figure 3 illustrates the redundancy architectures provided by TAS Platform. TAS Platform implements composite fail-safety between different hardware boards or virtual machines. The individual hardware boards or virtual machines are named Computing Element (CE), which together, e.g. as 2oo3, form a Computing Node (CN). The Computing Node is then the entity which implements and provides the safe function.



Source: Hitachi / GTS

Figure 4: TAS Platform Layered Architecture

The TAS Platform has a layered architecture as depicted in Figure 4. The top layer represents the functional aspects of the system, the applications. The middleware section (black boxes) provides an implementation of the API and a bridge between the programming models included in the safety layer, and the operating system on top of the hardware layer.

Application Layer: The application layer is split into two, very distinguishable parts: the safe applications and the non-safe applications part. This strict separation between non-SIL and safety related applications allows executing applications with different SIL levels on the same HW (mixed criticality).

Safe applications: The most important added value of the TAS Platform is the safe application RTE. This safe environment allows executing applications up to SIL4 according to EN 50129. The major property is that safe applications must reside on the Safe Platform API. All safety-relevant interfaces to lower-level SW components are developed as SIL4 Platform services. The TAS Platform OS provides SIL4 APIs to several library functions like configuration database (CDB) or diagnostic services (SNMP). For execution, the safe application uses the API and communication provided and supervised by the Safety Layer.

Non-safe applications: The non-safe part of the system applications is executed as ordinary Linux programs. This allows for Linux services like diagnostics, non-safe protocols or security applications as provided by the Linux community. Additionally, services like MNT (maintenance/download services) are implemented according to their needs for safety integrity. Also, non-safe applications are executed in this way. Applications running in this partition execute on one CE (e.g., HW board, Server) without support of the safe voting and redundancy mechanisms of the Safety Layer of the

TAS Platform. They interface directly to the POSIX/Linux API and can utilize all available services of the underlying OS.

Safety Layer: The Safety Layer consists of services such as deterministic scheduling, voting and fault management, redundancy management, and hardware supervision. To fulfil the safety requirements requested by EN 50129 concerning HW faults in CPU or memory, the Safety Layer provides integrity checks for data between the single replicas at start-up and operational phase. The Safety Layer implements the functionality of the POSIX/Linux API in a SIL4 developed and supervised way. Due to the synchronization and voting mechanism, the behaviour of the Safety Layer itself is checked frequently to detect failures of the underlying SW and HW components. Hence, each potential failure mode of e.g., pre-existing Linux functionality, is detected, and the safe state can be guaranteed. For availability reasons, the safe applications are pushed to high real-time priorities of the Linux OS. This enforces decoupling of safe and non-safe applications which are executing on lower priority levels. Therefore, the safe applications claim the near real-time behaviour whereas non-safe applications utilize the unused free computation power with a best effort approach.

Operating System Layer: The TAS Platform delivers an own OS distribution based on open-source Linux. TAS Platform tailors the Linux kernel and services for the releases of TAS Platform individually, regarding functionality, size, CPU family, board specifics, etc. The key focus in the OS is security.

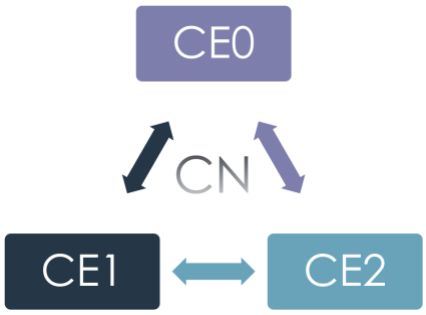
Hardware Layer: The TAS Platform Core provides a broad variety of HW boards fulfilling the environmental requirements of different systems: starting with the most stringent requirements of trackside environments (i.e., field elements), further going to on-board systems equipment used for CENELEC defined indoor systems (i.e., interlocking systems, Radio Block Centre, etc.). As a next step, the safe application computation can execute in environmental conditions that do not need to follow the stringent conditions of a CENELEC defined environment: safe computation on COTS industrial server boards, which are running in data centres without the physical requirements requested by CENELEC, assuming conditions of typical railway environments like relay rooms.

2.1.2 The TAS Platform Programming Model

The TAS Platform programming model divides each application into TaskSets that together provide the application's functionality. These TaskSets are further divided in Model 1 TaskSets used for safety-critical computation, and Model 3 TaskSets used for non-safety-critical functionality:

- Model 1 TaskSets for safe computation that are replicated, can only use a limited subset of POSIX and which outputs' can be voted
- Model 3 TaskSets which are not replicated for safety purpose, but that can perform I/O and/or communication to other POSIX threads. For availability several Model 3 TaskSets on different computing elements can be used for redundant I/O.

Communication to and from Model 1 TaskSets is only possible via message queues, provided by the TAS Platform.



Source: Hitachi / GTS

Figure 5: 2oo3 Redundancy Architecture

Considering the 2oo3 redundancy architecture depicted in Figure 5, for Model 1 TaskSets each CE has a respective instance running, while for Model 3 TaskSets only one CE executes such an instance.

Application execution consists of the three phases: start-up, operation, and shutdown, during which different parts of the API are permitted to be used:

- During start-up, Model 1 TaskSets must allocate all necessary resources and prepare the operational phase. Model 1 TaskSets declare to be finished by calling one of a defined set of functions which halts execution until all replicated instances have been started on all necessary CEs and all checks have been performed successfully.
- In the operation phase, the Model 1 TaskSets perform their safety-critical functions and use only a very limited set of the API functions. No resources must be allocated. I/O and external communication must be performed either via specific communication libraries, e.g., OCS, or in Model 3 TaskSets.
- During shutdown, application execution is prevented by the TAS Platform and only diagnostic tasks are allowed to run.

2.1.3 Cyber Security Considerations

The TAS Platform is assessed according to IEC 62443 4-2 SL 3 (security level) as enabling component. It has a managed security process and provides all necessary evidence and guidance's for use in an overall secure system according to IEC 62443 4-1 ML 3 (maturity level).

2.2 Functional Cluster FRMCS Onboard

Chapter 3.6.4.1 "FRMCS Gate" of D36.1 Architecture [7] introduces a concept for an FRMCS adaption on Modular Platforms for onboard systems. The main objective of this implementation is to realise a safe and secure end-2-end communication between applications hosted on the Modular Platform and their peers on the trackside. Instead of incorporating FRMCS-specific functionality directly into the applications, the approach is to centralize FRMCS-specific protocol details and communication implementation within a separate, reusable gateway component. This allows for greater flexibility and modularity within the system architecture.

In case the hosted application has a safety target (up to SIL4), it will run replica deterministic on top of the safety layer of the platform. The same applies to the FRMCS communication gateway, at least to those parts that communicate directly with the safe application or implement safety protocols. Redundancy can as well be beneficial to realise availability targets on the FRMCS specific part. In

favour to reduce the amount of “safe code”, the FRMCS functions and protocols shall be realised as a basic integrity implementation, reducing certification effort and cost, and enabling maintainability in the sense that the FRMCS implementation can be adapted later, without the need to modify any safety specific application code or the used safety protocols.

The following diagrams provide an overview and a deployment proposal for a potential later implementation in Task 36.3. The first diagram tries to comprehend and introduce all proposed components in a simplified view. The deployment diagrams show a more detailed sample deployment on three computing elements (CEs), where the dedicated software components build up communication paths between a replicated safe demo application towards an external entity, utilising voted queues in the safe part and non-voted queues and IP-based socket-connections in the non-safe FRMCS-specific implementation. The FRMCS Gateway is communicating bidirectional to the platform external TOBA-Box via the FRMCS OBapp protocol, that has been introduced in D36.1 Architecture [7], the specification is published at uic.org [10].

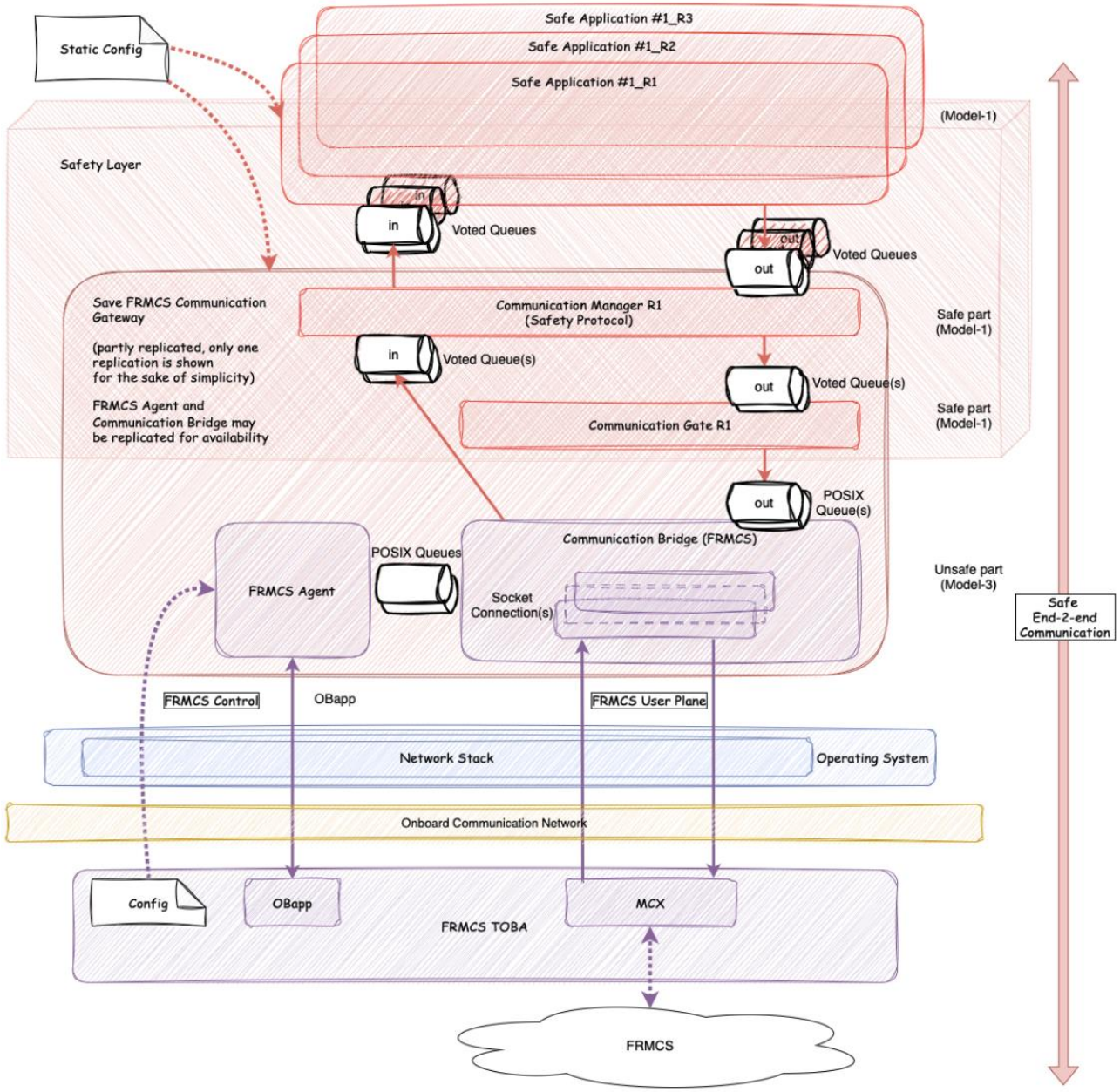


Figure 6: FRMCS Safe Communication Gateway Overview

2.2.1 Assumptions

- The replicated application(s), the FRMCS Communication Gateway software components along with the needed resources (application side communication endpoints, task-sets, voted and non-voted queues, etc.) will be instantiated based on a static configuration during the start-up phase of the Modular Platform.
- FRMCS-connection(s) to the corresponding endpoint(s) at trackside must be considered as dynamic and will be established and linked to the statically instantiated resources during runtime.
- Data Integrity and Security are supposed to be established on Transport and Application Layer by the utilisation of appropriate protocols. The provision of certificates and the utilisation of e.g., TLS is to be considered “state of the art” and thus might not be realised in the demonstrator implementation due to effort constraints.
- Any kind of safety measures (e.g., replication and voting) of the safety layer are transparent to the applications.
- The Communication Manager implements a specific Safety Protocol that provides an intrinsic end-to-end safety approach in line with the platform safety programming model.
- The FRMCS Agent receives its configuration during startup to setup network connections between the FRMCS Communication Bridge and TOBA using the OBapp protocol as described in the relevant FRMCS standards, please refer to [10], [11] and [12] for details. The control of those connections is handled by the FRMCS Agent, the network connections get established within the Communication Bridge and get linked to static allocated queues of the Communication Manager and Communication Gate.
- Due to the dynamic nature of TOBA network connections, it is possible to open, close and re-establish communication channels between the FRMCS Communication Bridge and TOBA during runtime.

2.2.2 Proposed Software Components

Software Component	Description
Communication Manger	The Communication Manger acts as the direct communication peer of the FRMCS gateway towards the applications hosted on the Modular Platform. It implements the adaption and distribution of voted send and receive data and provides and applies the safety protocol towards the external communication channels.
Communication Gate	The Communication Gate ensures that outgoing replicated data gets conjoint to one valid dataset and forwarded to the active Communication Bridge.
Communication Bridge	The Communication Bridge manages all active network connections and links the incoming and outgoing data towards the static queues of the Communication Manager and the Communication Gate
FRMCS Agent (Control Handler)	The FRMCS Agent receives a FRMCS specific configuration from the TOBA-Box and applies it towards the Communication

Software Component	Description
	Bridge. The FRMCS Agent implements the OBapp protocol [10] that controls all FRMCS connections between the TOBA-Box and the Safe Communication Gateway.

Table 1: Safe Communication Gateway Components

The following FRMCS Gateway deployment diagrams illustrate an example deployment of the former introduced software components on three computing elements of the Modular Platform. Since the Communication Manager and Communication Bridge are safe software components and communicate over voted queues, their implementation needs to be realised replica deterministic.

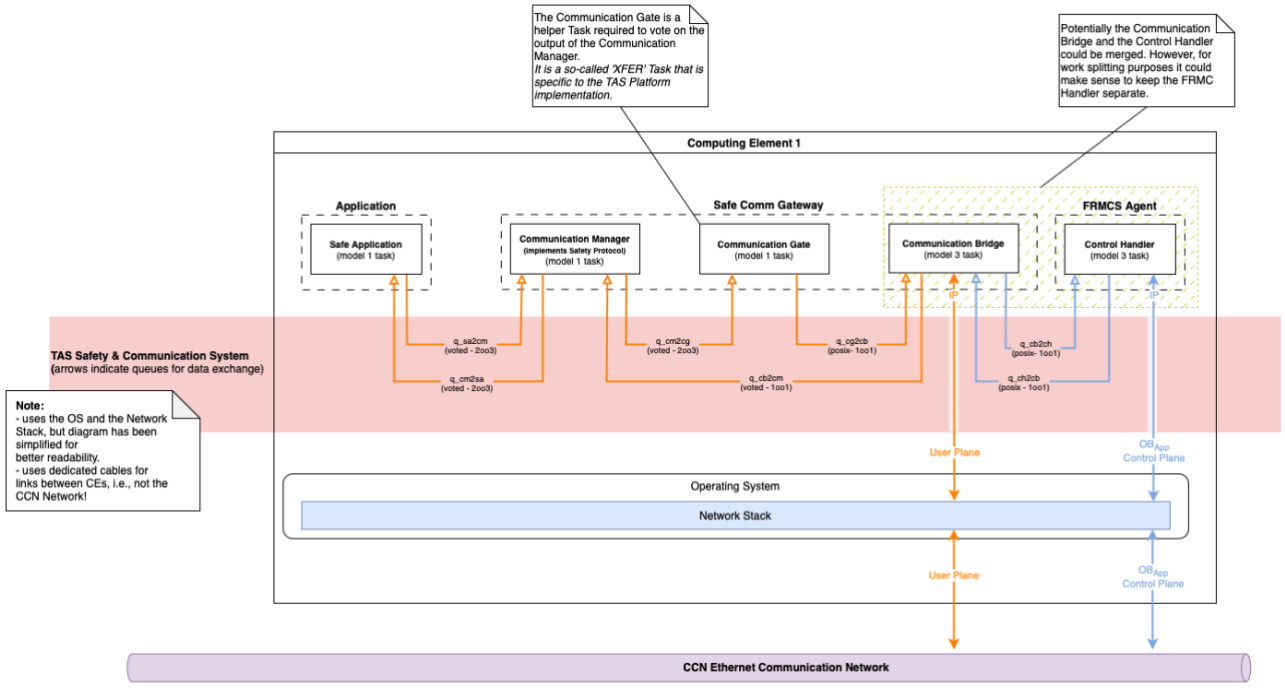


Figure 7: FRMCS Gateway example deployment CE-0

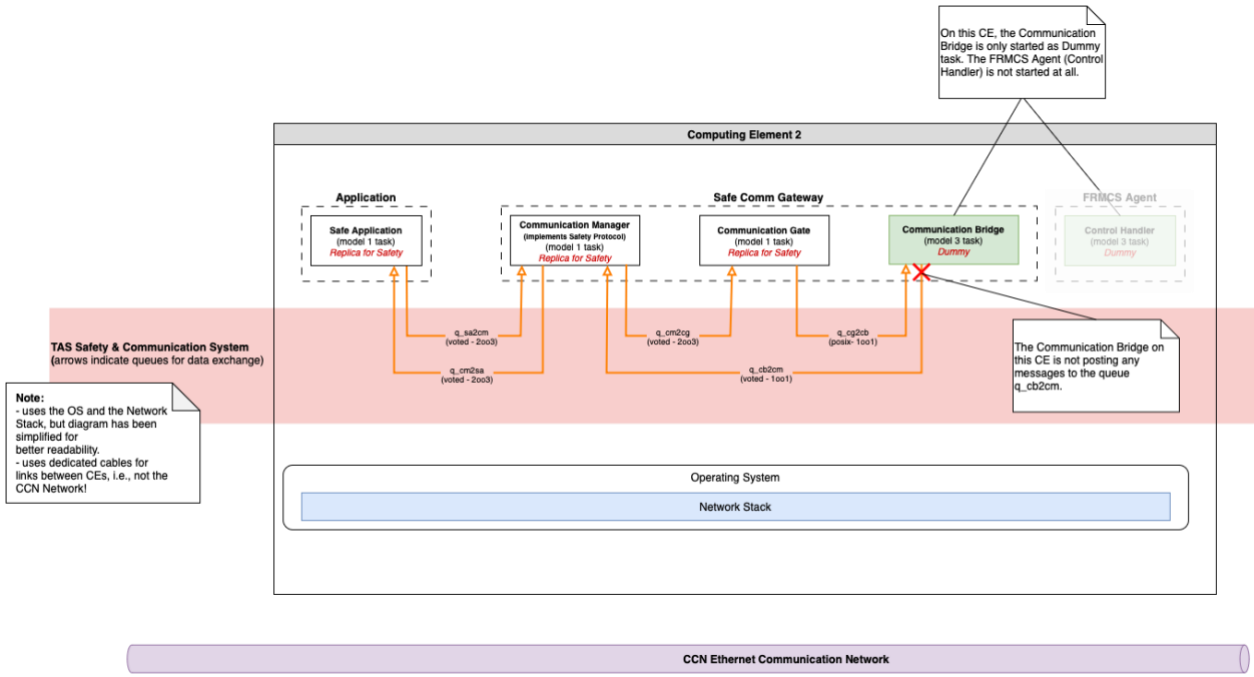


Figure 8: FRMCS Gateway example deployment CE-1

The FRMCS specific Communication Bridge and FRMCS Agent on the other hand can be realised in basic integrity. Additional instances as shown on the CE-2 deployment (on CE-1 they don't get functionally deployed) may run in a passive standby mode to increase availability and could be activated in case that CE-0 gets stopped/restarted e.g., in reaction to a voting mismatch, detected by the safety layer of the Modular Platform.

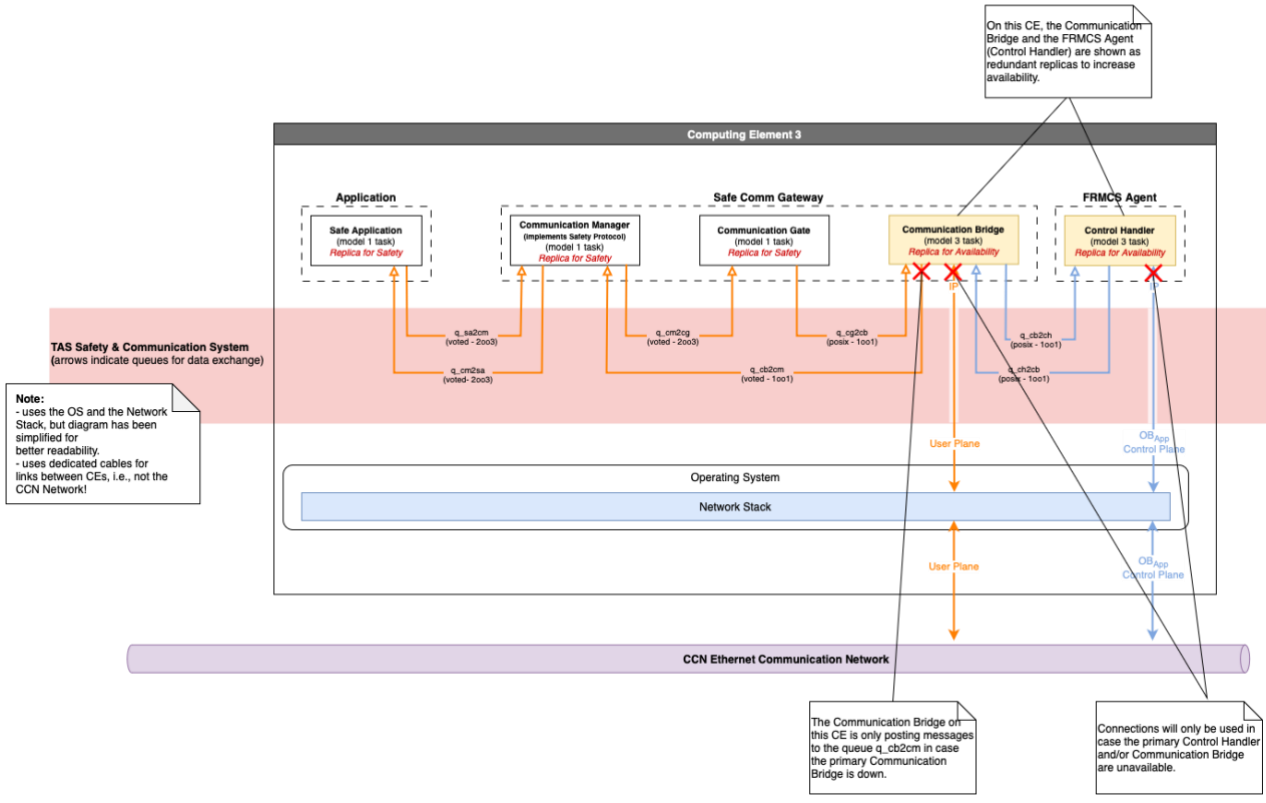


Figure 9: FRMCS Gateway example deployment CE-2

2.3 FUNCTIONAL CLUSTER DIAGNOSTICS

As introduced in chapter 3.7 of D36.1 Architecture [7], the primary objective of the diagnostics implementation is to demonstrate how health and performance data can be obtained on the Modular Platform and provided to and external Monitoring, Diagnostics, Configuration & Maintenance (MDCM) client through a common interface.

MDCM system requirements have been defined in OCORA MDCM SRS [9]. Chapter 3.3. of this document states the need for a “Common diagnostic communication interface to CCS-OB building blocks”, which targets the external MCP-DIA interface of the Modular Computing Platform.

This interface shall be vendor-neutral and scalable. Reuse and adaption of common standards and best practices from railway industry or other sectors (such as automotive) shall be considered when defining the Modular Computing Platform Diagnostics (MCP-DIA) concept and its implementation within the Onboard Platform Demonstrator. Detailed conceptual work, including the diagnostics service interface towards on the Modular Platform hosted applications, has been started in July 2024 and will be presented in future deliverables.

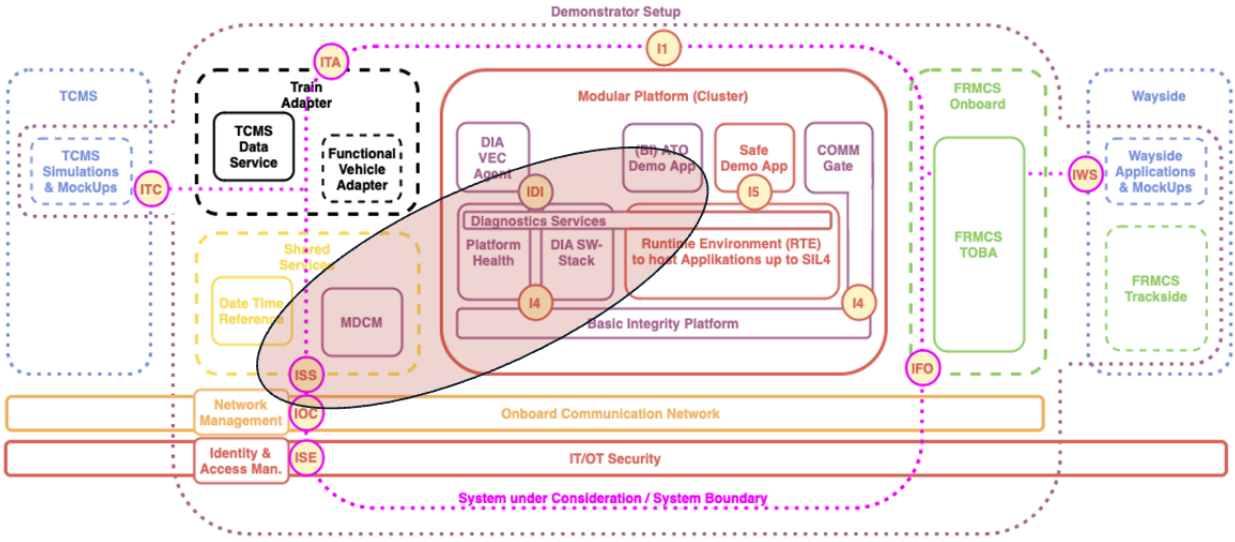


Figure 10: Focus area for MCP-DIA

The envisioned MCP-DIA scope in R2DATO WP36 demonstrator is depicted in the above figure. MCP-DIA represents the diagnostics services that get implemented on the Modular Platform.

The following diagram introduces a more detailed view on MCP-DIA, its peers and interfaces.

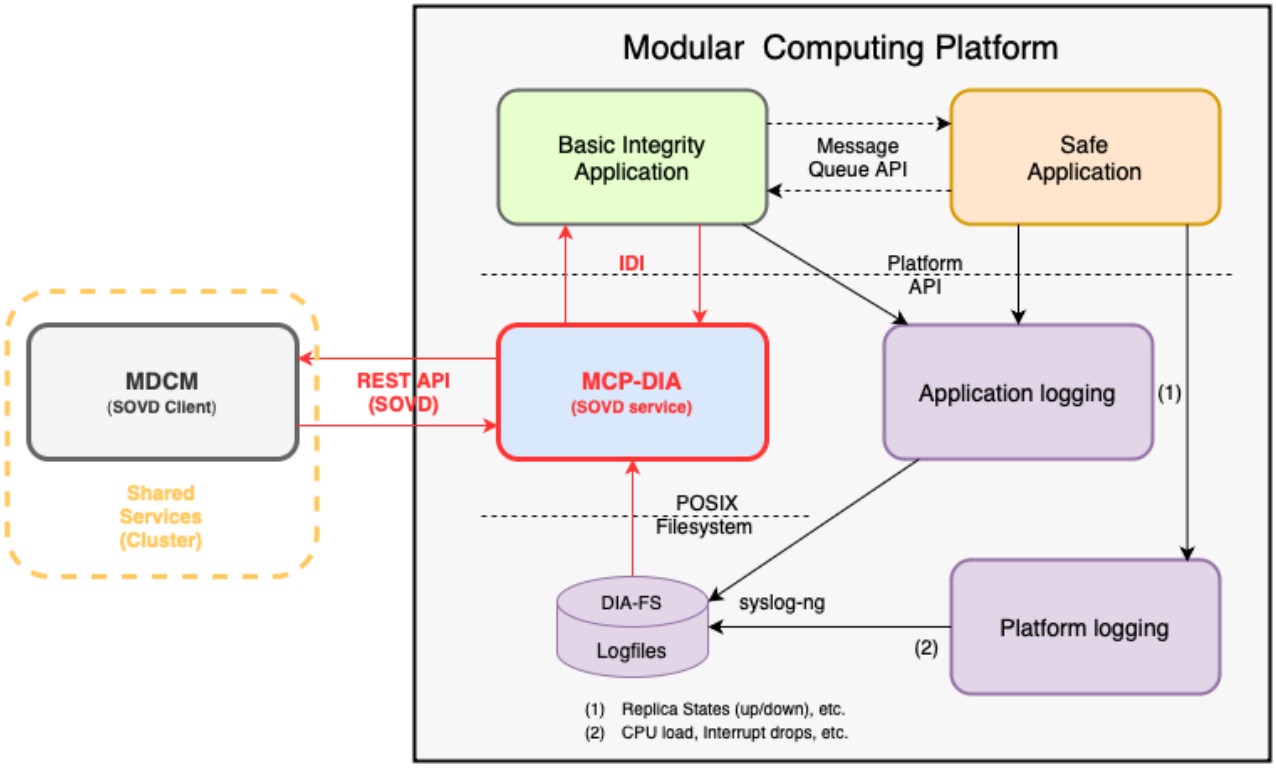


Figure 11: MCP-DIA Concept for R2DATO WP36 Demonstrator

For the realization of MCP-DIA the application of the recently published automotive diagnostic standard SOVD [1], [2] is pursued. SOVD has been primarily defined to address the diagnostic needs of new onboard architectures.

The SOVD standard defines a HTTP-REST-based approach to access diagnostic data, and to call procedures. For a particular SOVD instance, the REST API and associated services are configured using an OpenAPI-based scheme, called “capability description” [3].

The main functions of MCP-DIA are:

- collecting logging data from the hosted applications and health- and performance data from the computing platform (e.g., via logfiles, syslog-ng and dedicated diagnostics filesystems)
- providing diagnostics services to hosted (basic integrity) applications via an interface (IDI)
- providing collected data and enable the processing of diagnostics events through a REST API to an external MDCM client

Diagnostics clients, such as MDCM, can read health and performance data using SOVD API’s HTTP-GET method in combination with corresponding URIs describing the targeted entity (e.g. an application hosted on the platform) and type of diagnostic information, e.g. logfiles (bulk-data), faults, or parameters (data). The MDCM is currently planned as SOVD client application running as a VM/container on a laboratory server for shared services as introduced in the Network Architecture Figure of chapter 4.2.5 and chapter 7.4.1.4 of D36.1 Architecture [7].

Demo applications and platform services utilizing the diagnostics services are assumed to be provided latest in Task 36.4.

2.4 FUNCTIONAL CLUSTER TRAIN INTEGRATION

The intention of this chapter is to provide a first high-level design and integration proposal for the basic integrity applications “TCMS Data Service” and “DIA-VEC”. Both concepts have been introduced in D36.1 Architecture [7] chapter 3.7.3.1 chapter 3.8.2.2.

As proposed in the D36.1 Statement of Work [6], DB will integrate those two basic integrity diagnostics applications to the Modular Computing Platform next to the Safety Layer / Runtime Environment (RTE). The intention is to demonstrate how formalised data from the TCMS domain can be collected (e.g., from simulated TCMS brakes system), harmonised and further aggregated for a potential provision to future CCS onboard applications or systems. Those applications may utilise this health and performance diagnostics data, e.g., in future train operation scenarios as GoA4. The data may as well be transferred to the wayside for the purpose of conditional and predictive maintenance.

Hereby simulated TCMS train systems will generate diagnostic data from a data model, transmit it to the TCMS Data Service where it gets harmonised and provided via a standardized interface to DIA-VEC. The data flow from the TCMS domain to DIA-VEC has already been introduced in chapter 3.8.2.4 of D36.1 Architecture [7].

Related WP36 implementations will be aligned with the R2DATO Work Package 31, Task 2 “TCMS formalisation”. Since no GoA4 related applications will be realised in WP36, the collected data is mainly intended to be visualised in a kind of dashboard application which will not be integrated on the Modular Platform.

2.4.1 Proposed Software Components

For the demonstrator project four software components will be developed.

Software Component	Description
Simulation of a brakes system of a train (provides data as input for TCMS Data Service)	Simulation of a TCMS brakes system (CoSimBrakes) is a Matlab Simulink simulation, to (virtually) drive a train and (de-)activate specific failures. Hence, it outputs diagnostics data, referring to the brakes system of the train. CoSimBrakes will be connected to the TCMS Data Service.
TCMS Data Service (TCMS_DS)	TCMS_DS is implemented as a publish-subscribe service, receiving and transmitting data, which was previously defined in a YANG [14] model. The idea of the TCMS_DS is based on the EuroSpec document ‘Specification TCMS Data Service’ [13]. TCMS_DS shall be executed on the TAS platform. TCMS_DS receives its input data from CoSimBrakes and provides data to its subscribers. In this case the subscriber is DIA-VEC (Brakes).
The Basic Integrity application DIA-VEC (Brakes)	DIA-VEC (Brakes) is realised based on a Matlab Simulink model, which is specifically configured to handle mainly diagnostic information from a vehicle brakes system. DIA-VEC (Brakes) is connected to the TCMS_DS and aggregates data according to pre-defined expert rules. These rules are included in the configuration of DIA-VEC (Brakes), using CLIPS [15], a notation

Software Component	Description
	syntax developed by NASA. DIA-VEC (Brakes) shall be executed on the TAS platform.
DIA-VEC client	It is foreseen to realise a simplified client running on a separate hardware, providing a graphical interface to visualise results on a Node Red [16] based dashboard.

Table 2: Diagnostics Software Components

2.4.2 Proposed Deployment

The physical deployment is planned as depicted in the following diagram. TCMS_DS and DIA-VEC (Brakes) are executed on the Modular Platform. The visualisation of the collected and aggregated data will happen on a Maintenance Terminal (Laptop, DIA-VEC client), connected to the Onboard Communication network.

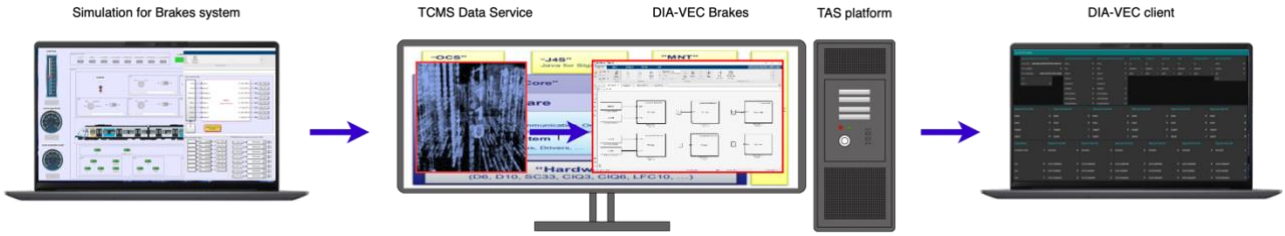


Figure 12: Diagnostics Applications physical Deployment

DIA-VEC will be based on a Matlab Simulink library. This library will provide three different types of agents and additionally a publish-subscribe service. The publish-subscribe data distribution acts for demonstration purpose only. Due to resource constraints, it is not foreseen that DIA-VEC (Brakes) will utilise a FRMCS Gateway as introduced in Chapter 2.2 to transfer data to the wayside. Nor it is decided yet to which extent DIA-VEC may utilise MCP-DIA services.

2.4.3 DIA-VEC Configuration

The used Matlab Simulink library provides generic modules (agents and publisher, that can be used for diverse systems (e.g., Doors, Brakes, etc). By applying a specific configuration, the Matlab Simulink library 'turns' from a generic library to a specific system. DIA-VEC implements several agents, that provide different functionality.

DIA-VEC Agent	Description
MON_Agent	receives data and extracts signals and their values for further aggregation
TEST_agent	tests signals for specific values (needed for further aggregation)
FUSION_ agent	executes expert rules on the signals. Expert rules are programmed in CLIPS [15].
FA_publisher	transmits the output to subscribed clients

Table 3: DIA-VEC Agents

The configuration specifies properties of those agents (number of instances, definition of input signals, test of specific conditions, etc.). Besides logical operations and definition of dependencies it also includes expert rules (refer to CLIPS, mentioned above) of the FUSION_agent. The FA_publisher transmits the output.

A configuration example is presented in the following diagram, depicting how a combination of single and generic components (MON-, TEST-, FUSION Agents, FA-Publisher) are combined in a specific configuration to create a DIA-VEC model for a specific system (like Brakes) out of a toolbox of generic agents.

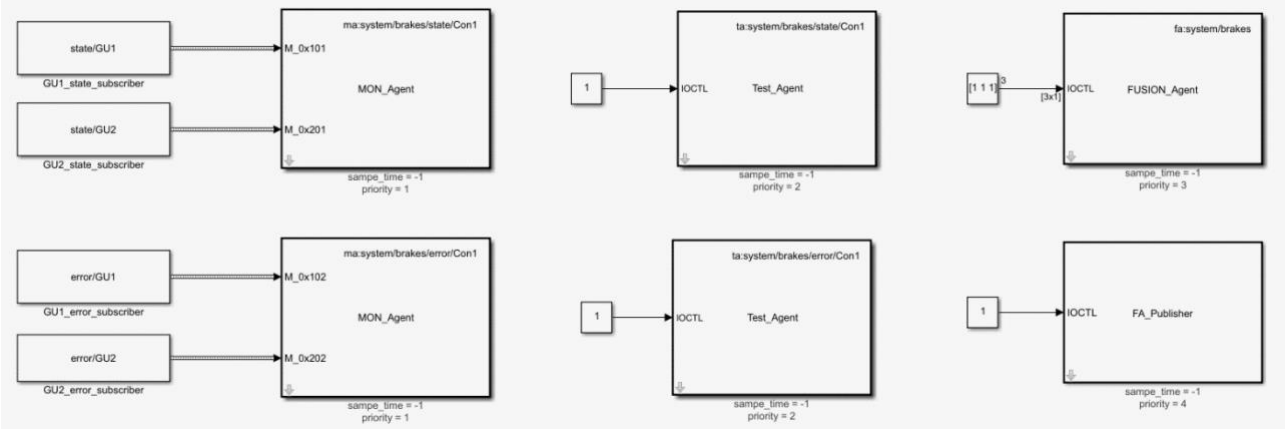


Figure 13: DIA-VEC publish-subscribe configuration

The process steps from creating a configuration until deploying DIA-VEC is shown in the following figure.

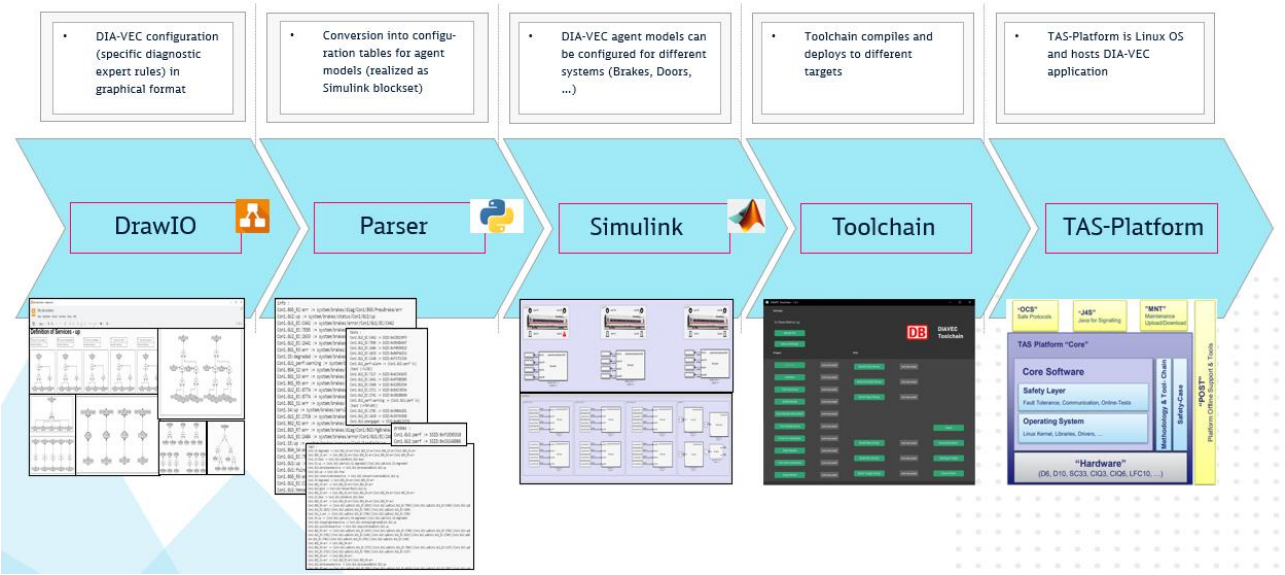


Figure 14: DIA-VEC Configuration and Deployment Process

2.5 FUNCTIONAL CLUSTER ONBOARD COMMUNICATION NETWORK

The Onboard Communication Network functional cluster provides connectivity between the onboard functional clusters Modular Platform, FRMCS TOBA, Shared Services and TCMS. The lower OSI layers 1 & 2 are specified in the current SUBSET-147 v1.0.0. The Onboard Communication Network

is based on a standard Ethernet network according to IEEE 802.3. It supports the separation of the physical network into different logical networks or zones based on VLANs according to IEEE 802.1Q. Quality-of-Service (QoS) on the data link layer is targeted on OSI layer 2 by using Priority Code Points (PCP) as of IEEE 802.1Q. With this measure, a defined priority can be assigned to each data class.

Note: SUBSET-147 will be enhanced with the OSI layers 3 to 6 in its next release. Therefore, R2DATO WP23 defines requirements for the extension within the deliverable D23.2 [4]. The proposal of the SUBSET-147 update with an initial specification on the layers 3 to 6 shall be available as part of the deliverable D23.4.

2.6 FUNCTIONAL CLUSTER IT/OT SECURITY

A risk assessment process is used to derive a security architecture for a railway system/product. There are certain processes to be followed, see also TS 50701 and upcoming standard PT IEC 63452 as basis.

After alignment with WP3 Task 3, the partners of WP36 concluded that a risk assessment for the demonstrator cannot be foreseen in this first implementation Task 36.2. Reasons for that are the lack of the final hosted functional applications and the currently realised deployment within a pure virtual environment. A major functional extension and transfer into real railway assets will happen in Task 36.4 with the integration into the final laboratory setup. Even then, a full risk assessment for the laboratory would not be able to cover the physical access strategy of a real product deployment. Any concrete details as existing security assessments and certifications of the TAS Platform are intellectual property of the platform supplier and cannot be disclosed here. Therefore, this chapter provides only some non-exhaustive guidelines, how a cybersecurity risk assessment can be performed:

- A cybersecurity risk assessment can be done on component level but must be considered on the end-product (system level) to have the full evidence of the cybersecurity risk considered.
- The product can trust on cybersecurity assessments of modules used, if available. This usually also means, that some instruction and documentation is available, which then serves as input, guideline and argumentation help to make the overall assessment more efficient.
- A final product (Functional System) can usually not achieve a higher security level than individual modules used. If this needs to be achieved, a re-consideration of the security architecture must be done by considering countermeasures based on the cybersecurity detailed risk assessment.
- If a module with an already available cybersecurity assessment is used, usually only the delta needs to be considered, dependent on the change and the overall security architecture. There must be a clear reason and a validation of the impact of the deviation from the given instruction. For rules which are not violated (taken as described), only the prove (in code, config, etc.) for the unchanged usage needs to be shown.
- A penetration test is an essential part of an overall cybersecurity assessment. The ultimate penetration test verifies if all requested security provisions are available with the requested SL (security level). Typically, these tests are executed by an independent company specialized on security vulnerabilities.

- Current available standards such as IEC 62443, TS 50701 and PT IEC 63452 must be considered.

Steps important during product development:

- Define the target security level (SL). Usually influenced by the customer.
- Identification of the system under consideration and its treat environment shall be defined.
- Initial cybersecurity risk assessment shall be performed to identify the worst-case unmitigated cybersecurity risks.
- Detailed cybersecurity risk assessment shall be performed to consider all risk perspectives and the results must be applied consistently to all considered zones and conduits. There may be an update needed when compensating countermeasures need to be evaluated or if new treats or vulnerabilities become known.
- Cybersecurity must be considered from the very beginning as secure by design. Learning instructions and capabilities of used modules is necessary and key.
- A cybersecurity organisation and a cybersecurity process shall be set up (e.g. according to IEC 62443-4-1).
- Cybersecurity requirements shall be considered such as IEC 62443, TS 50701, requirements based on the cybersecurity risk assessment.
- Cybersecurity assurance shall be considered, such as system validation, system acceptance, see IEC 62443, TS 50701 and PT IEC 63452.
- Operational maintenance and disposal shall be considered, such as vulnerability management, incident management, cybersecurity case updates, patch management.

2.7 PHYSICAL ARCHITECTURE

2.7.1 Laboratory Preview

To achieve the aimed TRL of 5/6 it is foreseen to realise multiple user stories that demand implementation and realisation on real hardware in a heterogenous laboratory environment. In this chapter we would like to provide an early preview on the planned laboratory setup, that is foreseen to be realised in the following two implementation tasks 36.3 and 36.4, depicting the ongoing partner alignment to ensure that the objectives of the overall endeavour will be achieved.

Most of the demonstrator assets will be integrated in a laboratory hosted by SBB. The main setup deployed there will be complemented by a remote integration of FRMCS trackside systems hosted in a Kontron laboratory. Both labs will be connected by VPN connections to ensure a secured laboratory integration. The following diagram provides a very simplified and preliminary view on the distributed laboratory setup.

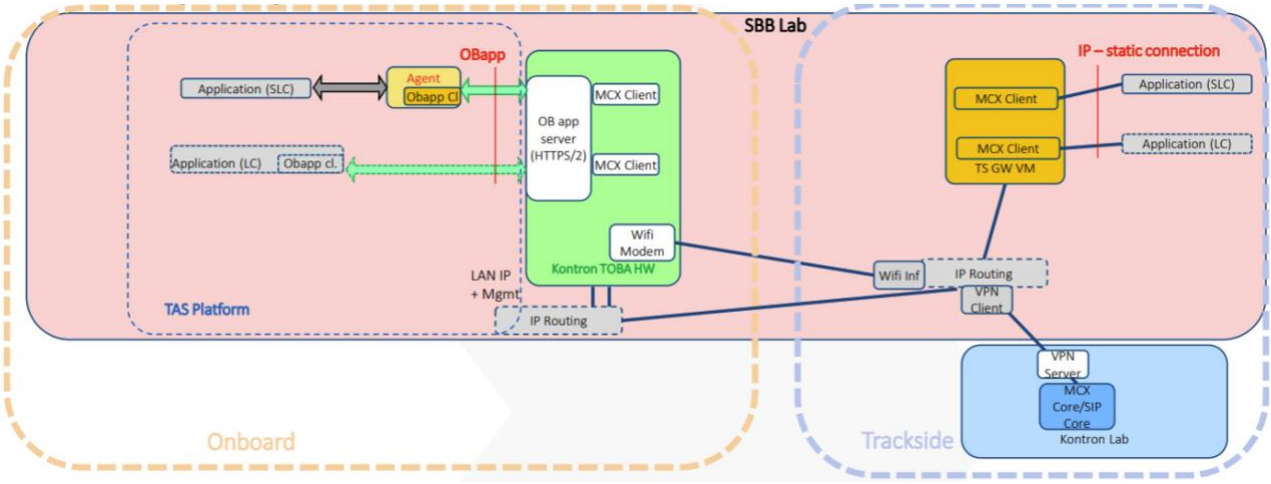


Figure 15: Laboratory Split

3 IMPLEMENTED MODULES

This chapter describes the overall demonstrator architecture focus that has been set in context of D36.2, followed by a section on the utilised virtual test environment. The first demonstrator application itself is then introduced in the last section of this chapter.

3.1 DEPLOYMENT

As proposed in the Statement of Work of D36.1 [6] this deliverable D36.2 sets its focus mainly on the software functionality, communication and integration of a demo application hosted on top of the Modular Platform, in particular the TAS platform, as introduced in chapter 2.1.

The main difference between the demonstrator architecture defined in D36.1 and the realisation within this first implementation task 36.2 is, that the TAS platform is not yet running on a physical hardware but on a virtual test environment. Furthermore, the communication to platform external counterparts is purely IP based, neither implementing any kind of safety or security protocols yet, nor utilising the foreseen FRMCS infrastructure.

It is essential, that all virtually deployed artefacts are as close as possible to those that would be integrated on real hardware. The virtual environment needs to be well suited to conduct automated testing, without any constraints to realise this first set of User Stories.

More details on the implemented User Stories and Test Cases can be found in Chapter 4.

3.2 VIRTUAL TEST ENVIRONMENT

There are several approaches on how the replication of safe demo applications can be realized in a virtual test environment. Within this task 36.2 four approaches have been evaluated:

1. **Hardware virtualization using QEMU**

   This approach provides the most realistic virtual environment to the deployed TAS platform runtime and allows to test almost all its features. However, the host system also must fulfil special requirements, like providing access to the Kernel-based Virtual Machine (KVM). This feature is usually only available on dedicated hardware while public cloud services usually disallow nested virtualization.

2. **Containerization using Docker**

   This setup as illustrated in Figure 16 brings the advantage that it only requires a virtual machine with a docker runtime, allowing fast and flexible deployment and functional testing of the basic platform user stories that are in the current scope. Therefore, this approach has been chosen to validate the user stories listed in the chapter Tested User Stories.

3. **Running all replicas in a single CE**

   For testing purposes all replicas could be started within the same virtual machine or container (taking the place of the CE in the virtual environment). Even though such a setup would allow testing the detection of a voting error, some safety features of the TAS Platform cannot be tested. Also, special deployment artefacts (configuration) would be needed that distinguish this setup from deployment to real hardware.

4. **Running without replication on a single CE**

   Functional testing and interfaces validation (e.g. for message queue communication as introduced in The TAS Platform Programming Model) usually doesn't require safety features as TaskSet replication and voting.

   Such kind of testing can be realised in a simplified setup with the drawback that hardware failures cannot be detected, and that separate configurations for the virtual und physical targets need to be maintained.

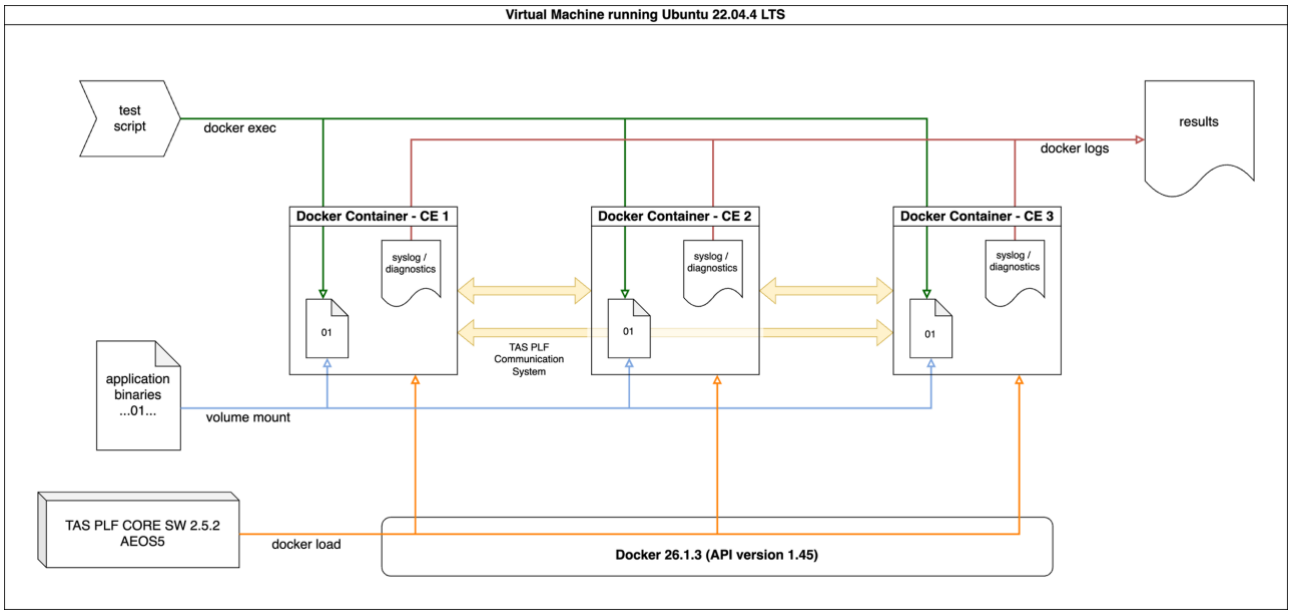


Figure 16: Setup of Virtual Test Environment

It is already established that many user stories in the scope of later tasks can only be validated using a physical test environment (e.g. such that require special FRMCS hardware of which there is no virtual equivalent available). Thus, after validating the basic platform user stories in this task, the virtual test environment will in future mainly serve as a supporting environment for automated testing before deployment to the physical targets as further elaborated in the chapter Continuous Integration / Continuous Deployment.

For this purpose, a fast, easy and cost-efficient deployment were the main goals for the setup of the virtual test environment of WP36. The containerization in Docker outperforms the other approaches in all those metrics.

3.2.1 Azure DevOps

For the implementation of the “Onboard Platform Demonstrator”, the partners agreed to utilize a state-of-the-are DevOps platform for agile organisation, source code / configuration version control and continuous integration. Being one of the leaders in the domain of DevOps platforms [17], Azure DevOps by Microsoft has been chosen.

The virtual test environment described above is realized by utilizing a Microsoft-hosted pipeline agent out of the Azure Pipelines agent pool using the “ubuntu-latest” virtual machine image.

3.3 DEMO APPLICATIONS

The focus of this work package is properly validating the platform technology by deploying representative demo applications on this platform. As committed in the D36.1 Statement of Work [6] meaningful demo application out of the railway domain will not be integrated prior to task 36.4. To be able to validate already in this implementation task first basic platform user stories, some artificial demo applications were developed that are minimal in complexity regarding the purpose they serve.

3.3.1 Initial Demonstrator Based on TAS Platform

In the context of WP36.2 the focus is set on creating an initial Demonstrator Application named Ping on top of the TAS Platform (Modular Platform) in a virtual execution and test environment.

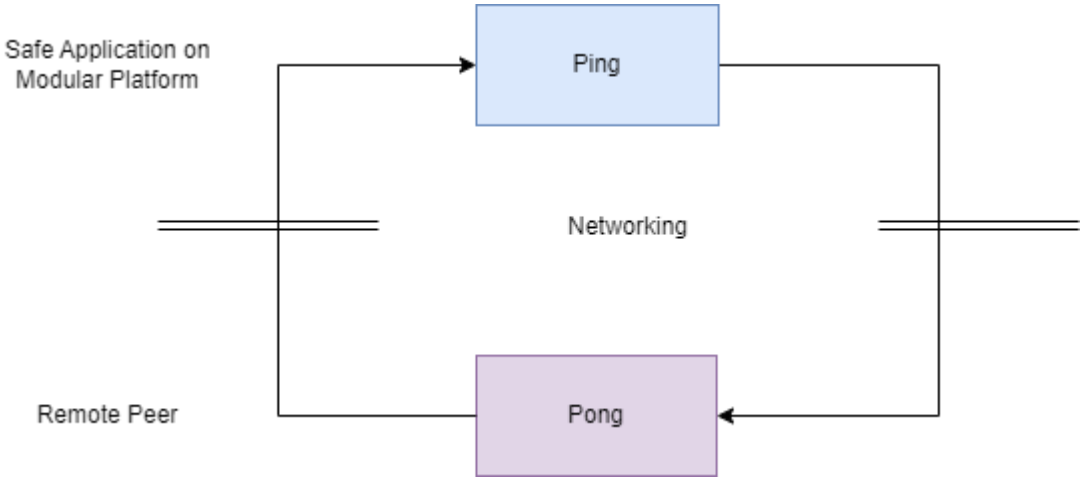


Figure 17: Demonstrator Application Context

The Ping application itself is running on top of the Modular Platform. It communicates via IP with a Pong application that can be running on a remote peer as depicted in Figure 17. Mapped to the overall demonstrator architecture introduced in the Figure of chapter 4.2.5 of D36.1 Architecture [7], the Ping application represents the application running on top of the modular platform, while Pong represents for example a trackside entity.

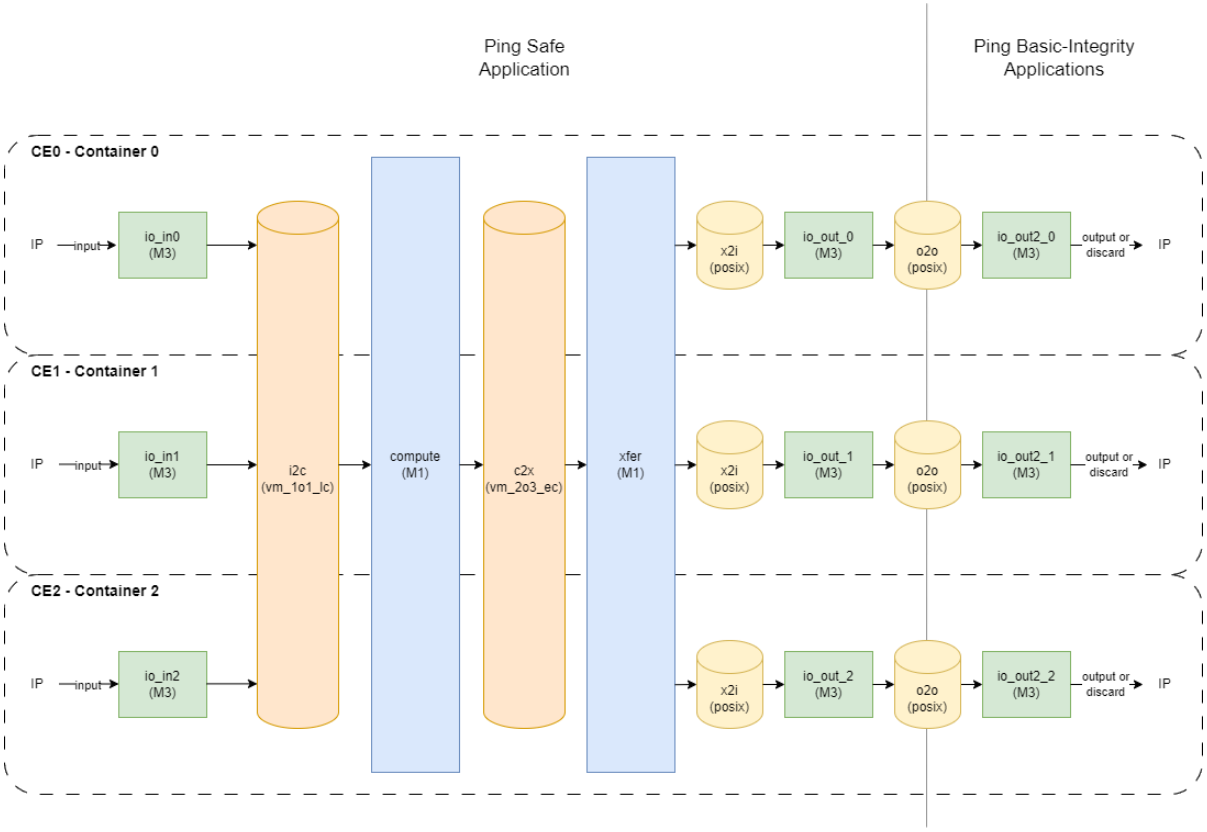


Figure 18: Demonstrator Application Ping

Figure 18 illustrates the Ping application deployed on top of TAS platform, where TaskSets are depicted as boxes and queues as cylinders. The demo implementation contains all elements needed to realise a safe application and a non-safe application that communicate via IP to a platform external peer, to cover a large part of the initial tests. The figure omits the Pong process on an external host since the input and output is part of the tests themselves.

The Ping safe application follows the safe programming model and utilizes the safety mechanisms of the TAS Platform. In the safe application the `io_in` TaskSets are used for input from IP, `compute` for safe computing the result, `xfer` for forwarding the voted result and the `io_out` TaskSets for output to the non-safe applications. The non-safe application `io_out2` then forwards the output via IP. Note that the Ping safe application itself also contains Model 3 TaskSets which must not perform safe computation.

The detailed safe Ping application consists of:

- Model 3 TaskSet `io_in_0`: Receives UDP messages from a socket on CE0 and sends the messages to the compute TaskSet via the message queue `i2c`.
- Model 3 TaskSet `io_in_1`: Receives UDP messages from a socket on CE1 and sends the messages to the compute TaskSet via the message queue `i2c`.
- Model 3 TaskSet `io_in_2`: Receives UDP messages from a socket on CE2 and sends the messages to the compute TaskSet via the message queue `i2c`.
- Message queue `i2c`: distributes the input received from the Model 3 `io_in` TaskSets to the `compute` TaskSet.

- Model 1 TaskSet `compute`: Receives messages from the message queue `i2c` and sends the messages to `xfer` via the voted message queue `c2x`. This TaskSet is responsible for the safe computation, which in the demonstrator is limited to simply forwarding the received message.
- Voted message queue `c2x`: votes the output received from compute and forwards it to `xfer`
- TaskSet `xfer`: Receives messages from the voted message queue `i2c` and CE-locally sends the message to the respective `io_out` TaskSet via the respective POSIX message queue `x2i`. The `xfer` ("transfer") TaskSet is required on the TAS Platform when connecting e.g. SIL4 (Model 1) TaskSets to basic integrity (Model 3) TaskSets to ensure replica-determinism and to support the flow control mechanism of the message queues. With regards to the information flow, this setup ensures that the output of `compute` is voted (by `c2x`) before being passed on to the basic integrity TaskSets.
- POSIX message queue `x2i`: On each CE a local POSIX message queue `x2i` is created to communicate from the Model 1 `xfer` TaskSet to the local Model 3 `io_out` TaskSet instance.
- TaskSet `io_out_0`: Receives messages from the CE 0 local POSIX message queue `x2i` and sends them to the basic-integrity application via the POSIX message queue `o2o`.
- TaskSet `io_out_1`: Receives messages from the CE 1 local POSIX message queue `x2i` and sends them to the basic-integrity application via the POSIX message queue `o2o`.
- TaskSet `io_out_2`: Receives messages from the CE 2 local POSIX message queue `x2i` and sends them to the basic-integrity application via the POSIX message queue `o2o`.
- POSIX message queue `o2o`: CE local queue used for communication between safe Ping application and basic-integrity Ping application.

The detailed basic-integrity Ping application consists of the `in_out2` Model 3 TaskSets, which are started on each CE and read from the `o2o` POSIX message queue and output the read message via UDP if they are configured to send.

The Ping application uses also the TAS Platform recovery to demonstrate that the loss of one CE and its recovery do not disturb the overall service.

This demonstrator shows the feasibility of safe and non-safe input, output, as well as the integration of safe computation and basic integrity computation on top of the modular platform TAS Platform.

4 CONDUCTED TESTS AND THEIR RESULTS

Main objective of the WP36 "Onboard Platform Demonstrator" is to demonstrate the feasibility of general concepts to satisfy the User Stories selected in D36.1 User Stories and Test Cases. Such proven feasibility shall guide the direction of further studies and lower the business risk of product development based on bespoken concepts.

D36.2 sets the focus is on a set of basic platform User Stories realising concepts for Deployment & Orchestration, Modularity and Communication. The following table provides an overview of such User Stories. The reference number refers to the chapter in D36.1 User Stories & Test Cases [5]. The User Stories validated in task 36.2 have been selected to lay the foundation for the next demonstrator implementation steps. At the same time also their executed validation process can now serve as a blueprint for how to validate further User Stories in the upcoming implementation tasks. Details related to conducted testing are provided in the chapter 4.2 Tested User Stories.

Ref. to [5]	Description	Val. in Task 36.2[1]
Modular Computing Platform – Deployment & Orchestration		
2.1.1	Run a Parallel Basic Integrity Application on Platform	Yes
2.1.2	Run a Basic Integrity Application on Safety Layer / RTE	Yes
2.1.3	Run a Safe Application (up to SIL 4) on Safety Layer / RTE	Yes
2.1.4	Run Multiple Applications	Yes[2]
2.1.5	Execute Declarative Configuration	Yes[2]
2.1.6	Restart Failing Replicas	No
2.1.7	React to Hardware Failure	No
2.1.8	Failover to Cold Backup Hardware	No
2.1.9	Replace Failing Hardware	Yes[3]
Modular Computing Platform – Modularity		
2.2.1	Change to Different Hardware	No
2.2.2	Change to Different Platform or Safety Layer / RTE	No
2.2.3	Use Diverse Hardware	No
Modular Computing Platform – Communication		
2.3.1	Communicate Safe App on RTE ↔ Safe App on RTE	Yes
2.3.2	Communicate Safe App on RTE ↔ External Safe App	No
2.3.3	Communicate Safe App on RTE ↔ Basic Integrity App on RTE	Yes
2.3.4	Communicate Safe App on RTE ↔ Parallel Basic Integrity App	Yes

[1] All User Stories have been deployed and tested in a virtual execution and test environment
[2] Implicitly proven by the execution of the described Test Cases (chapter 4.2)
[3] Hardware is simulated with containers

Ref. to [5]	Description	Val. in Task 36.2[1]
2.3.5	Communicate Safe App on RTE ↔ External Basic Integrity App	No
2.3.6	Communicate Basic Integrity App on RTE ↔ Basic Integrity App on RTE	Yes
2.3.7	Communicate Basic Integrity App on RTE ↔ Parallel Basic Integrity App	Yes
2.3.8	Communicate Basic Integrity App on RTE ↔ External Basic Integrity App	Yes
2.3.9	Communicate Parallel Basic Integrity App ↔ External Basic Integrity App	No
2.3.10	Communicate Independent of RTE Instance	No
2.3.11	Communicate Independent of Replication	No

Table 4: Realised User Stories Overview

4.1 Test Strategy

This chapter describes the test strategy followed by the tested user stories in context of the D36.2 demonstrator.

In the sense of scientific research, we aim to support the hypothesis that the concepts for the modular platform as they are described in D36.1 Architecture as well as the Architecture Update chapter of this document satisfy the selected User Stories. This is done by validating the User Stories by the means of verifying concrete Test Cases that are implemented as part of the demonstrator.

For each User Story in scope, at least one Test Case is defined with the following properties:

- **Test ID**
  Unique title for the Test Case
- **Test Scenario**
  High-level, human readable summary of the Test Steps
- **Test Steps**
  Reproducible steps to be taken to deterministically generate the data that can be verified against a defined expected outcome

The detailed Test Steps are not disclosed in this document, because they might contain intellectual property belonging to the owners of the Implemented Modules.

For the matter of this work package, Test Cases should be agnostic of the concrete implementation of the system under test (e.g. its version and configuration) to allow regression tests and to avoid the possible confirmation bias that comes with white box testing.

For task 36.2 all relevant Test Cases have been implemented as automated test. On one hand this provides for a fast, easily reproduceable and deterministic test execution, on the other hand this method – compared to a tutorial for a manual test execution – guarantees that all necessary details for the test execution have been exhaustively defined.

It should be noted that due to the nature of the validated User Stories, most Test Cases implemented in task 36.2 are able to reuse a very similar test setup as it is described in the chapter Demo Applications.

Each execution of the Test Steps of a Test Case is referred to as a Test Run. The automatic execution of the Test Cases was done within pipelines that are described in the subchapter Continuous Integration / Continuous Deployment. Each Test Run can thus be uniquely referenced by the ID of the respective pipeline run which is given below in the format `#YYYYMMDD.n`.

Sufficient proof of the Test Run execution as well as its result needs to be retained together with all necessary means to independently reproduce the Test Run with the same results. (Note that repeating a Test Run must always lead to the same result as they specify a concrete version and configuration of the system under test. Different Test Runs of the same Test Case however can have different results based on the version and configuration of the system under test.)

4.1.1 Continuous Integration / Continuous Deployment

Automizing the process of integration testing and deployment of software changes is highly valuable. Not only because it might be more convenient and efficient, but even more important because it makes the process reliable and reproducible. As in product development the latter being especially important when using the deployment for the matter of scientific research.

For the demo applications and test cases in WP36 a continuous integration pipeline is used as depicted in Figure 19. Sources, configurations, artifacts and logs of pipeline executions have been retained as evidence.

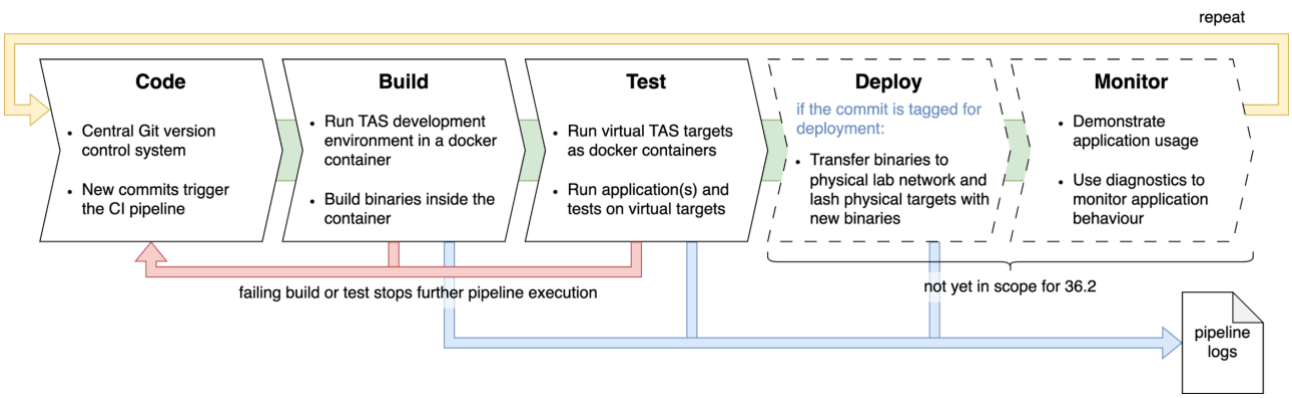


Figure 19: Continuous Integration Pipeline

The pipeline is executed using Microsoft-hosted pipeline agents out if the Azure Pipelines agent pool. It is configured by a YAML file defining the pipeline trigger, environment and steps (see listing below). The step type of `script` can be used to execute bash commands in the virtual machine.

At a later stage the pipelines can also be used to deploy the build artifacts to real hardware targets in the physical lab. For security reasons, the cloud pipeline will not be able to directly access those devices. Instead, an Azure Pipelines agent will run on a server within the lab network. The deploy steps of the pipeline job will be handed over to this self-hosted agent which only communicates to specifically whitelisted cloud resources.

`azure-pipelines.yml` for task 36.2 (extract):

```
trigger:
- main

pool:
  vmImage: ubuntu-latest

steps:
```

```
- checkout: self
- checkout: git://tas_ci_image
  lfs: true
- script: tar -xf [...]
  displayName: 'extract images'
- script: docker load < [...]
  displayName: 'load build container'
- script: |
        docker run -i --rm \
        -v $(pwd)/tas_ci_36-2:/appl \
        [...]
        erju_tasplf_build_container:2.5.2 \
        make -C /appl/src clean install \
        [...]
  displayName: 'compile in build container'
- script: docker load < [...]
  displayName: 'load runtime container'
- script: [...]
  displayName: 'configure VM kernel'
- script: sudo apt-get update && sudo apt-get install -y [...]
  displayName: 'install test tools'
- script: [...]
  displayName: 'perform tests'
- script: [...]
  displayName: 'archive logs for debugging in case of errors'
  condition: always()
```

4.2 TESTED USER STORIES

In the document WP 36.1 User Stories Test Cases [5] we have proposed and listed 64 user stories. This chapter summarises which user stories have been realised and tested with the WP 36.2 demo applications.

A selection of seven user stories has been identified as “basic platform user stories” in the sense of the D36.1 Statement of Work [6]. Test cases for those user stories were established and run in the Virtual Test Environment during this task 36.2.

For each tested user story we provide a unique Test Case ID, the tested User Story as reference to WP 36.1 User Stories Test Cases [5], a description of the Test Scenario, a reference to the Test Run, the verification / validation result and an excerpt of the retained log.

4.2.1 Run a Parallel Basic Integrity Application on Platform

Test ID: TEST 2.1.1 Run a Parallel Basic Integrity Application on Platform

Tested User Story: WP 36.1 User Stories Test Cases [5] chapter 2.1.1

Test Scenario: Start Ping applications on all CEs of the safe computing platform. Start syslog in parallel to the safe application. Check that the syslog daemon is running.

Reference to Test Run: #20240621.1 on 2024-06-21

Result: PASS

Log excerpt:

```
2024-06-21T09:16:02.1382374Z  => TEST 2.1.1 Run a Parallel Basic Integrity Application on
Platform
2024-06-21T09:16:02.1383493Z  ==> check that syslog-ng (Parallel Basic Integrity Application) is
running
2024-06-21T09:16:02.1394243Z bash-5.2# pidof syslog-ng && echo syslog-ng is running
2024-06-21T09:16:02.1406674Z bash-5.2# pidof syslog-ng && echo syslog-ng is running
```

```
2024-06-21T09:16:02.1427239Z bash-5.2# pidof syslog-ng && echo syslog-ng is running
2024-06-21T09:16:02.1429549Z  => TEST 2.1.1: PASS
```

4.2.2 Run a Basic Integrity Application on Safety Layer / RTE

Test ID: TEST 2.1.2 Run a Basic Integrity Application on Safety Layer / RTE

Tested User Story: WP 36.1 User Stories Test Cases [5] chapter 2.1.2

Test Scenario: Start Ping applications on all CEs of the safe computing platform. Check that basic integrity part of Ping application io_in is running on all CEs.

Reference to Test Run: #20240621.1 on 2024-06-21

Results: PASS

Log excerpt:

```
2024-06-21T09:16:02.1429728Z  => TEST 2.1.2 Run a Basic Integrity Application on Safety Layer /
RTE
2024-06-21T09:16:02.1430173Z  ==> check that io_in (Basic Integrity Application on Safety Layer)
is running
2024-06-21T09:16:02.1430348Z 107 106
2024-06-21T09:16:02.1430540Z syslog-ng is running
2024-06-21T09:16:02.1440361Z bash-5.2# pidof io_in && echo io_in is running
2024-06-21T09:16:02.1444733Z 107 106
2024-06-21T09:16:02.1445116Z syslog-ng is running
2024-06-21T09:16:02.1457644Z bash-5.2# pidof io_in && echo io_in is running
2024-06-21T09:16:02.1461699Z 107 106
2024-06-21T09:16:02.1462067Z syslog-ng is running
2024-06-21T09:16:02.1479457Z bash-5.2# pidof io_in && echo io_in is running
2024-06-21T09:16:02.1479888Z  => TEST 2.1.2: PASS
```

4.2.3 Run a Safe Application (up to SIL 4) on Safety Layer / RTE

Test ID: TEST 2.1.3 Run a Safe Application (up to SIL 4) on Safety Layer / RTE

Tested User Story: WP 36.1 User Stories Test Cases [5] chapter 2.1.3

Test Scenario: Start Ping applications on all CEs of the safe computing platform. Check that the safe compute program is running on all CEs.

Reference to Test Run: #20240621.1 on 2024-06-21

Results: PASS

Log excerpt:

```
2024-06-21T09:16:02.1480173Z  => TEST 2.1.3 Run a Safe Application (up to SIL 4) on Safety Layer
/ RTE
2024-06-21T09:16:02.1547205Z  ==> check that compute (Safe Application on Safety Layer) is
running
2024-06-21T09:16:02.1547377Z 232
2024-06-21T09:16:02.1547501Z io_in is running
2024-06-21T09:16:02.1547856Z pidof compute && echo compute is running
2024-06-21T09:16:02.1548179Z bash-5.2# pidof compute && echo compute is running
2024-06-21T09:16:02.1548297Z 242
2024-06-21T09:16:02.1548400Z compute is running
2024-06-21T09:16:02.1548497Z 234
2024-06-21T09:16:02.1548591Z io_in is running
2024-06-21T09:16:02.1548836Z bash-5.2# pidof compute && echo compute is runningpidof compute &&
echo compute is running
2024-06-21T09:16:02.1549169Z  => TEST 2.1.3: PASS
```

4.2.4 Communicate Safe App on RTE <-> Safe App on RTE

Test ID: TEST 2.3.1 Communicate Safe App on RTE <-> Safe App on RTE

Tested User Story: WP 36.1 User Stories Test Cases [5] chapter 2.3.1

Test Scenario: Start Ping applications on all CEs of the safe computing platform. Send a message via UDP to io_in_0 of Ping. Check that the second safe TaskSet xfer is receiving the message on all CEs.

Reference to Test Run: #20240621.1 on 2024-06-21

Results: PASS

Log excerpt:

```
2024-06-21T09:16:02.1549396Z  => TEST 2.3.1 Communicate Safe App on RTE <-> Safe App on RTE
2024-06-21T09:16:03.1525465Z  ==> message from xfer (Safe App) is received by compute (Safe App)
2024-06-21T09:16:03.1531131Z bash-5.2# tail /var/log/messages
2024-06-21T09:16:03.1539840Z 2024-06-21T09:15:57.100600+00:00 erju0 user notice cs[214]:
(fm_ts.c:583) (R:0, r:104, g:315.907065) TS PING:TS_compute@PING[1] UP (id=10)
2024-06-21T09:16:03.1540986Z 2024-06-21T09:15:57.100605+00:00 erju0 user notice cs[214]:
(fm_ts.c:583) (R:0, r:104, g:315.907065) TS PING:TS_compute@PING[2] UP (id=10)
2024-06-21T09:16:03.1548562Z 2024-06-21T09:15:57.100609+00:00 erju0 user notice cs[214]:
(fm_ts.c:518) (R:0, r:104, g:315.907065) TS PING:TS_compute@PING UP (id=10)
2024-06-21T09:16:03.1549098Z 2024-06-21T09:16:02.151054+00:00 erju0 user info PING:io_in0: io_in:
Received msg from UDP socket: TEST 2.3.1
2024-06-21T09:16:03.1555377Z 2024-06-21T09:16:02.151120+00:00 erju0 user info PING:io_in0: io_in:
Main loop, wait on socket
2024-06-21T09:16:03.1555771Z 2024-06-21T09:16:02.200757+00:00 erju0 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.1
2024-06-21T09:16:03.1556295Z 2024-06-21T09:16:02.200791+00:00 erju0 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:03.1556641Z 2024-06-21T09:16:02.300741+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.1
2024-06-21T09:16:03.1557177Z 2024-06-21T09:16:02.300876+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.1
2024-06-21T09:16:03.1557714Z 2024-06-21T09:16:02.301033+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.1
2024-06-21T09:16:03.1557940Z bash-5.2#
2024-06-21T09:16:03.1558045Z 244
2024-06-21T09:16:03.1558144Z compute is running
2024-06-21T09:16:03.1558488Z bash-5.2# tail /var/log/messages
2024-06-21T09:16:03.1558899Z 2024-06-21T09:15:56.902453+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:03.1562570Z 2024-06-21T09:15:57.100532+00:00 erju1 user notice cs[216]:
(fm_ts.c:583) (R:1, r:104, g:315.907065) TS PING:TS_compute@PING[0] UP (id=10)
2024-06-21T09:16:03.1565257Z 2024-06-21T09:15:57.100576+00:00 erju1 user notice cs[216]:
(fm_ts.c:583) (R:1, r:104, g:315.907065) TS PING:TS_compute@PING[1] UP (id=10)
2024-06-21T09:16:03.1568607Z 2024-06-21T09:15:57.100582+00:00 erju1 user notice cs[216]:
(fm_ts.c:583) (R:1, r:104, g:315.907065) TS PING:TS_compute@PING[2] UP (id=10)
2024-06-21T09:16:03.1569853Z 2024-06-21T09:15:57.100586+00:00 erju1 user notice cs[216]:
(fm_ts.c:518) (R:1, r:104, g:315.907065) TS PING:TS_compute@PING UP (id=10)
2024-06-21T09:16:03.1571676Z 2024-06-21T09:16:02.200685+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.1
2024-06-21T09:16:03.1572044Z 2024-06-21T09:16:02.200745+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:03.1572742Z 2024-06-21T09:16:02.300840+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.1
2024-06-21T09:16:03.1573117Z 2024-06-21T09:16:02.300947+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.1
2024-06-21T09:16:03.1573581Z 2024-06-21T09:16:02.301078+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.1
2024-06-21T09:16:03.1573822Z bash-5.2# tail /var/log/messages
```

```
2024-06-21T09:16:03.1574105Z 2024-06-21T09:15:56.902453+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:03.1574499Z 2024-06-21T09:15:57.100532+00:00 erju1 user notice cs[216]:
(fm_ts.c:583) (R:1, r:104, g:315.907065) TS PING:TS_compute@PING[0] UP (id=10)
2024-06-21T09:16:03.1574930Z 2024-06-21T09:15:57.100576+00:00 erju1 user notice cs[216]:
(fm_ts.c:583) (R:1, r:104, g:315.907065) TS PING:TS_compute@PING[1] UP (id=10)
2024-06-21T09:16:03.1575351Z 2024-06-21T09:15:57.100582+00:00 erju1 user notice cs[216]:
(fm_ts.c:583) (R:1, r:104, g:315.907065) TS PING:TS_compute@PING[2] UP (id=10)
2024-06-21T09:16:03.1575773Z 2024-06-21T09:15:57.100586+00:00 erju1 user notice cs[216]:
(fm_ts.c:518) (R:1, r:104, g:315.907065) TS PING:TS_compute@PING UP (id=10)
2024-06-21T09:16:03.1576143Z 2024-06-21T09:16:02.200685+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.1
2024-06-21T09:16:03.1576475Z 2024-06-21T09:16:02.200745+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:03.1576806Z 2024-06-21T09:16:02.300840+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.1
2024-06-21T09:16:03.1577160Z 2024-06-21T09:16:02.300947+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.1
2024-06-21T09:16:03.1577508Z 2024-06-21T09:16:02.301078+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.1
2024-06-21T09:16:03.1577751Z bash-5.2#  => TEST 2.3.1: PASS
```

4.2.5 Communicate Safe App on RTE <-> Basic Integrity App on RTE

Test ID: TEST 2.3.3 Communicate Safe App on RTE <-> Basic Integrity App on RTE

Tested User Story: WP 36.1 User Stories Test Cases [5] chapter 2.3.3

Test Scenario: Start Ping applications on all CEs of the safe computing platform. Send a message via UDP to io_in_0 of Ping. Check that the safe compute TaskSet receives the message from the non-safe io_in TaskSet on all CEs. Check that the non-safe io_out TaskSet receives the message from the safe xfer TaskSet on all CEs.

Reference to Test Run: #20240621.1 on 2024-06-21

Results: PASS

Log excerpt:

```
2024-06-21T09:16:03.1577967Z  => TEST 2.3.3 Communicate Safe App on RTE <-> Basic Integrity App
on RTE
2024-06-21T09:16:04.1575476Z  ==> message from io_in (Basic Integrity App) is received by compute
(Safe App)
2024-06-21T09:16:04.1583018Z tail /var/log/messages
2024-06-21T09:16:04.1595618Z 2024-06-21T09:16:02.300741+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.1
2024-06-21T09:16:04.1596025Z 2024-06-21T09:16:02.300876+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.1
2024-06-21T09:16:04.1596536Z 2024-06-21T09:16:02.301033+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.1
2024-06-21T09:16:04.1597017Z 2024-06-21T09:16:03.156447+00:00 erju0 user info PING:io_in0: io_in:
Received msg from UDP socket: TEST 2.3.3
2024-06-21T09:16:04.1597381Z 2024-06-21T09:16:03.156513+00:00 erju0 user info PING:io_in0: io_in:
Main loop, wait on socket
2024-06-21T09:16:04.1597839Z 2024-06-21T09:16:03.200761+00:00 erju0 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.3
2024-06-21T09:16:04.1598196Z 2024-06-21T09:16:03.200827+00:00 erju0 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:04.1598644Z 2024-06-21T09:16:03.300716+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.3
2024-06-21T09:16:04.1599893Z 2024-06-21T09:16:03.300872+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.3
2024-06-21T09:16:04.1600263Z 2024-06-21T09:16:03.300959+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.3
2024-06-21T09:16:04.1609608Z bash-5.2# tail /var/log/messages
```

```
2024-06-21T09:16:04.1610058Z 2024-06-21T09:16:02.200685+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.1
2024-06-21T09:16:04.1610409Z 2024-06-21T09:16:02.200745+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:04.1610876Z 2024-06-21T09:16:02.300840+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.1
2024-06-21T09:16:04.1611344Z 2024-06-21T09:16:02.300947+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.1
2024-06-21T09:16:04.1611717Z 2024-06-21T09:16:02.301078+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.1
2024-06-21T09:16:04.1612195Z 2024-06-21T09:16:03.200721+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.3
2024-06-21T09:16:04.1612778Z 2024-06-21T09:16:03.200790+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:04.1613151Z 2024-06-21T09:16:03.300804+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.3
2024-06-21T09:16:04.1613617Z 2024-06-21T09:16:03.300937+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.3
2024-06-21T09:16:04.1613997Z 2024-06-21T09:16:03.301046+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.3
2024-06-21T09:16:04.1628169Z bash-5.2# tail /var/log/messages
2024-06-21T09:16:04.1628512Z 2024-06-21T09:16:02.200685+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.1
2024-06-21T09:16:04.1628838Z 2024-06-21T09:16:02.200745+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:04.1629175Z 2024-06-21T09:16:02.300840+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.1
2024-06-21T09:16:04.1629524Z 2024-06-21T09:16:02.300947+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.1
2024-06-21T09:16:04.1629928Z 2024-06-21T09:16:02.301078+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.1
2024-06-21T09:16:04.1630283Z 2024-06-21T09:16:03.200721+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.3
2024-06-21T09:16:04.1630608Z 2024-06-21T09:16:03.200790+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:04.1630935Z 2024-06-21T09:16:03.300804+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.3
2024-06-21T09:16:04.1639043Z 2024-06-21T09:16:03.300937+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.3
2024-06-21T09:16:04.1639447Z 2024-06-21T09:16:03.301046+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.3
2024-06-21T09:16:04.1640105Z bash-5.2#  ==> message from xfer (Safe App) is received by io_out
(Basic Integrity App)
2024-06-21T09:16:04.1640248Z 233
2024-06-21T09:16:04.1640622Z io_in is running
2024-06-21T09:16:04.1640834Z bash-5.2# pidof compute && echo compute is running
2024-06-21T09:16:04.1640969Z 244
2024-06-21T09:16:04.1641067Z compute is running
2024-06-21T09:16:04.1659951Z bash-5.2# tail /var/log/messages
2024-06-21T09:16:04.1663601Z 2024-06-21T09:16:02.300748+00:00 erju2 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.1
2024-06-21T09:16:04.1663955Z 2024-06-21T09:16:02.300855+00:00 erju2 user info PING:io_out_2:
io_out: Received msg from xfer: TEST 2.3.1
2024-06-21T09:16:04.1664322Z 2024-06-21T09:16:02.300991+00:00 erju2 user info PING:io_out2_2:
io_out2: Received msg from io_out: TEST 2.3.1
2024-06-21T09:16:04.1664649Z 2024-06-21T09:16:02.301038+00:00 erju2 user info PING:io_out2_2:
io_out2: Sending to socket
2024-06-21T09:16:04.1664997Z 2024-06-21T09:16:03.200822+00:00 erju2 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.3
2024-06-21T09:16:04.1665504Z 2024-06-21T09:16:03.200855+00:00 erju2 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:04.1665845Z 2024-06-21T09:16:03.300703+00:00 erju2 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.3
2024-06-21T09:16:04.1666244Z 2024-06-21T09:16:03.300813+00:00 erju2 user info PING:io_out_2:
io_out: Received msg from xfer: TEST 2.3.3
```

```
2024-06-21T09:16:04.1666603Z 2024-06-21T09:16:03.300895+00:00 erju2 user info PING:io_out2_2:
io_out2: Received msg from io_out: TEST 2.3.3
2024-06-21T09:16:04.1666926Z 2024-06-21T09:16:03.300931+00:00 erju2 user info PING:io_out2_2:
io_out2: Sending to socket
2024-06-21T09:16:04.1667103Z  => TEST 2.3.3: PASS
```

4.2.6 Communicate Safe App on RTE <-> Parallel Basic Integrity App

Test ID: TEST 2.3.4 Communicate Safe App on RTE <-> Parallel Basic Integrity App

Tested User Story: WP 36.1 User Stories Test Cases [5] chapter 2.3.4

Test Scenario: Start Ping applications on all CEs of the safe computing platform. Send a message via UDP to io_in_0 of Ping. Check that the safe compute TaskSet logs the message to the parallel running syslog daemon on all CEs.

Reference to Test Run: #20240621.1 on 2024-06-21

Results: PASS

Log excerpt

```
2024-06-21T09:16:04.1667328Z  => TEST 2.3.4 Communicate Safe App on RTE <-> Parallel Basic
Integrity App
2024-06-21T09:16:05.1666951Z  ==> compute (Safe App) sends data to syslog (Parallel Basic
Integrity App)
2024-06-21T09:16:05.1667224Z tail /var/log/messages
2024-06-21T09:16:05.1667833Z 2024-06-21T09:16:02.300741+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.1
2024-06-21T09:16:05.1668191Z 2024-06-21T09:16:02.300876+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.1
2024-06-21T09:16:05.1668696Z 2024-06-21T09:16:02.301033+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.1
2024-06-21T09:16:05.1669044Z 2024-06-21T09:16:03.156447+00:00 erju0 user info PING:io_in0: io_in:
Received msg from UDP socket: TEST 2.3.3
2024-06-21T09:16:05.1669519Z 2024-06-21T09:16:03.156513+00:00 erju0 user info PING:io_in0: io_in:
Main loop, wait on socket
2024-06-21T09:16:05.1669970Z 2024-06-21T09:16:03.200761+00:00 erju0 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.3
2024-06-21T09:16:05.1670327Z 2024-06-21T09:16:03.200827+00:00 erju0 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:05.1673490Z 2024-06-21T09:16:03.300716+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.3
2024-06-21T09:16:05.1674009Z 2024-06-21T09:16:03.300872+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.3
2024-06-21T09:16:05.1674366Z 2024-06-21T09:16:03.300959+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.3
2024-06-21T09:16:05.1689073Z bash-5.2# tail /var/log/messages
2024-06-21T09:16:05.1689474Z 2024-06-21T09:16:03.300716+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.3
2024-06-21T09:16:05.1692258Z 2024-06-21T09:16:03.300872+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.3
2024-06-21T09:16:05.1693109Z 2024-06-21T09:16:03.300959+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.3
2024-06-21T09:16:05.1693673Z 2024-06-21T09:16:04.165515+00:00 erju0 user info PING:io_in0: io_in:
Received msg from UDP socket: TEST 2.3.4
2024-06-21T09:16:05.1694029Z 2024-06-21T09:16:04.165577+00:00 erju0 user info PING:io_in0: io_in:
Main loop, wait on socket
2024-06-21T09:16:05.1694538Z 2024-06-21T09:16:04.200675+00:00 erju0 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.4
2024-06-21T09:16:05.1694873Z 2024-06-21T09:16:04.200715+00:00 erju0 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:05.1695363Z 2024-06-21T09:16:04.300708+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.4
```

```
2024-06-21T09:16:05.1695706Z 2024-06-21T09:16:04.300861+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.4
2024-06-21T09:16:05.1696511Z 2024-06-21T09:16:04.300974+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.4
2024-06-21T09:16:05.1696947Z bash-5.2# tail /var/log/messages
2024-06-21T09:16:05.1697246Z 2024-06-21T09:16:02.200685+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.1
2024-06-21T09:16:05.1697701Z 2024-06-21T09:16:02.200745+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:05.1698047Z 2024-06-21T09:16:02.300840+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.1
2024-06-21T09:16:05.1701163Z 2024-06-21T09:16:02.300947+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.1
2024-06-21T09:16:05.1701950Z 2024-06-21T09:16:02.301078+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.1
2024-06-21T09:16:05.1702432Z 2024-06-21T09:16:03.200721+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.3
2024-06-21T09:16:05.1702881Z 2024-06-21T09:16:03.200790+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:05.1703234Z 2024-06-21T09:16:03.300804+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.3
2024-06-21T09:16:05.1703701Z 2024-06-21T09:16:03.300937+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.3
2024-06-21T09:16:05.1704077Z 2024-06-21T09:16:03.301046+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.3
2024-06-21T09:16:05.1704479Z bash-5.2# tail /var/log/messages
2024-06-21T09:16:05.1713255Z 2024-06-21T09:16:03.200721+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.3
2024-06-21T09:16:05.1713631Z 2024-06-21T09:16:03.200790+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:05.1714225Z 2024-06-21T09:16:03.300804+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.3
2024-06-21T09:16:05.1714756Z 2024-06-21T09:16:03.300937+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.3
2024-06-21T09:16:05.1715133Z 2024-06-21T09:16:03.301046+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.3
2024-06-21T09:16:05.1715697Z 2024-06-21T09:16:04.200754+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.4
2024-06-21T09:16:05.1716029Z 2024-06-21T09:16:04.200781+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:05.1716516Z 2024-06-21T09:16:04.300736+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.4
2024-06-21T09:16:05.1716855Z 2024-06-21T09:16:04.300890+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.4
2024-06-21T09:16:05.1717302Z 2024-06-21T09:16:04.301056+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.4
2024-06-21T09:16:05.1717694Z bash-5.2# tail /var/log/messages
2024-06-21T09:16:05.1720556Z 2024-06-21T09:16:03.200721+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.3
2024-06-21T09:16:05.1721054Z 2024-06-21T09:16:03.200790+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:05.1721394Z 2024-06-21T09:16:03.300804+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.3
2024-06-21T09:16:05.1721743Z 2024-06-21T09:16:03.300937+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.3
2024-06-21T09:16:05.1722097Z 2024-06-21T09:16:03.301046+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.3
2024-06-21T09:16:05.1730472Z 2024-06-21T09:16:04.200754+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.4
2024-06-21T09:16:05.1730820Z 2024-06-21T09:16:04.200781+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:05.1731164Z 2024-06-21T09:16:04.300736+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.4
2024-06-21T09:16:05.1731505Z 2024-06-21T09:16:04.300890+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.4
```

```
2024-06-21T09:16:05.1731862Z 2024-06-21T09:16:04.301056+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.4
2024-06-21T09:16:05.1732093Z bash-5.2#  => TEST 2.3.4: PASS
```

4.2.7 Communicate Basic Integrity App on RTE <-> Basic Integrity App on RTE

Test ID: TEST 2.3.6 Communicate Basic Integrity App on RTE <-> Basic Integrity App on RTE

Tested User Story: WP 36.1 User Stories Test Cases [5] chapter 2.3.6

Test Scenario: Start Ping applications on all CEs of the safe computing platform. Send a message via UDP to io_in_0 of Ping. Check that the message is passed from the non-safe io_out TaskSets to the non-safe io_out2 TaskSets on all CEs.

Reference to Test Run: #20240621.1 on 2024-06-21

Results: PASS

Log excerpt:

```
2024-06-21T09:16:05.1732337Z  => TEST 2.3.6 Communicate Basic Integrity App on RTE <-> Basic
Integrity App on RTE
2024-06-21T09:16:06.1723996Z  ==> io_out (Basic Integrity App) sends data to io_out2 (Basic
Integrity App)
2024-06-21T09:16:06.1730648Z tail /var/log/messages
2024-06-21T09:16:06.1756111Z 2024-06-21T09:16:04.300708+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.4
2024-06-21T09:16:06.1756924Z 2024-06-21T09:16:04.300861+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.4
2024-06-21T09:16:06.1757801Z 2024-06-21T09:16:04.300974+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.4
2024-06-21T09:16:06.1758412Z 2024-06-21T09:16:05.171152+00:00 erju0 user info PING:io_in0: io_in:
Received msg from UDP socket: TEST 2.3.6
2024-06-21T09:16:06.1758779Z 2024-06-21T09:16:05.171218+00:00 erju0 user info PING:io_in0: io_in:
Main loop, wait on socket
2024-06-21T09:16:06.1759393Z 2024-06-21T09:16:05.200668+00:00 erju0 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.6
2024-06-21T09:16:06.1763133Z 2024-06-21T09:16:05.200761+00:00 erju0 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:06.1763841Z 2024-06-21T09:16:05.300963+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.6
2024-06-21T09:16:06.1764227Z 2024-06-21T09:16:05.301063+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.6
2024-06-21T09:16:06.1769640Z 2024-06-21T09:16:05.301197+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.6
2024-06-21T09:16:06.1770136Z bash-5.2# tail /var/log/messages
2024-06-21T09:16:06.1770455Z 2024-06-21T09:16:04.200754+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.4
2024-06-21T09:16:06.1771018Z 2024-06-21T09:16:04.200781+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:06.1771373Z 2024-06-21T09:16:04.300736+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.4
2024-06-21T09:16:06.1771973Z 2024-06-21T09:16:04.300890+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.4
2024-06-21T09:16:06.1772329Z 2024-06-21T09:16:04.301056+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.4
2024-06-21T09:16:06.1772958Z 2024-06-21T09:16:05.200679+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.6
2024-06-21T09:16:06.1773361Z 2024-06-21T09:16:05.200822+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:06.1773700Z 2024-06-21T09:16:05.301065+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.6
2024-06-21T09:16:06.1774038Z 2024-06-21T09:16:05.301251+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.6
```

```
2024-06-21T09:16:06.1774393Z 2024-06-21T09:16:05.301343+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.6
2024-06-21T09:16:06.1774698Z bash-5.2# bash-5.2# tail /var/log/messages
2024-06-21T09:16:06.1774990Z 2024-06-21T09:16:04.300687+00:00 erju2 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.4
2024-06-21T09:16:06.1775331Z 2024-06-21T09:16:04.300876+00:00 erju2 user info PING:io_out_2:
io_out: Received msg from xfer: TEST 2.3.4
2024-06-21T09:16:06.1775687Z 2024-06-21T09:16:04.301049+00:00 erju2 user info PING:io_out2_2:
io_out2: Received msg from io_out: TEST 2.3.4
2024-06-21T09:16:06.1776011Z 2024-06-21T09:16:04.301059+00:00 erju2 user info PING:io_out2_2:
io_out2: Sending to socket
2024-06-21T09:16:06.1776356Z 2024-06-21T09:16:05.200728+00:00 erju2 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.6
2024-06-21T09:16:06.1776677Z 2024-06-21T09:16:05.200814+00:00 erju2 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:06.1777014Z 2024-06-21T09:16:05.300979+00:00 erju2 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.6
2024-06-21T09:16:06.1777356Z 2024-06-21T09:16:05.301165+00:00 erju2 user info PING:io_out_2:
io_out: Received msg from xfer: TEST 2.3.6
2024-06-21T09:16:06.1777711Z 2024-06-21T09:16:05.301270+00:00 erju2 user info PING:io_out2_2:
io_out2: Received msg from io_out: TEST 2.3.6
2024-06-21T09:16:06.1778188Z 2024-06-21T09:16:05.301278+00:00 erju2 user info PING:io_out2_2:
io_out2: Sending to socket
2024-06-21T09:16:06.1778416Z bash-5.2#  => TEST 2.3.6: PASS
```

4.2.8 Communicate Basic Integrity App on RTE <-> Parallel Basic Integrity App

Test ID: TEST 2.3.7 Communicate Basic Integrity App on RTE <-> Parallel Basic Integrity App

Tested User Story: WP 36.1 User Stories Test Cases [5] chapter 2.3.7

Test Scenario: Start Ping applications on all CEs of the safe computing platform. Send a message via UDP to io_in_0 of Ping. Check that the non-safe io_out logs the data to the parallel running syslog daemon on all CEs.

Reference to Test Run: #20240621.1 on 2024-06-21

Results: PASS

Log excerpt:

```
2024-06-21T09:16:06.1778651Z  => TEST 2.3.7 Communicate Basic Integrity App on RTE <-> Parallel
Basic Integrity App
2024-06-21T09:16:07.1776561Z  ==> io_out (Basic Integrity App) sends data to syslog (Parallel
Basic Integrity App)
2024-06-21T09:16:07.1780360Z tail /var/log/messages
2024-06-21T09:16:07.1793362Z 2024-06-21T09:16:05.300963+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.6
2024-06-21T09:16:07.1793727Z 2024-06-21T09:16:05.301063+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.6
2024-06-21T09:16:07.1794098Z 2024-06-21T09:16:05.301197+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.6
2024-06-21T09:16:07.1794446Z 2024-06-21T09:16:06.176573+00:00 erju0 user info PING:io_in0: io_in:
Received msg from UDP socket: TEST 2.3.7
2024-06-21T09:16:07.1794788Z 2024-06-21T09:16:06.176644+00:00 erju0 user info PING:io_in0: io_in:
Main loop, wait on socket
2024-06-21T09:16:07.1795139Z 2024-06-21T09:16:06.200741+00:00 erju0 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.7
2024-06-21T09:16:07.1795486Z 2024-06-21T09:16:06.200780+00:00 erju0 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:07.1795814Z 2024-06-21T09:16:06.300767+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.7
2024-06-21T09:16:07.1796165Z 2024-06-21T09:16:06.300974+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.7
```

```
2024-06-21T09:16:07.1796517Z 2024-06-21T09:16:06.301069+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.7
2024-06-21T09:16:07.1796705Z tail /var/log/messages
2024-06-21T09:16:07.1804439Z 2024-06-21T09:16:05.200679+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.6
2024-06-21T09:16:07.1804852Z 2024-06-21T09:16:05.200822+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:07.1805337Z 2024-06-21T09:16:05.301065+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.6
2024-06-21T09:16:07.1805700Z 2024-06-21T09:16:05.301251+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.6
2024-06-21T09:16:07.1806197Z 2024-06-21T09:16:05.301343+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.6
2024-06-21T09:16:07.1806676Z 2024-06-21T09:16:06.200671+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.7
2024-06-21T09:16:07.1807024Z 2024-06-21T09:16:06.200735+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:07.1807480Z 2024-06-21T09:16:06.300750+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.7
2024-06-21T09:16:07.1807842Z 2024-06-21T09:16:06.300883+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.3.7
2024-06-21T09:16:07.1808336Z 2024-06-21T09:16:06.301046+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.3.7
2024-06-21T09:16:07.1817151Z bash-5.2# tail /var/log/messages
2024-06-21T09:16:07.1822426Z 2024-06-21T09:16:05.300979+00:00 erju2 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.6
2024-06-21T09:16:07.1822812Z 2024-06-21T09:16:05.301165+00:00 erju2 user info PING:io_out_2:
io_out: Received msg from xfer: TEST 2.3.6
2024-06-21T09:16:07.1823163Z 2024-06-21T09:16:05.301270+00:00 erju2 user info PING:io_out2_2:
io_out2: Received msg from io_out: TEST 2.3.6
2024-06-21T09:16:07.1823501Z 2024-06-21T09:16:05.301278+00:00 erju2 user info PING:io_out2_2:
io_out2: Sending to socket
2024-06-21T09:16:07.1823837Z 2024-06-21T09:16:06.200672+00:00 erju2 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.7
2024-06-21T09:16:07.1824176Z 2024-06-21T09:16:06.200713+00:00 erju2 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:07.1824504Z 2024-06-21T09:16:06.300867+00:00 erju2 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.7
2024-06-21T09:16:07.1825069Z 2024-06-21T09:16:06.300967+00:00 erju2 user info PING:io_out_2:
io_out: Received msg from xfer: TEST 2.3.7
2024-06-21T09:16:07.1825415Z 2024-06-21T09:16:06.301050+00:00 erju2 user info PING:io_out2_2:
io_out2: Received msg from io_out: TEST 2.3.7
2024-06-21T09:16:07.1825814Z 2024-06-21T09:16:06.301057+00:00 erju2 user info PING:io_out2_2:
io_out2: Sending to socket
2024-06-21T09:16:07.1826036Z bash-5.2#  => TEST 2.3.7: PASS
```

4.2.9 Communicate Basic Integrity App on RTE <-> External Basic Integrity App

Test ID: TEST 2.3.8 Communicate Basic Integrity App on RTE <-> External Basic Integrity App

Tested User Story: WP 36.1 User Stories Test Cases [5] chapter 2.3.8

Test Scenario: Start Ping applications on all CEs of the safe computing platform. Send a message via UDP to the non-safe TaskSet io_in_0 of Ping. Receive the message from io_out2_0 via UDP. Note that the test itself using netcat is the external application in the sense of this user story.

Reference to Test Run: #20240621.1 on 2024-06-21

Results: PASS

Log excerpt:

```
2024-06-21T09:16:07.1826268Z  => TEST 2.3.8 Communicate Basic Integrity App on RTE <-> External
Basic Integrity App
```

```
2024-06-21T09:16:08.1826500Z  ==> io_in (Basic Integrity App) receives data from an UDP socket
which is sent by netcat (External Basic Integrity App)
2024-06-21T09:16:08.1832855Z bash-5.2# tail /var/log/messages
2024-06-21T09:16:08.1841383Z 2024-06-21T09:16:06.300767+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.7
2024-06-21T09:16:08.1841880Z 2024-06-21T09:16:06.300974+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.7
2024-06-21T09:16:08.1844788Z 2024-06-21T09:16:06.301069+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.7
2024-06-21T09:16:08.1845255Z 2024-06-21T09:16:07.182092+00:00 erju0 user info PING:io_in0: io_in:
Received msg from UDP socket: TEST 2.3.8
2024-06-21T09:16:08.1845686Z 2024-06-21T09:16:07.182165+00:00 erju0 user info PING:io_in0: io_in:
Main loop, wait on socket
2024-06-21T09:16:08.1846112Z 2024-06-21T09:16:07.200666+00:00 erju0 user info PING:compute:
compute: Received msg from io_in: TEST 2.3.8
2024-06-21T09:16:08.1846525Z 2024-06-21T09:16:07.200770+00:00 erju0 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:08.1846938Z 2024-06-21T09:16:07.300798+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.3.8
2024-06-21T09:16:08.1847367Z 2024-06-21T09:16:07.301015+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.3.8
2024-06-21T09:16:08.1847808Z 2024-06-21T09:16:07.301132+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.3.8
2024-06-21T09:16:08.1848135Z  ==> io_out2 (Basic Integrity App) sends data to an UDP socket which
is listened on by netcat (External Basic Integrity App)
2024-06-21T09:16:08.1848433Z TEST 2.3.1
2024-06-21T09:16:08.1848722Z TEST 2.3.3
2024-06-21T09:16:08.1849002Z TEST 2.3.4
2024-06-21T09:16:08.1849265Z TEST 2.3.6
2024-06-21T09:16:08.1849545Z TEST 2.3.7
2024-06-21T09:16:08.1849806Z TEST 2.3.8
2024-06-21T09:16:08.1850093Z => TEST 2.3.8: PASS
```

4.2.10 Replace Failing Hardware

Test ID: TEST 2.1.9 Replace Failing Hardware

Tested User Story: WP 36.1 User Stories Test Cases [5] chapter 2.1.9

Test Scenario: Start Ping applications on all CEs of the safe computing platform. Send messages and check that all CEs receive and process them. Shutdown the container of CE1 and check that the other CEs notice that CE1 is gone. Check that the TaskSets on CE0 and CE2 are still running and continue to process messages as before. Start CE1 again and wait until recovery is finished. Check that all TaskSets on all CEs are running again. Send a message and check that all CEs receive and process it.

Reference to Test Run: #20240621.1 on 2024-06-21

Results: PASS

Log excerpt:

```
2024-06-21T09:16:08.1850314Z  => TEST 2.1.9 Replace Failing Hardware
[...]
2024-06-21T09:16:16.3087440Z bash-5.2# tail /var/log/messages
2024-06-21T09:16:16.3087731Z 2024-06-21T09:16:09.300802+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.1.9 - 02
2024-06-21T09:16:16.3088213Z 2024-06-21T09:16:09.300919+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.1.9 - 02
2024-06-21T09:16:16.3088587Z 2024-06-21T09:16:09.301051+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.1.9 - 02
2024-06-21T09:16:16.3089081Z 2024-06-21T09:16:10.207832+00:00 erju0 user info PING:io_in0: io_in:
Received msg from UDP socket: TEST 2.1.9 - 03
```

```
2024-06-21T09:16:16.3089545Z 2024-06-21T09:16:10.207899+00:00 erju0 user info PING:io_in0: io_in:
Main loop, wait on socket
2024-06-21T09:16:16.3089917Z 2024-06-21T09:16:10.300819+00:00 erju0 user info PING:compute:
compute: Received msg from io_in: TEST 2.1.9 - 03
2024-06-21T09:16:16.3090389Z 2024-06-21T09:16:10.300851+00:00 erju0 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:16.3090735Z 2024-06-21T09:16:10.400943+00:00 erju0 user info PING:xfer: xfer:
Received msg from compute: TEST 2.1.9 - 03
2024-06-21T09:16:16.3091225Z 2024-06-21T09:16:10.401056+00:00 erju0 user info PING:io_out_0:
io_out: Received msg from xfer: TEST 2.1.9 - 03
2024-06-21T09:16:16.3091591Z 2024-06-21T09:16:10.401120+00:00 erju0 user info PING:io_out2_0:
io_out2: Received msg from io_out: TEST 2.1.9 - 03
2024-06-21T09:16:16.3092109Z bash-5.2# cat /var/diag/cs-0-0/diag/ft/cn
2024-06-21T09:16:16.3095418Z NODE  STATE     LAST-UP (round & time)      LAST-DOWN (round & time)
DOWNS  NAME         RENABLED
2024-06-21T09:16:16.3095606Z CN    OK             11          306.607            0
0.000     0  PING   1
2024-06-21T09:16:16.3095894Z CE[0] OK             11          306.607            0
0.000     0
2024-06-21T09:16:16.3096166Z CE[1] FAIL           11          306.607          247
330.207     1
2024-06-21T09:16:16.3096296Z CE[2] OK             11          306.607            0
0.000     0
2024-06-21T09:16:16.3103334Z bash-5.2# grep -c 'OK' /var/diag/cs-0-0/diag/ft/ts
[...]
2024-06-21T09:16:16.3146860Z  ==> only tasksets of CE 0 and 2 are up
2024-06-21T09:16:16.3147094Z  ==> (re-)start CE 1
2024-06-21T09:16:16.3147278Z spawn ./test.sh --ce 1 start-application
2024-06-21T09:16:17.4160670Z  ==> wait until recovery of CE 1 is finished
[...]
2024-06-21T09:16:29.4328544Z CE 1 recovery finished
2024-06-21T09:16:29.4329435Z cat /var/diag/cs-0-0/diag/ft/cn
2024-06-21T09:16:29.4333295Z bash-5.2# cat /var/diag/cs-0-0/diag/ft/cn
2024-06-21T09:16:29.4346884Z NODE  STATE     LAST-UP (round & time)      LAST-DOWN (round & time)
DOWNS  NAME         RENABLED
2024-06-21T09:16:29.4348095Z CN    OK             11          306.607            0
0.000     0  PING   1
2024-06-21T09:16:29.4348412Z CE[0] OK             11          306.607            0
0.000     0
2024-06-21T09:16:29.4348771Z CE[1] OK            350          340.507          247
330.207     1
2024-06-21T09:16:29.4348999Z CE[2] OK             11          306.607            0
0.000     0
2024-06-21T09:16:29.4355095Z bash-5.2# grep -c 'OK' /var/diag/cs-0-0/diag/ft/ts
2024-06-21T09:16:29.4369531Z 26
2024-06-21T09:16:29.4370080Z tail /var/log/messages
2024-06-21T09:16:29.4370612Z 2024-06-21T09:16:09.200633+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.1.9 - 02
2024-06-21T09:16:29.4371352Z 2024-06-21T09:16:09.200699+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:29.4371859Z 2024-06-21T09:16:09.300712+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.1.9 - 02
2024-06-21T09:16:29.4372602Z 2024-06-21T09:16:09.300869+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.1.9 - 02
2024-06-21T09:16:29.4373124Z 2024-06-21T09:16:09.301015+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.1.9 - 02
2024-06-21T09:16:29.4374906Z 2024-06-21T09:16:10.300743+00:00 erju1 user info PING:compute:
compute: Received msg from io_in: TEST 2.1.9 - 03
2024-06-21T09:16:29.4375453Z 2024-06-21T09:16:10.300786+00:00 erju1 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:29.4376575Z 2024-06-21T09:16:10.400796+00:00 erju1 user info PING:xfer: xfer:
Received msg from compute: TEST 2.1.9 - 03
2024-06-21T09:16:29.4377599Z 2024-06-21T09:16:10.400885+00:00 erju1 user info PING:io_out_1:
io_out: Received msg from xfer: TEST 2.1.9 - 03
2024-06-21T09:16:29.4378125Z 2024-06-21T09:16:10.401053+00:00 erju1 user info PING:io_out2_1:
io_out2: Received msg from io_out: TEST 2.1.9 - 03
```

```
[...]
2024-06-21T09:16:29.4460356Z  ==> all tasksets of all CEs are up
[...]
2024-06-21T09:16:30.4585904Z 2024-06-21T09:16:29.499903+00:00 erju2 user info PING:compute:
compute: Received msg from io_in: TEST 2.1.9 - 04
2024-06-21T09:16:30.4586241Z 2024-06-21T09:16:29.499992+00:00 erju2 user info PING:compute:
compute: waiting for message
2024-06-21T09:16:30.4586591Z 2024-06-21T09:16:29.599799+00:00 erju2 user info PING:xfer: xfer:
Received msg from compute: TEST 2.1.9 - 04
2024-06-21T09:16:30.4586940Z 2024-06-21T09:16:29.599918+00:00 erju2 user info PING:io_out_2:
io_out: Received msg from xfer: TEST 2.1.9 - 04
2024-06-21T09:16:30.4587305Z 2024-06-21T09:16:29.600074+00:00 erju2 user info PING:io_out2_2:
io_out2: Received msg from io_out: TEST 2.1.9 - 04
2024-06-21T09:16:30.4587632Z 2024-06-21T09:16:29.600108+00:00 erju2 user info PING:io_out2_2:
io_out2: Sending to socket
2024-06-21T09:16:30.4587861Z bash-5.2# TEST 2.1.9 - 04
2024-06-21T09:16:30.4588070Z => TEST 2.1.9: PASS
```

5 CONCLUSION

This report provides concrete results of the first implementation and testing phase, highlighting key factors and achievements that are crucial for project success.

The partners involved in the WP36, succeeded driving the project in a collaborative way to fulfil the foreseen goals of this first implementation task 36.2. This includes key achievements like validating the feasibility of mixed-SIL (Safety Integrity Level) deployments as proposed by the Modular Platform concepts of WP26 in a Virtual Test Environment.

The two chapters Implemented Modules and Conducted Tests and Their Results provide a detailed view towards the realisation of User Stories, as proposed in the Demonstrator Specification D36.1.

In total 12 User Stories have been realised, and the Demonstrator Specification has been further developed, providing a comprehensive level of detail as introduced in the chapter Architecture Update.

Adherence to the D36.1 Statement of Work [6] and proactive risk identification and mitigation are vital for successful implementation. It is important to remain agile in response to needed adaptability in project planning and execution to continuously evaluate and adjust implementation and test strategies accordingly. However, it is important to recognize that risks may still evolve or emerge throughout the project lifecycle, necessitating ongoing monitoring and adjustment of risk management.

Securing allocated funding and resources is paramount for project success. So is continued effort to ensure effective utilisation of allocated resources.

While simple demo applications have proven effective in demonstrating aspects of a complex architecture (e.g., hosting mixed-SIL applications), there is still much to realise in the upcoming implementation tasks. Ongoing research and development efforts are necessary to also prove the Modular Platform's benefits in the context of real railway applications, achieving the targeted TRL. Concretely, the integration of FRMCS, railway applications and diagnostics services will follow on rolling stock hardware in a physical laboratory environment.

A focus on automated testing has been proven for driving proper test documentation, but it is important to continuously enhance and refine testing methodologies to ensure comprehensive coverage and accuracy.

While knowledge sharing and best practices within the project are significant, there is a need for continued efforts to ensure that knowledge is effectively disseminated beyond the project boundaries. Maximizing impact and avoiding duplication of efforts requires sustained communication and collaboration with work package external stakeholders.

Engaging relevant stakeholders throughout the project is essential, but it is important to acknowledge that there may still be gaps in stakeholder alignment that need to be addressed to proceed and succeed according to the objectives of the project.

While this chapter may underscore the need for ongoing work and continuous improvement, this document highlights important achievements and progress in the project. By addressing the remaining challenges, refining strategies, and ensuring sustained collaboration and partner engagement, the project will be well positioned to achieve its objectives and make a significant impact in the field of rail automation and digitalization.

REFERENCES

[1] ASFAM. (2022). SOVD, Service-Oriented Vehicle Diagnostics, API Specification, version 1.0.0. ASAM. Retrieved from https://www.asam.net/standards/detail/sovd/

[2] Weidmann, T. (2022). ASAM SOVD v1.0 - Release Presentation. ASAM, Retrieved from https://www.asam.net/index.php?eID=dumpFile&t=f&f=5035&token=e64977333fa4379bc8222b5ed74849270627f91f

[3] VECTOR (2022) SOVD Hands On – SOVD in Practice Whitepaper V1.0 Retrieved from https://cdn.vector.com/cms/content/know-how/SOVD/doc/White_Paper_SOVD_Hands_On_SOVD_in_Practice_EN.pdf

[4] ERJU Innovation Pillar R2DATO (n.d.). D23.2: List of System Requirements, v1.1. Retrieved from https://projects.rail-research.europa.eu/eurail-fp2/deliverables/

[5] Europe's Rail. (2023). R2DATO, D36.1 – Demonstrator Specification, User Stories & Test Cases, https://projects.rail-research.europa.eu/eurail-fp2/deliverables/

[6] Europe's Rail. (2023). R2DATO, D36.1 – Demonstrator Specification, Statement of Work, https://projects.rail-research.europa.eu/eurail-fp2/deliverables/

[7] Europe's Rail. (2023). R2DATO, D36.1 – Demonstrator Specification, Architecture, https://projects.rail-research.europa.eu/eurail-fp2/deliverables/

[8] OCORA. (2022). High-Level Requirements - Generic Safe Computing Platform. Retrieved from https://github.com/OCORA-Public/Publications/blob/master/00_OCORA%20Latest%20Publications/Latest%20Release/OCORA-TWS03-020_Computing-Platform-Requirements.pdf

[9] OCORA (2023). System Requirements Specification - Monitoring, Diagnostics, Configuration & Maintenance subsystem. Retrieved from https://github.com/OCORA-Public/Publications/blob/master/10_OCORA%20Release%20R5/OCORA-TWS08-030_MDCM-SRS.pdf

[10] UIC (2023) FRMCS FFFIS Specification 7950-1.0.0 Retrieved from https://uic.org/rail-system/telecoms-signalling/frmcs

[11] UIC (2023) FRMCS Functional Requirements Specification FU-7120-1.0.0 Retrieved from https://uic.org/rail-system/telecoms-signalling/frmcs

[12] UIC (2023) TOBA Functional Requirements Specification FU-7510-1.0.0 Retrieved from https://uic.org/rail-system/telecoms-signalling/frmcs

[13] EuroSpec (2019). TCMS Data Service V1.0. Retrieved from https://eurospec.eu/tcms-data-service/

[14] YANG (2016) Data Modeling Language. Retrieved https://www.ietf.org/rfc/rfc6020.txt

[15] CLIPS (2023) - A Tool for Building Expert Systems. Retrieved from https://www.clipsrules.net

[16] Node-RED (2024) Low-code programming for event-driven applications, latest version v4.0.2. Retrieved from https://nodered.org

[17] Magic Quadrant for DevOps Platforms (2023) Retrieved from https://www.gartner.com/doc/reprints?id=1-2DHNOAC7&ct=230504