Rail to Digital automated up to autonomous train operation

D26.3 – Final Modular Platform requirements, architecture and specification

Due date of deliverable: 31/10/2024

First submission date: 30/09/2024

Final submission date: 25/07/2025

Leader/Responsible of this Deliverable: Maik Fox, Oliver Mayer-Buschmann / DB InfraGO AG

Reviewed: Y

Document status		
Revision	Date	Description
01	02/09/2024	First issue for internal Review
02	20/09/2024	Second issue for internal Review
03	30/09/2024	Issue for TMT Review
04	09/01/2025	Resolved review comments from JU
05	13/06/2025	Resolved review comments from MCP
06	25/07/2025	Resolved final comments from MCP

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 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
Maik Fox	DB InfraGO AG	Deliverable Lead, Executive Summary, Chapters 3, 4, 5, 8, 9, Appendix A, Appendix F
Oliver Mayer-Buschmann	DB InfraGO AG	Chapter 8
Patrick Marsch	DB InfraGO AG	Chapters 5, 8, Appendix D
Julian Wissmann	DB InfraGO AG	Chapter 8
Nikolaus König	Hitachi Rail GTS	Chapter 6
Ignacio Alguacil Ventas	INECO	Chapter 3.12
Giovanni Venturi	MER MEC	Chapters 7, Appendix C
Francesco Inzirillo	MER MEC	Chapters 1, 2, 7, Appendix C
Patrick Rozijn	NS	Chapters 3.8.2, 6 and 8
Thomas Martin	SBB	Chapter 3.7.2
Sonja Steffens	Siemens Mobility	Chapter 6, Appendix B, Appendix E
Thomas Bernburg	Siemens Mobility	Chapter 6

Disclaimer

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

EXECUTIVE SUMMARY

Computers are ubiquitous and their number is increasing, and the railway sector is no different. A huge amount of computing platforms is needed today, and even more will be needed in the future. They are essential in on-board systems, trackside elements and in data centres for efficient, reliable, and safe operation. By increasing the automation in the railway infrastructure including fully automated trains, new requirements and expectations towards these platforms will arise and add complexity, while the goal of safety must never be in jeopardy.

Supporting this growth in complexity and sheer number of computing systems and platforms, this work package, supported by railway companies and industry partners, aims to define a specification for modular platforms and deliver it to the ERJU System Pillar and future Innovation Pillar activities. Based on the consolidated learnings of the work package's first task and the intermediate deliverable of the second task, this third deliverable provides the specification for the "Modular Platforms Concept" (MPC).

The purpose of the MPC is to provide an up to SIL4 capable computing platform that enables the decoupling of hardware and software lifecycles and their respective update cycles. The MPC allows extensions to systems, updates, re-use of previous developments and consolidation of more software on less hardware.

This deliverable achieves the specification of the MPC by presenting the concept itself and providing a consolidated set of high-level platform requirements, collected from previous work, and updated to align with ERJU work. The MPC architecture is presented, introducing three different domains to help with the complexity of the topic: The Application-Level Platform Independence (ALPI) domain – with a strong focus on software and runtime environments, the Hardware-Level Platform Independence (HLPI) domain – focusing on hardware abstraction and virtualisation aspects, and the internal and external interfaces – providing interoperability to the outside and adaptability on the inside of the MPC. For each of the domains, a thorough discussion of its aspects and resulting requirements are presented.

The detailed analysis of the topics also resulted in an explicit list of open points that need further technical investigations and clarification – for example in demonstration projects – and, in some cases, more detailed input from the ERJU System Pillar domain than currently available.

As of today, the MPC specification cannot be implemented immediately in a way that would reach all its goals, due to the many important interchangeability details still being worked on in ERJU. Nevertheless, together with the future results of work package 36 (the on-board platform demonstrator), the ERJU Innovation Pillar FP2 "R2DATO" will be able to showcase its vision of railway modular compute platforms on a solid foundation.

This deliverable concludes task 2 of work package 26. Modular certification approaches will be discussed in task 3.

ABBREVIATIONS AND ACRONYMS

Note: A glossary for ERJU terms can be found in chapter 3.7.2, a local glossary in Appendix F.

ALPI	Application-level Platform Independence
ATO	Automatic Train Operation
BTM	Balise Transmission Module
COTS	Commercial Off The Shelf
CCS	Command, Control and Signalling
CPI	Compatible Platform Implementation
DDP	Deliverable Development Plan
DMI	Driver Machine Interface
ERTMS	European Rail Traffic Management System
ETCS	European Train Control System
FRMCS	Future Railway Mobile Communication System
GoA	Grade of Automation
HLPI	Hardware-level Platform Independence
HW	Hardware
ICT	Information and Communications Technology
OCORA	Open CSS On-Board Reference Architecture
OT	Operational Technology
PI API	Platform-Independent Application Programming Interface
POSIX	Portable Operating System Interface
R2DATO	Rail to digital automated up to autonomous train operation
RBC	Radio Block Centre
RCA	Reference CCS Architecture
RTE	Run Time Environment
SCP	Safe Computing Platform
SRACs	Safety Related Application Conditions
SW	Software
TCMS	Train Control Management System

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This document, inside the framework of the R2DATO project, provides the detailed specification on Modular Platforms developed in the Task 26.2.

The objective of the activities in WP26 are to obtain: i) a Modular Platform independent from the applied functions, and ii) a platform that can be introduced into the railway environment on-board and wayside.

One of the objectives of this work is to consolidate the Modular Platform definition describing its architecture and the interactions between the different modules that compose it.

The activities that produced the document are the continuation of this work package's existing deliverables, previous projects such as RCA/OCORA, Shift2Rail and others. See chapter 2.3 for details on input documents used.

The partners involved in this project have leveraged their extensive skills and expertise acquired over the years, including their contributions to previous projects and the development of proprietary platforms. As a result, they have collaborated to create a comprehensive summary, the details of which are outlined in this document.

The goal followed is to define the basis for providing a Modular Platform that allows to implement Functional Systems in a platform independent manner (HLPI), and also Functional Applications (ALPI) so that Business Logic is agnostic as much as possible about complexity of HW, Safety, Security and that can be used for Safe and Non-Safe functions and is applicable for wayside and on-board systems. Furthermore, the Modular Platform enables aggregating systems of different vendors on same COTS hardware.

Within R2DATO, the Modular Platform Concept discussed in deliverable D26.3 are relevant for migrating existing products as well as new functionalities, due to their need of regular updates and more advanced computational needs. These fields of applicability will need investigation in a future phase of R2DATO, though. This work package focuses in the current phase on the use of standard COTS components.

The work in this deliverable unifies on-board and wayside use cases as much as possible. However, there are environmental constraints like available volume and climate conditions, leading to differences in size and available computing resources in a typical system. In the end no major impacts on the basic concept have been identified in this deliverable, even though on-board and wayside realisations may choose partly different solutions depending on their specific needs and their technical and normative constraints.

The deliverable describes the approach for a suitable Modular Platform by describing the goals and concept in detail, presenting an updated set of high-level requirements for such a platform, deriving an architectural basis, and detailing out three main components of the concept: application-level platform independence, hardware-level platform independence, and necessary interfaces.

1.1 SCOPE

This document constitutes the Deliverable D26.3 "Final Modular Platform requirements, architecture and specification" in the framework of the work package 26 of FP2 R2DATO.

The document offers a collection of Operational Requirements, Use Case, Architecture and Functional Requirements divided into categories as indicated in Table 1.

1.2 Document Structure

The following table outlines the document structure.

§	Title	Description
1	Introduction	Provide an overview of the entire document.
2	Development Methodology	Describe the activities performed for obtaining this document.
3	Modular Platforms Concept (MPC)	Reports all the concepts that were discussed and agreed during the activities of WP26.
4	Modular Platforms Requirements	Explains the sources and methodology of high-level requirements collection.
5	Modular Platforms Architecture	Describes the architecture used as a basis for this document.
6	Hardware-Level Platform Independence (HLPI)	This chapter describes the approach followed for obtaining a set of hardware platform independence principles, functions and their interfaces.
7	Application-Level Platform Independence (ALPI)	This chapter describes the approach followed for obtaining a set of application platform independence principles, functions and their interfaces.
8	Management, Diagnostics and Security related Interfaces	Explains the details of internal interfaces inside of the modular platform.
9	Conclusions	In this chapter are summarised the achievements of the task 26.2 results and reported in the deliverable D26.3, as well as next steps.
	References	Provides relevant references used throughout the document.
A	MPC Requirements	Reports high-level requirements of the MPC.
B	HLPI Requirements	Reports requirements of the HLPI.
C	ALPI Requirements	Reports requirements of the ALPI.
D	Management, Diagnostics and Security related Interface Requirements	Reports requirements of Management, Diagnostics and Security related interfaces.
E	Collected Open Points for the MPC	Reports all open points collected for future work in the context of the Modular Platform Concept.
F	MPC Glossary	Reports terms introduced in this document.

Table 1: Document Structure

1.3 LIMITATIONS

No additional specific limitations to be added to those present in chapter 3.6.

2 DEVELOPMENT METHODOLOGY

In this chapter, the methodology on how this deliverable has been developed is discussed. The methodology section consists of four sections: i) Deliverable Objectives; ii) Process Overview; iii) Existing and Relevant Documents; iv) Methodology for Deliverable Development.

2.1 DELIVERABLE OBJECTIVES

This deliverable is created on the basis of the guidelines as described in the Grant Agreement (GA). From GA the Task 26.2 of WP26 has this mandate:

> In this activity, based on Task 26.1, the detailed work on any specifications needed in the context of modular platforms is performed. This would likely (subject to the agreements in Task 26.1) include:
>
> - The further collection and consolidation of requirements from railway applications to IT platforms.
> - The further development of the architecture of IT platforms for the future railway system.
> - The further specification of the API between (safety-critical and non-safety-critical) railway applications and IT platforms.
> - Specification of common diagnostics, orchestration, and remote update interfaces.
> - Specification of common safety-related application constraints (SRACs), to the extent that these would be needed.

The input available for the creation of this deliverable was as follows:

1. Relevant input from partner projects, mainly from System Pillar domains deliverables, was collected, analysed and if relevant, included in the deliverable. No further input was considered after May 2024.
2. The Deliverable D26.2, which contains what was achieved in the initial phases of task 26.2 and which this document is based upon.
3. Feedback from the “Onboard Platform Demonstator”, R2DATO work package 36.

2.2 PROCESS OVERVIEW

The Figure 1 shows the process followed in task 26.2 to develop deliverable D26.3.

As can be seen from the process shown in the Figure 1, the main steps that participated in the achievement of the final document are indicated.

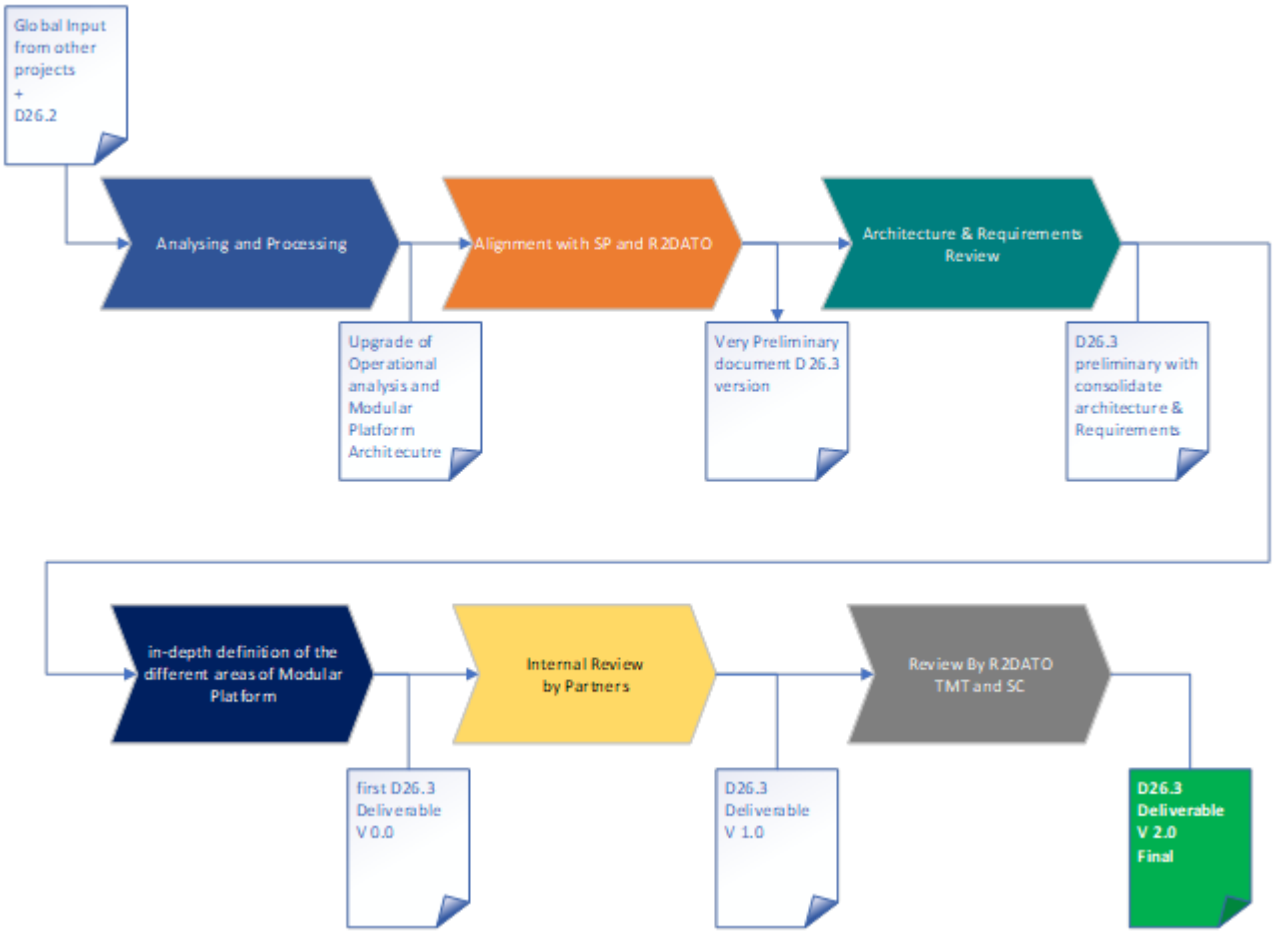


Figure 1: Process Overview

2.3 Existing and Relevant Documents

As input to the Work Package 26 process, the state of the art was considered and deliverables from past projects were identified and actively requested at the Work Package level. For this process, inputs were collected from several relevant projects:

- Computing Platform – Whitepaper from OCORA [5]
- Computing Platform – Requirements from OCORA [6]
- Computing Platform – Specification of the PI API between Application and Platform from OCORA [7]
- OCORA-TWS08-030 MDCM SRS [10]
- OCORA-TWS01-035 CCS On-Board Architecture [11]
- OCORA-BWS02-030 Technical Slide Deck [12]
- ERJU System Pillar – Computation Environment Domain [13]
- ERJU System Pillar, Computing Environment – Deliverable “Recommendation on interfaces to be standardised [14]
- ERJU System Pillar, Computing Environment – Deliverable “Operational Analysis Specification” [15]
- ERJU System Pillar – Common Business Objectives [16]

- ERJU System Pillar – Transversal Domain (not published yet)
- ERJU Innovation Pillar FP2 R2DATO, Work Package 26, Deliverable D26.1 "High level Consolidation" [18]
- ERJU Innovation Pillar FP2 R2DATO, Work Package 26, Deliverable D26.2 "Intermediate specification of the Modular Platform" [19]
- X2RAIL-3 Deliverable 8.2 [21]

To avoid further delay and ensure a viable result before the deadline M23 of WP26, the work package team started work on the basis of these deliverable and draft documents, establishing the potential gap with the intended WP26 results.

2.4 Methodology For Deliverable Development

The deliverable D26.3 as the successor of D26.2 can be considered as the final consolidation of specifications for the wave one of R2DATO. This work group relies on previous deliverables e.g., from X2RAIL and OCORA. In parallel, inputs were also sourced through diverse SP Task 2 domains. These became the fundamental inputs to start the process of input collection.

At the beginning of the activities a DDP (Deliverable Development Plan) was developed and agreed by all the partners, it was followed for the D26.3 writing activities.

The work carried out in Task 26.2 involved the study of the results obtained in previous projects.

Without prejudice to the valid work carried out in the previous projects, a series of points have been highlighted by the WP26 partners which are reported below:

- Identify suitable solutions;
- Try to reuse experience coming from specific partners;
- Integrate the parts coming from previous projects that were considered stable;
- Find a compromise between complexity and feasibility;
- Try to provide solutions that do not favour a specific partner/provider;
- Be able to have an application-independent solution;
- reuse of available standard COTS products on the market, taking full advantage of the evolution of ICT and OT technologies.

Once these activity development points have been identified, the first step was to define the operational aspects in which the Modular Platform shall operate.

After the chapters were drafted, the workgroup followed a structured approach from the point of drafting the chapters to finalizing of the deliverable with the required consensus and approval. In line with the DDP the following steps illustrate this process:

1. First development stage – responsibility of partner writing the chapters;
2. Review – responsibility of partner reviewing the chapters;
3. Second development stage – responsibility of partner writing the chapters;
4. Formal review – responsibility of partner reviewing the document;
5. Third development stage – responsibility of partner writing the document;

6. Finalized (stored in Cooperation Tool the Deliverable document D26.3) – responsibility of partner writing the document;

As can be seen in the process described, it provided with a good collaboration among the partners for writing and reviewing the chapters before agreeing on the finalized document.

3 MODULAR PLATFORMS CONCEPT (MPC)

The concept for modular platforms as presented in this deliverable enables interchangeability and maintainability for railway computing applications and the platforms they are deployed on. The concept is agnostic to trackside and on-board use, targets COTS hardware and integrates workloads with different criticality – basic integrity up to SIL4.

The results created by R2DATO work package 26 are based on several inputs. Previous work has already been captured in the first deliverable [18], and an intermediate version of the concept has been introduced in the second deliverable [19]. The basic architectural concepts, however, are taken directly from the ERJU System Pillar Computing Environment Domain deliverables ([14], [15], see the discussion in chapter 3.7) and are not developed in work package 26. Work package 26, however, decided to treat all defined interfaces with similar priority for this deliverable, extending our scope. Nevertheless, there has been and still is a constant expert exchange between work package 26 and the Computing Environment Domain.

The subchapters present the Modular Platform Concept's (MPC) purpose, scope, stakeholders, goals & non-goals, assumptions, and known issues and limitations. The alignment with the ERJU System Pillar activities is discussed in detail, including the glossary (see chapter 3.7.2) that has been developed in alignment with the SP CE domain. Details on the handling of PRAMSS in MPC are stated. Based on ERJU System Pillar input, user stories, and operational context and scenarios are given. The intended usage for the MPC and exemplary platform environments are discussed last.

The following three chapters will discuss in detail the important puzzle pieces of the Modular Platform Concept, namely hardware-level platform independence (HLPI) in chapter 6, application-level platform independence (ALPI) in chapter 7 and platform management interfaces in chapter 8.

Additional terms introduced for the MPC can be found in the glossary in Appendix F.

3.1 PURPOSE

The Modular Platform Concept is an extension of the ERJU System Pillar Computing Environment architecture proposal. The purpose of the Modular Platform Concept (MCP) – and of work package 26 – is to explore the feasibility of the following items and educate other R2DATO work packages and the System Pillar about MPC to foster its adaptation and integration.

MPC-P01 The decoupling of hardware and software lifecycles, leading to the decoupling of their respective update cycles.

MPC-P02 The decoupling of software with different lifecycles, such as security related components, services like diagnostics, base software, and applications.

MPC-P03 Providing a generic and basic framework for future Basic Integrity up to SIL4 applications.

MPC-P04 Enabling the extension of functionality to an already deployed modular platform, for example adding ATO to an existing CCS after initial deployment.

MPC-P05 Enabling re-use. For example, a diagnostics stack, e.g. as defined in EULYNX [2], can be used in different projects. Also, the need to deploy and maintain multiple variants of the same function can be reduced.

MPC-P06 Consolidating more software on less hardware.

MPC-P07 Shaping a picture of the feasibility, scenarios, and benefits around potential multi-vendor approaches (spanning hardware, consolidated systems, and software), including functional safety applications.[1]

3.2 Scope

The MPC is a generic approach to computing, embedded in a railway landscape. As such, it can potentially cover wayside and rolling stock systems.

As these environments might have specific requirements towards computing systems, the following scope limitations are necessary.

MPC-S01 Wayside and on-board are to be treated equally where possible. Where differences are encountered, the more encompassing variant guides the definition of the MPC, with potential simplifications stated.

MPC-S02 The MPC as defined in here is not intended to achieve full compatibility to all or even specific existing applications.

3.3 Stakeholders

Within ERJU, there are several stakeholders for the MPC.

- The ERJU System Pillar Computing Environment domain delivering the basis and analysing the work package 26 results
- Other ERJU System Pillar domains with interfaces towards or interest in the MPC
- The work package 26 members
- ERJU R2DATO, in particular the on-board platform demonstrator (R2DATO work package 36) as a potential specification consumer, validator, and feedback instance
- R2DATO Wave-2 work packages for the potential continuation (e.g. continuation in R2DATO, e.g., future in demonstrators) and other prototying projects

3.4 Goals & Non-Goals

For this deliverable on the MPC specification, the work package defined several goals and also explicit non-goals.

MPC-G01 Goal: Create a basic and initial technical reference respectively feasibility study for railway-grade modular platforms. Target audience: IMs, RUs, Industry. Focus questions:
Where are the pain points?
What are needs and benefits from standardization in a technical context?
What would need to be agreed on from a technical perspective to enable the concept?

MPC-G02 Goal: Create a basic set of requirements (for the layers respectively interfaces and between them) and suggestions how to evaluate them (potentially in R2DATO Wave-2)

[1] This is not going to be covered in this deliverable, but is rather the topic for the subsequent task 3 of work package 26. The contents of this deliverable will be an input to the activity, however.

against the stated MPC purposes with regards to: complexity, performance limitations, technical innovation needed, ...

MPC-G03 Goal: Shape and influence R2DATO Phase 2 with our output (future work and potential demonstration).

MPC-G04 Goal: Create a common reference document (this deliverable) for similar problems currently encountered (e.g., virtualization in data centres).

MPC-G05 Goal: Create a public reference for influencing future developments and procurement activities.

MPC-G06 Goal: Provide input usable for work package 26 task 3 (study on modular certification).

MPC-G07 Goal: Provide feedback to the System Pillar Computing Environment domain.

MPC-G08 Goal: Enable work package 36 demonstrator and demonstrators of future R2DATO Waves to incorporate and evaluate work package 26 MPC results.

MPC-G09 Non-Goal: Direct input for standardization.

MPC-G10 Non-Goal: Business analysis.

MPC-G11 Non-Goal: Direct input into the specification of work package 36 “On-board Platform Demonstrator”, as it is not feasible from a timeline perspective.

3.5 Assumptions

The MPC is based on several assumptions that were compiled by the work package members.

MPC-A01 The R2DATO Grant Agreement describes an MPC that is fully agnostic to its environment, so that all combinations on all systems are captured. For the actual work of this work package, the scope is assumed to be trackside and on-board systems with independent lifecycles of software and hardware.

MPC-A02 The MPC is based on the System Pillar Computing Environment domain architecture concept [14], its glossary (see chapter 3.7.2) and first insights out of their work on operational scenarios [15] as available by May 2024.

MPC-A03 The MPC’s interfaces provide the same functionality, independent of the technologies used. The Functional Systems state a clear demand on resources needed for their execution (memory, processing time, IO...).

MPC-A04 Functional Applications can assume to be free from unintended interference (from each other, and other parts of the MPC). Rationale: Memory & instruction corruption, and similar effects, are not allowed to happen unrecognized.

MPC-A05 Execution of safe and non-safe functions in mixed critically conditions on the same hardware is possible.

MPC-A06 Standardized modular safety certification approaches enriching the MPC will be investigated in work package 26 task 3.

MPC-A07 Resource partitioning shall only drive availability and security requirements but not safety requirements in the VE (Virtualization Environment). Rationale: IO access towards the hardware from the compartment is needed, e.g., to access an Ethernet controller. Resources can be memory, processing time, and IO. This is true for all SIL and BI cases.

MPC-A08 Application specific hardware is not part of the Modular Platform Concept (e.g., input/output controllers for special interfaces, hardware acceleration, etc.). As such, (external) communication is limited to IP-based interfaces.

MPC-A09 There are (soft/hard) real time requirements with guaranteed reaction times equal to or more than "1" second.[2]

MPC-A10 Composite fail safety (as defined by EN50129) is used in the MPC.

3.6 Known Issues & Limitations

Some aspects of the MPC are not going to be solved by this deliverable. These known issues and limitations are listed in this section. A potential solution option is given where known.

MPC-L01 Safe reaction times are set on system level and need to be budgeted to the participating functions that are implemented on a Modular Platform. This budgeting cannot be solved on the Modular Platform level, as such we cannot derive a universal number for the needed safe reaction time that is true for all possible scenarios. Timings nevertheless must be specified for actual implementations later to enable the purposes stated for the MPC. This information can potentially become part of the ERJU architecture, apportioned for the individual functions (e.g., as already done in EULYNX). If new applications impose reaction times that are beyond the scope of the MPC, a re-evaluation is necessary. Potentially impacted components of the MPC: time stamping, voting, message passing, and others.
Solution options: discussion in work package 26 task 3; bring issue to System Pillar (SP); use working assumption of 0.5 up to 1 seconds; use EULYNX numbers as an initial working hypothesis.

MPC-L02 Modular certification to enable all multi-vendor scenarios is not solved right now.
Solution options: discussion in work package 26 task 3; wait for SP PRAMS domain results.

MPC-L03 The benefits analysis depends on potential future demonstration activities (e.g., in R2DATO phase 2) of the MPC as presented in this deliverable.

MPC-L04 Requirements towards the platform from applications in the R2DATO and ERJU context are missing.
Solution options: Re-visit in R2DATO phase 2 when application needs are clearer.

MPC-L05 Specific hardware acceleration for ML applications is excluded and for future work.

3.7 Alignment with ERJU System Pillar activities

The Europe's Rail Joint Undertaking (ERJU) System Pillar (SP) Computing Environment (CE) domain is working on modular computing environments for the railway. The domain's goal is a holistic top-down approach, staying agnostic to implementation details and especially towards

[2] This number in isolation is not helpful for the Modular Platform specification. It is a budgeting issue from the system level to appropriate the reaction times across participating systems. See known issue MPC-L01 in chapter 3.6 for a detailed discussion.

trackside and on-board differences[3]. So far, the domain has released two deliverables: "Recommendation on interfaces to be standardised" [14] (referred to as "RIS" in this chapter) and "Operational Analysis Specification (OAS)" [15]. Furthermore, the domain is maintaining the glossary (see later in this document's chapter 3.7.2). Both documents and the aligned glossary are crucial inputs for the Modular Platform Concept in work package 26 and build its basis.

Other domains of relevance are the SP Transversal CCS (TCCS) domain and the SP PRAMSS domain. The alignment is captured in dedicated subchapters.

3.7.1 ERJU SP CE domain: RIS

The RIS [14] derived computing-relevant user stories from the "Common Business Objectives" provided by the System Pillar [16]. In summary, the user stories are asking for advanced functionality not available in today's railway- or on-board infrastructure, such as remote updates or hardware replacement with minimal (re-)certification effort (see also chapter 3.9). The user stories were assessed towards their benefits. Subsequently, an architecture was derived, together with a set of five interfaces named I1 to I5 (see below in Figure 2).

The interfaces are as follows:

- I1: External Diagnostics, Logging, Orchestration & IT Security Interface
  - IF-DIAGNOSTICS
  - IF-LOGGING
  - IF-ORCHESTRATION
  - IF-IT-SEC
- I2: Hardware Abstraction Interface
- I3: Virtualisation Interface
- I4: Basic Integrity Platform Independence Interface
- I5: Safe Platform Independence Interface

In its initial planning, WP26 did not address resp. expect working on hardware and virtualization topics (I2 and I3). Nevertheless, to reach a useful description level for a Modular Platform Concept following the input from the SP CE domain, it is necessary to include those, at least by defining assumptions and requirements (see chapter 6 for this new topic).

Several deployment options based on the architecture and the interface definition were given as examples. These examples are illustrated in Figure 3, showing five ways how the layers could be used, or their functionality included into solutions spanning multiple interfaces.

In a next step, user stories were mapped to these interfaces and a feasibility assessment per story was conducted, giving individual analyses for basic integrity and SIL applications before deriving an overall feasibility conclusion. Individual cost assessments on all five interfaces were the next step.

[3] For example, so far, the SP CE domain did not touch on on-board specifics, such as IO needs and interfacing to specialized hardware that would still be considered COTS, albeit not in the form of "standard servers", as there are standard hardware systems from some suppliers available. Another example could be Subset 147, which is Ethernet based, needed for on-board systems in the future and likely to be supported by the same specialized COTS hardware.

The document ends with a conclusion on the interfaces to be standardized, based on the metrics derived before, and is shown in Figure 4.

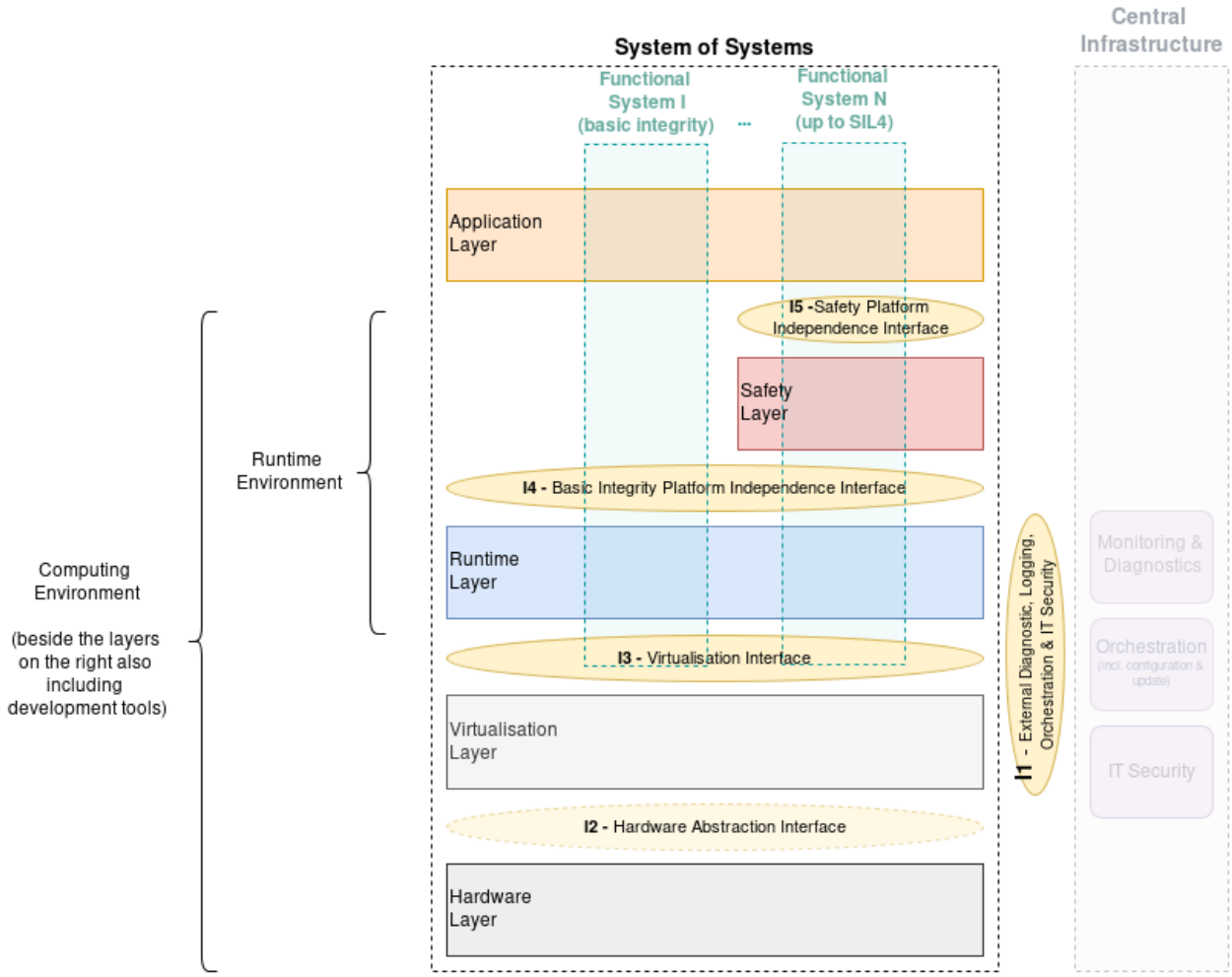
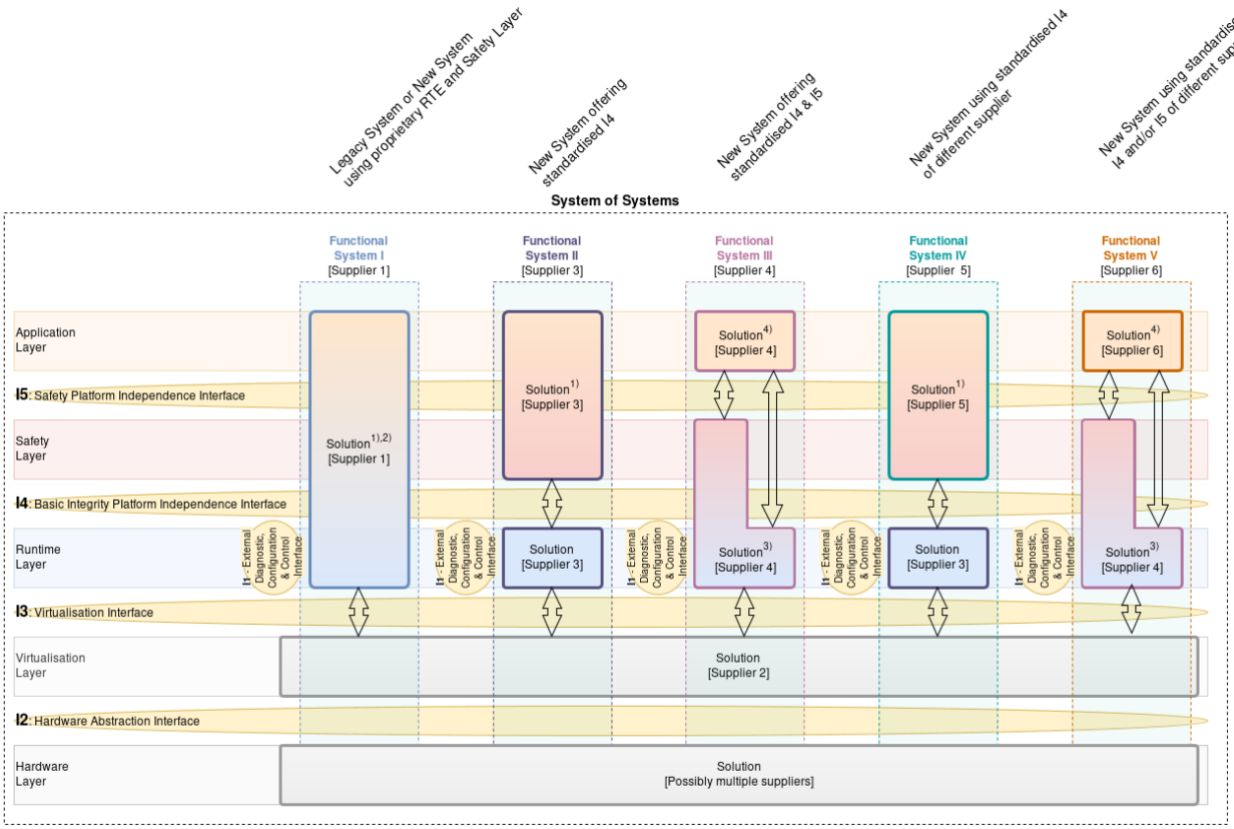


Figure 2: Overview of SP CE domain layer and interface structure, taken from [14]



1) In case the box implements a safe function, it includes a safety layer that may be proprietary.
2) This box could implement an existing legacy function or a new function.
3) This box implements a RTE and safety layer and exposes I4 and I5.
4) In case the box implements a safe function, it would use I5, otherwise it would use I4

Figure 3: Examples for deployment scenarios, taken from [14]

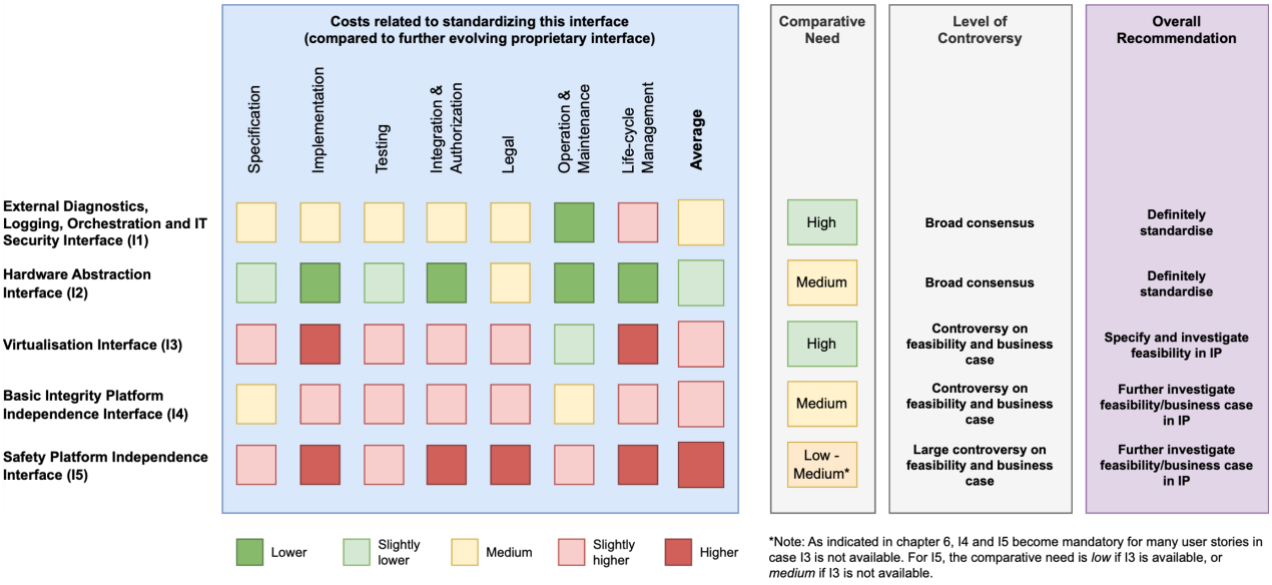


Figure 4: Recommendation on interfaces, taken from [14]

The recommendations shown in Figure 4 highlight the need of “feasibility investigations in IP” as now taken out by work package 26. Even if there are different levels of “comparative need” defined, work package 26 has taken a comprehensive view on all stated interfaces for the purpose of this deliverable. The business case investigation mentioned is not considered, see MPC-G10.

3.7.2 ERJU SP CE domain: Glossary

The goal of the Modular Platforms Glossary is to establish a common understanding of the terms used in this document and within the context of modular computing platforms in ERJU. This ongoing process of alignment and term definition is informed by our previous deliverables D26.1 [18], D26.2 [19], and the SP CE domain Glossary (as found in [15]). In cases where we were not aware of an existing definition, any additional content included here should be considered as input to the SP CE domain and other relevant areas.

We recognize the alignment and development of a comprehensive glossary as a crucial activity for the successful implementation of modular platforms.

The diagram presented below showcases the essential components of the domain-specific terminology employed in describing Modular Platforms. These terminologies adhere to the definitions outlined by the System Pillar Computing Environment Domain. To provide a comprehensive representation, the diagram incorporates two representations from the SP CE domain. It includes the interfaces between different layers and also highlights the connection to the central management system(s). By incorporating these elements, the diagram offers a holistic view of the key components and their relationships within the Modular Platforms framework.

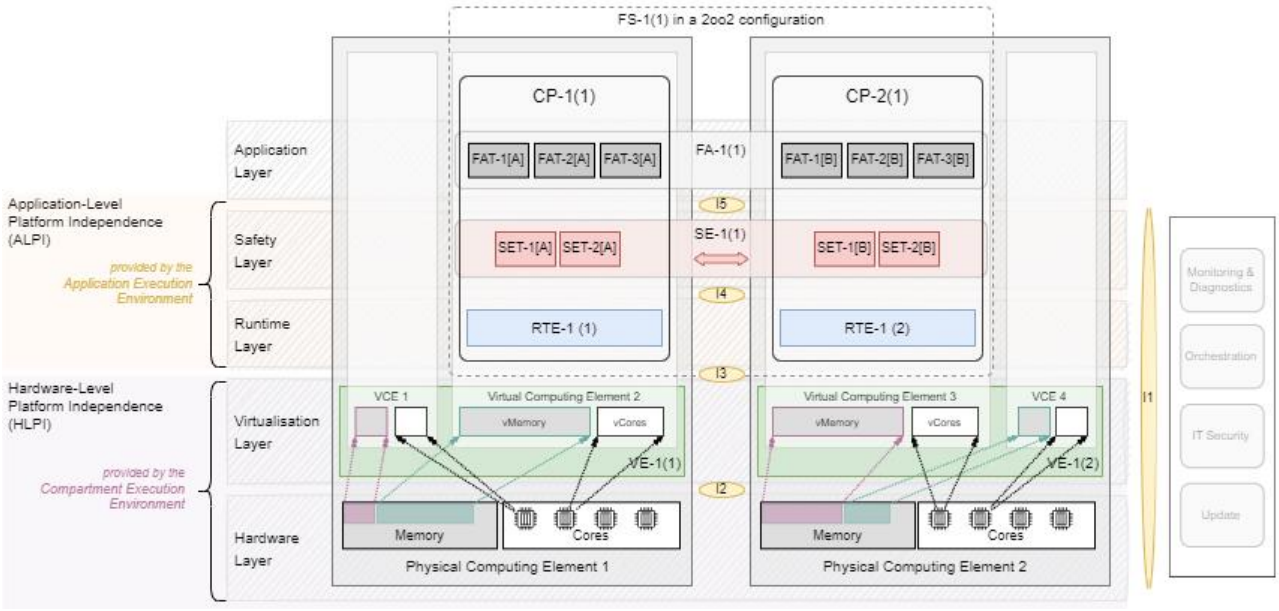


Figure 5: Terminology Landscape (aligned with SP CE domain)

Note: In a real-world scenario, it is important to consider the interfaces between different functional systems, commonly referred to as “I0”[4] interfaces. These interfaces facilitate communication and interaction between various systems. However, for the sake of simplicity, the diagram provided

[4] I0 is written as capital letter `I` and a zero `0`.

focuses on illustrating a single functional system and does not include depictions of interfaces between different functional systems. Furthermore, it does not show all variants for mixed criticality. The diagram above utilizes a specific notation to represent instances and replicas:

- The letters in the diagram represent the abbreviations for each corresponding entity. To understand the abbreviations used, please refer to the table provided below.
- Each abbreviation is followed by a hyphen and a number, which serves to differentiate between different kinds of the same entity.
- Instances of an entity are denoted by round brackets enclosing a number.
- Replicas of an entity, whether for safety or availability purposes, are denoted by square brackets enclosing a letter.

This notation enables clear identification and distinction of entities, instances, and replicas within the diagram.

Examples:

CP-1(1): The first instance of Compartment kind one

RTE-1(2): The second instance of Runtime Environment kind one

FAT-2[B]: The second replica of Functional Application Task kind two

The table below contains "*update notes*" to the definitions that were shared with the SP CE domain.

Term	Abbreviation	Definition
Application Execution Environment	AAE	The Application Execution Environment refers to the combination of Runtime Environment and Safety Environment. Update Note: The SE is optional if it's only a BIL application.
Application Layer	AL	The Application Layer contains Functional Applications that constitute Functional Systems.
Basic Integrity Platform Independence Interface	I4	The Basic Integrity Platform Independence Interface I4 (Interface 4) is used to perform a non-safety related platform independence with the applications. In other words, this API is an interface limited to non-safety functionalities between runtime environment and applications.
Compartment	CP	A Compartment is a consistent, integrated entity comprising exactly one Runtime Environment Instance, Safety Environment Task Replicas of at most one Safety Environment, and Functional Application Task Replicas of its respective Functional Applications. It can be deployed on either a Physical or a Virtual Computing Element.
Compartment Execution Environment	CEE	The Compartment Execution Environment refers to the combination of Physical Computing Element and Virtualization Environment.
Computing Element	CE	The Computing Element provides physical or virtual compute resources.

Term	Abbreviation	Definition
External Diagnostic, Logging, Orchestration, and IT Security Interface(s)	I1	The External Diagnostic, Configuration & Orchestration, and IT Security Interface I1 (Interface 1) comprises communication-based interfaces between rail systems and central infrastructure components (Shared Services) such as diagnostics, IT-security services, and remote update.
Functional Application	FA	A Functional Application is a comprehensive set of self-contained software functions, *assumed to be provided as one product by a single vendor*. Depending on its role in the overall function provided by the Functional System, it has a specific SIL (BIL up to SIL4) assigned (in-line with total FS SIL definition). Update Note: The technical definition of FA should not make assumptions on the sourcing (e.g., being a product of a vendor).
Functional Application Task	FAT	A Functional Application Task implements part of the functionality provided by a Functional Application. Depending on its role in the overall function provided by the Functional Application, it has a specific SIL assigned (in-line with total FA SIL definition). It may run replicated in multiple Compartments as FA Task Replicas.
Functional System	FS	A Functional System is a comprehensive set of self-contained Compartments, *assumed to be provided as one product by a single vendor*. Depending on its overall function, it has a specific SIL assigned. Update Note: The technical definition of FA should not make assumptions on the sourcing (e.g., being a product of a vendor).
FS Deployment Rules	FSDR	The FS Deployment Rules comprises all necessary information for deploying the respective Functional System onto specific approved Compartment Execution Environment(s). These deployment rules are compiled as part of the FS integration process and are part of each integrated, tested and certified/approved Functional System along with its FS Compartments and all necessary approval documentation.
Hardware Abstraction Interface	I2	The Hardware Abstraction Interface I2 (Interface 2) provides an abstraction of all technology layers above from the specific hardware used below, enabling easy replace ability of commercial of-the-shelf hardware procurable from a well-sized market of hardware vendors. Note: This is not really an interface, but rather a compatibility list of allowed hardware incl. CPU, memory, etc.
Hardware Layer	HL	The Hardware Layer contains the actual Physical Computing Elements providing the compute resources to the platform.
Instance	INS	An Instance is a specific realization of any entity. Update Note: “instantiation” could be used instead of “realization”.

Term	Abbreviation	Definition
Operational Interfaces	I0	The I0 is the sum of all operational interfaces used from Functional Systems (as e.g. an RBC) to communicate with other Functional Systems (as e.g. an IXL). Examples for these set of interfaces are the Eulynx Interfaces (SCI-xx) or interfaces like EuroRadio or TSI-standardized interfaces.
Orchestration Interface	OI	This interface is used to manage (monitor, control, diagnose, configure) the virtual computing environments. It only exists if a Virtualisation Interface is present. OI is part of I1. Update Note: The Orchestration Interface as described here is not part of I1 in the way that MPC and Shared Services are using the term.
Physical Computing Element	PCE	The Physical Computing Element refers to the physical device providing compute resources.
Replica	REP	A Replica is a specific realization of any entity in a cluster of peers used for composite fail safety and/or availability. Replicas of the same entity always run in distinct Compartments deployed to distinct Computing Elements. Update Note: "instantiation" could be used instead of "realization".
Runtime Environment	RTE	The Runtime Environment refers to the software needed to provide the services of the Runtime Layer in a single Compartment.
Runtime Layer	RL	The Runtime Layer refers to the system services (e.g., application and computing resource orchestration, monitoring of the Functional Applications and the Application Execution Environment, tracing and logging, communication services that are not related to safety, security means incl. authentication, encryption, key storage, etc.) and the communication stack for information exchange between Functional Applications running on the same Computing Environment and with external entities. It may also include an operating system.
Safety Environment	SE	The Safety Environment refers to all Safety Environment Tasks needed for a Functional System.
Safety Environment Task	SET	A Safety Environment Task implements part of the functionality provided by a Safety Environment. Depending on its role in the overall function provided, it has a specific SIL assigned (in-line with total SE SIL definition). It may run replicated in multiple Compartments as SE Task Replicas.
Safety Layer	SL	The Safety Layer implements all the technical safety principles related to fulfilling the requirements of EN 50126, EN 50716, EN 50129, EN 50159 (e.g., composite fail safety, fault tolerance, voting mechanisms, redundancy mechanisms for availability, safety communication layers etc.) that are needed to enable the execution of Functional Applications up to SIL4.

Term	Abbreviation	Definition
Safety Platform Independence Interface	I5	The aim of introducing Safety Platform Independence Interface I5 (Interface 5), is to be able to implement platform independent Safe Functional Applications (up to SIL4) i.e., applications, based on a generalized abstraction between the application logic and the system interfaces, will run unchanged on different platform implementations.
Virtual Computing Element	VCE	The Virtual Computing Element refers to virtually provided compute resources with computing resource guarantees.
Virtualisation Environment	VE	The Virtualisation Environment contains all software needed to provide (multiple) Virtual Computing Elements on a single Physical Computing Element.
Virtualisation Interface	I3	The Virtualization Interface I3 (Interface 3) is used to provide a standardized interface above the virtualisation layer so that applications or higher platform layers are independent of a specific implementation of the computing hardware.
Virtualisation Layer	VL	The Virtualisation Layer contains mechanisms that can provide Virtual Computing Elements needed to run multiple Compartments on a single physical hardware underneath. Update Note: FSDR for Compartment allocation to physical hardware have to be taken into account.
Virtual Machine Management	VMM	Virtual Machine Management refers to the software and processes used to create, monitor, and manage virtual machines.

Table 2: Modular Platforms Terms

The entity relationship diagrams depicted in Figure 6 illustrate the relationships between various entities, along with their respective multiplicities. The diagrams are divided into three sub-diagrams, each focusing on a specific entity: Modular Platform Instance, Compartment Instance, and Functional System Instance.

These sub-diagrams provide a detailed view of the relationships and associations of each entity within the context of the overall system. By examining these diagrams, one can gain insights into how the instances of Modular Platforms, Compartments, and Functional Systems are interconnected and interact with one another.

The entity relationship diagrams serve as a valuable tool for understanding the structural composition and dependencies within the system architecture.

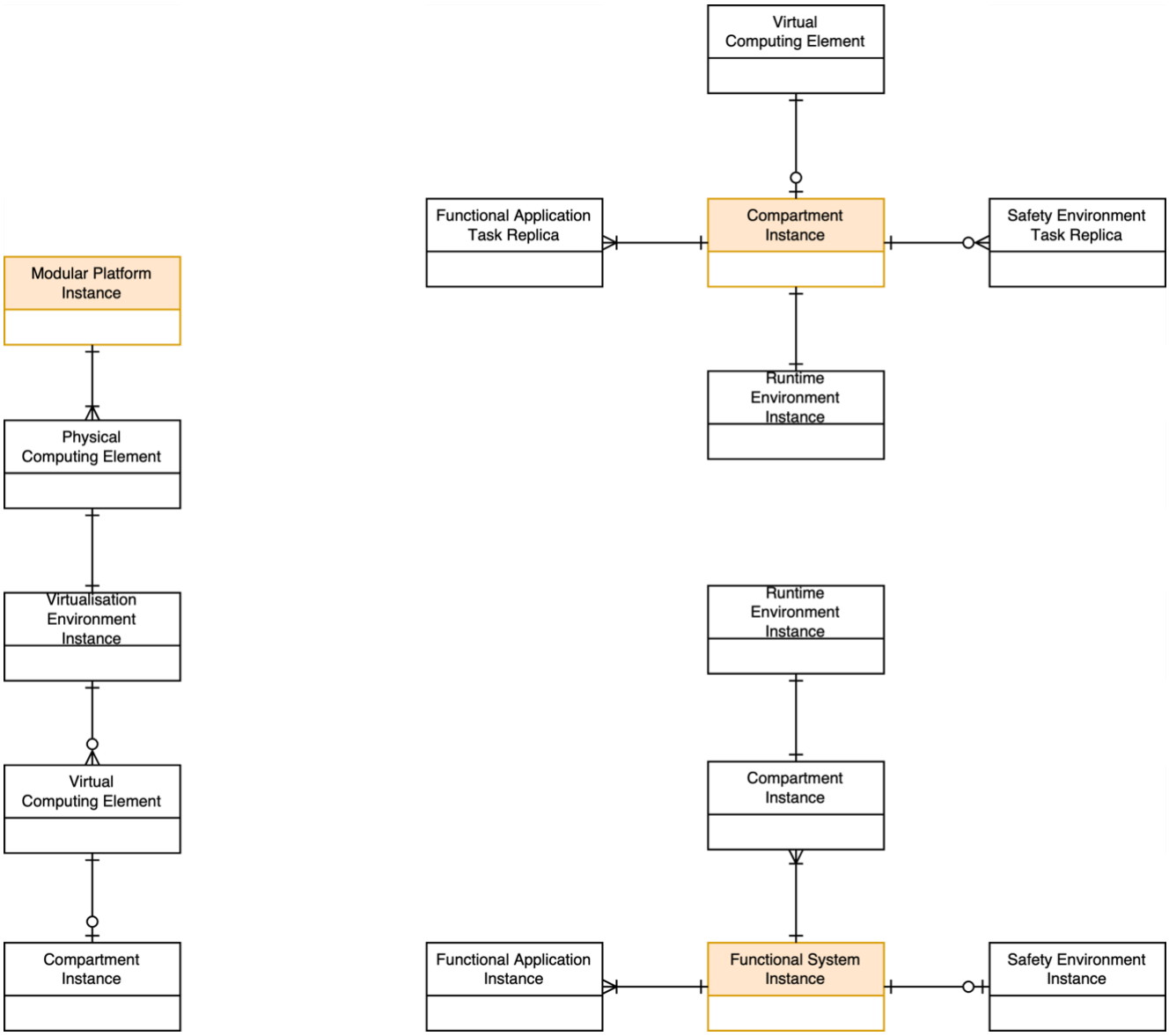


Figure 6: Entity relationship diagrams

3.7.3 ERJU SP CE domain: OAS

The second deliverable of the ERJU SP CE domain – called “Operational Analysis Specification” (OAS, as of June 2024) [15] – contains an updated terminology (that was considered for the aligned glossary of chapter 3.7.2) and focuses on discussing operational aspects in the context of I1, I2 and I3 (see chapter 3.7.1). Here, operational scenarios, an operational context and a first collection of operational requirements is developed for the computing environment (see chapter 3.10).

The operational context of the OAS shows exemplary deployment scenarios for Functional Systems in different configurations and with differing safety goals. Furthermore, a Functional System is defined to consist of the following:

- all Functional System Compartments
- Functional System Deployment Rules
- all necessary Approval Documentation

The operational scenarios discussed in the OAS are categorized into 4 categories:

- Integration: The act and pre-conditions of bringing a new item into the operational system.
- Deployment: Actual manipulations on computing elements (physical or virtual)
- Update: Scenarios for changing of different in-place elements
- Recovery: Reactions to different failure scenarios.

The operational scenarios are used later in this document to derive requirements for the internal interfaces of the modular computing platform, see chapter 8.

Relevant Actors and Entities are introduced in the OAS as well.

The OAS also derives a first set of requirements bases on the operational context and scenarios given. They are categorized into three categories: Hardware, Safety and Availability and Virtualization. A subset of the requirements is captured in this document in Appendix A.

3.7.4 ERJU SP TCCS domain

The ERJU SP Transversal CCS (TCCS) domain is working on overarching topics. For the MPC, their work on diagnosis, configuration and update is relevant. Nevertheless, work package 26 expects the general and detailed alignments to happen between the different SP domains (CE and TCCS).

3.7.5 ERJU SP PRAMS domain

The alignment with the PRAMS domain is intentionally still very limited for work package 26 task 2, which is the context for this deliverable. In the follow-up task 3, the alignment will be intensified.

3.7.6 ERJU SP Cyber Security domain

The alignment with the Cyber Security domain is as well limited but relevant content has been tried to be integrated into separate chapters of this deliverable as 3.8.2, 6.8 and 7.4.9.

3.8 PRAMSS

The abbreviation PRAMSS is an extension of “RAMS” (as defined by EN 50126-1), adding parameters of systems crucial in a modern railway environment.

- P: Performance (new)
- R: Reliability
- A: Availability
- M: Maintainability
- S: Safety
- S: Security (new)

The following subchapter give a brief discussion on how these parameters are of relevance for the MPC.

3.8.1 Safety

For a modular computing platform that targets applications like interlockings, functional safety is a property that influences almost every aspect of the concept presented here. Even if the MPC is used in scenarios where only Basic Integrity software is necessary, there are still requirements to be fulfilled from a regulatory standpoint, especially if modularity concepts are used.

Therefore, the MPC will enable most of its advantages if it's embedded into a modular safety certification approach. As per the MPC purpose MPC-P03 and assumption MPC-A05, all safety integrity levels ranging from Basic Integrity up to SIL 4 are to be supported, with the expectation that Functional Systems almost always will contain elements with different SIL requirements.

While the concept presented here takes into account all needs that arise from a potential standards-compliant implementation of the MPC, the scope of Task 2 of work package 26 – which is the context for this deliverable – is not including a full safety analysis. A study on modular platform certification is the topic for the work package's Task 3, which will document its findings in deliverable D26.4. The expectation there is to align with the work of the SP PRAMSS domain, see chapter 3.7.5.

In the meantime, insights into how to approach modular safety for state-of-the-art systems can, for example, be found in the "SIL4 Cloud" [3] and "SIL4 Data Center" [4] research reports.

For a layered architecture with FS compartments of different safety levels running aggregated on a virtualization environment, it's essential to define the safety architecture and to identify the safety related requirements which arise by SW based SE running on COTS hardware, for this see chapter 6.7.

3.8.2 Security

For modular platforms, appropriate cybersecurity approaches need to be identified that match their needs. In general, all relevant interfaces and layers of the modular platforms architecture have a need for special cybersecurity requirements.

Within the System Pillar the cyber security domain is responsible for the high level security architecture and requirements which includes and are applicable to the modular platform.

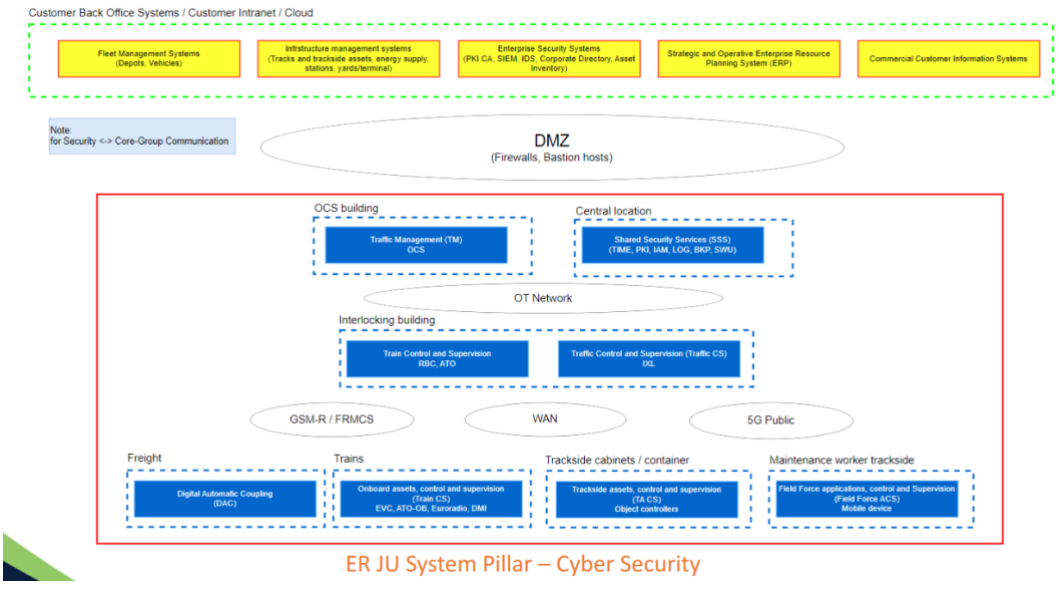


Figure 7: Scope of ERJU System Pillar - Cyber Security domain

The security domain is currently working on 3 documents which include requirements for the modular platform. The final versions are going to be available at the end of the year 2024.

- 30_Secure Component Specification v.0.85
- 40_Secure Communication Specification v.0.85
- 50_Shared Security Services Specification v0.85

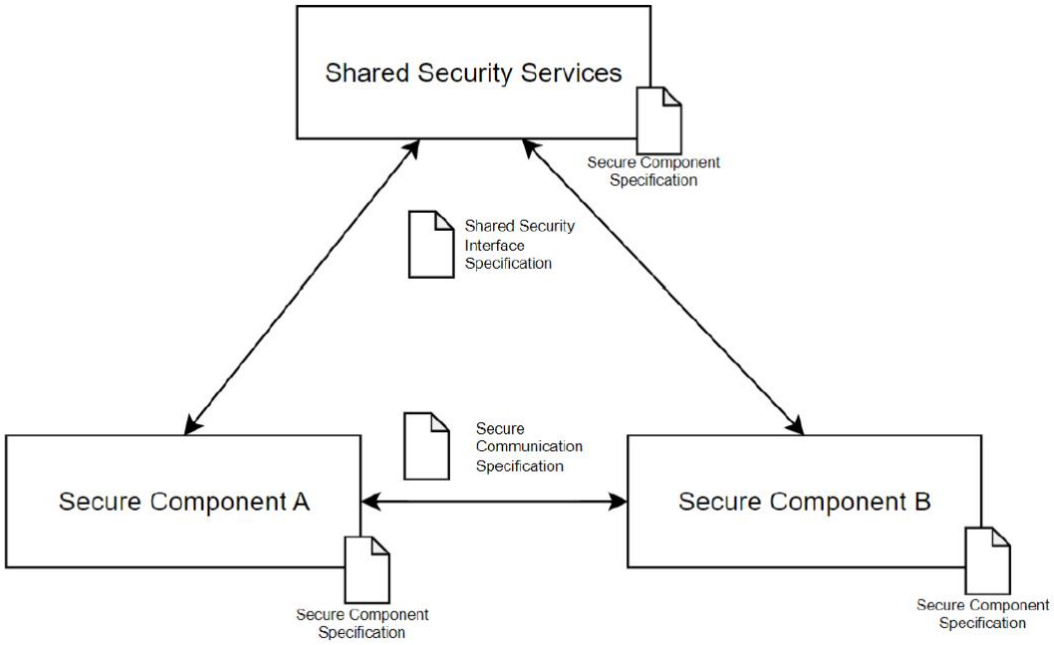


Figure 8: Key terms and technical specs used in the System Pillar Cyber Security domain

3.8.2.1 Secure Component Specification

The secure component specification is based on previous work done in EULYNX BL 4 R2 and ESCG and is based on the following standards.

- IEC 62443-4-2 Ed 1
- ESCG Requirements
- UNISIG Subset 146 v4.00
- UNISIG Subset 147 v4.00
- CEN TS 50701/IEC PT 63452

It describes the cyber security requirements for secure components like the modular platform which include for example hardening, encryption, certificate usage and making use of the shared security services and secure communication.

3.8.2.2 Secure Communication Specification

The secure communication specification is based on previous work done in EULYNX BL 4 R2 and ESCG and is based on the following standards.

- UNISIG Subset 146 v4.00
- RFC8446 The Transport Layer Security (TLS) Protocol Version 1.3

It describes the requirements for the secure communication which are needed to communicate between secure components. In case of the modular platform this will include communication between functional system compartments and from functional system compartments to external systems and shared services which require secure communication.

3.8.2.3 Shared Security Services

The shared security services specification is based on previous work done in EULYNX BL 4 R2, ESCG and UNISIG Subset 146 v4.00.

The specification defines the interfaces to the Shared Security Services which the modular platform will use and are required for interoperability in and harmonization of the European rail automation domain.

The Shared Security Services include the following services which can be used by applications on-board a trainset or by trackside applications:

- **STS** - Secure Time Synchronisation - service for secure time synchronisation to Secure Components
- **PKI** - Public Key Infrastructure - service for distributing certificates and their status to Secure Components, crucial for all secure communication
- **IAM** - Identity and Access Management - service for managing digital identities (human users and assets)
- **NAC** - Network Access Control - service for identifying, authenticating, and authorizing network access of Secure Components
- **LOG** - Security Logging - service for collecting log messages from Secure Components and relaying log messages (e.g. to another relay or SIEM)
- **SSO** - Single Sign On - service for managing roles for authorisation and single-sign on (SSO)

  Comment: SSI-SSO is not needed when there is no human userlog in (e.g. on embedded devices)
- **BKP** - Backup and Restore - service for creating and restoring backups to/from Secure Components

  Comment: SSI-BKP is not needed when there is nothing to backup (e.g. on devices without state with fixed configuration)
- **DNS** - Domain Name System - service for name resolution to map domain names to IP addresses

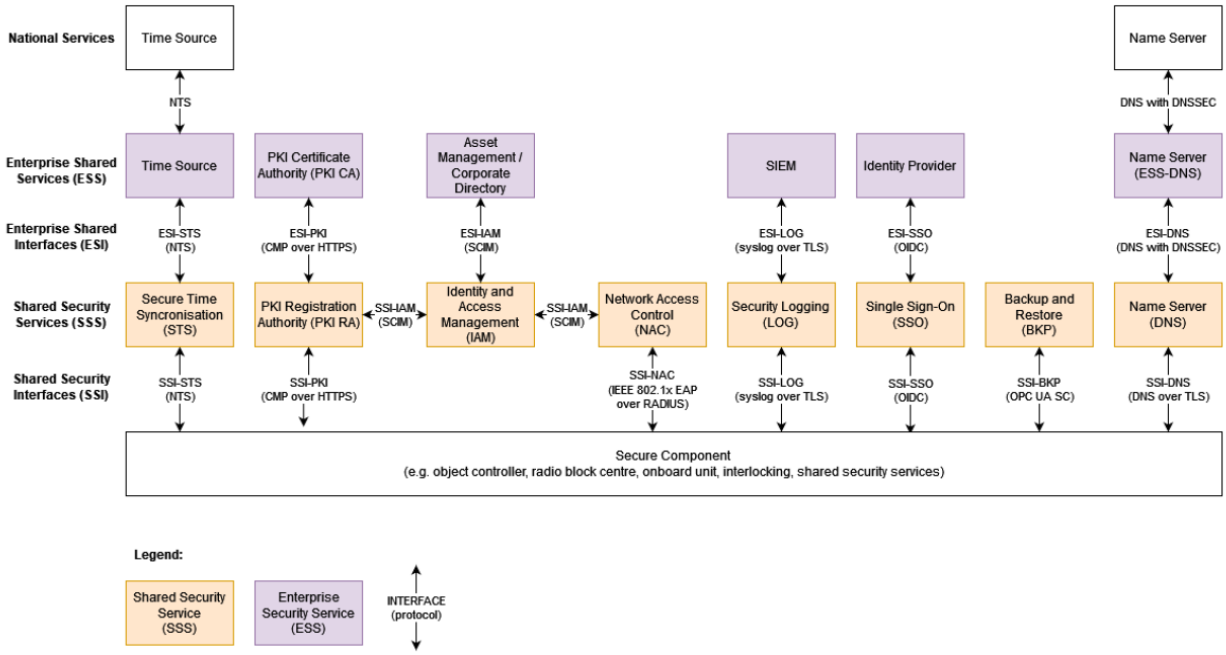


Figure 9: Hierarchy and interfaces of shared services

The figure above includes:

- the Shared Security Services which offer Shared Security Interfaces (SSI) to Secure Components
- the Enterprise Security Services (ESS), which offer Enterprise Security Interfaces (ESI) to SSS
- external services on a national level used by the ESS

Nevertheless, the cybersecurity aspects of the modular platforms need to be aligned with the appropriate SP domains, e.g., PRAMS, and they need to be analysed from the perspective of both, on-board and trackside, deployment options. This work is ongoing.

For a layered architecture with FS compartments of different safety levels running aggregated on a virtualization environment, it's essential to define the security architecture and identify the security related requirements for the individual SW layers, for this see chapter 6.8.

3.8.3 PRAM

The remaining letters of the PRAMSS acronym are briefly brought into context of the MPC in the following sections.

Performance (P)

The MPC will need to assure performance guarantees to the functions (implemented in the form of Functional Systems and their components). This is handled via a description of what level of performance is necessary (e.g., computing resources resp. time) in a defined format. This description can potentially be a part of the FS deployment rules (FSDR).

Reliability (R)

The MPC will allow assessment of the reliability properties and influences, determining safety and availability.

Availability (A)

The flexibility to reach necessary availability goals for a given function is part of the MPC, e.g., by allowing different redundancy configurations.

For a layered architecture with FS compartments of different safety levels running aggregated on a virtualization environment, it's essential to identify and define the needed requirements from view of availability, for this see chapter 6.9.

Maintainability (M)

To reach expected maintainability goals, the MPC includes fault detection and identification services like diagnostics and logging and offers clearly defined interfaces to allow maintenance and restoration of a failed system (see also chapter 6.11).

3.9 User Stories

As mentioned in chapter 3.7.1, the first SP CE domain deliverable (RIS, [14]) introduced thirteen user stories. They are derived from the SP "Common Business Objectives" [16]. The user stories with their advanced expectation towards a modular railway computing platform build the basis for the work of the SP CE domain and also the MPC. Several important user stories are given here, for all user stories and their details please refer to the RIS [14].

SPT2CE-18 – Minimize overall dependencies

As a supplier, rail infrastructure manager or railway undertaking, I would like to minimize dependencies among Functional Application, Runtime Environment and Hardware, in order to minimize obsolescence related risks and costs.

SPT2CE-19 – Aggregate multiple Functional Applications on the same Instance of a Computing Platform

[…]

SPT2CE-20 – Remotely add, modify, delete or configure functions

As a supplier, rail infrastructure manager or railway undertaking, I would like to be able to remotely add, modify, delete or configure functions via a harmonized approach (without or with minimal effort and lean process for new authorization), in order to reduce operational expenses, time to deployment, Functional Application Downtime and Service Unavailability.

SPT2CE-23 – Computing Platform suitable for Functional Applications up to SIL4

As a rail infrastructure manager or railway undertaking, I need a Computing Platform (Runtime Environment and Hardware) suitable for Functional Applications up to SIL4, in order to be able to integrate railway-in-house or third party developed Functional Applications.

SPT2CE-25 – Replace one Hardware by another with minimal or no re-authorisation effort

[…]

SPT2CE-28 – Interface a Computing Platform with existing systems

As a rail infrastructure manager or railway undertaking, I would like to interface a Computing Platform with existing systems through a communication network and/or discrete hardwired connections in order to minimize the acquisition and integration cost.

SPT2CE-30 – System operation and update deployment without or with minimal on-site presence

[…]

3.10 Operational Context and Operational Scenarios

The SP CE domain "OAS" deliverable [15], as introduced in chapter 3.7.3, discusses the operational context and several operational scenarios based on the terminology given in chapter 3.7.2 and the basic ideas from their first deliverable, "RIS" [14], as introduced in chapter 3.7.1.

A general assumption for both, operational context and scenarios, is, that the activities are focused on Functional Systems. This is a result of the SP CE domain's "main objective of standardising the Computing Environment […] to enable the operation of Functional Systems from various suppliers on a shared pool of physical computing resources" [15]. Depending on the needs of the FS, as documented in its FS deployment rules (FSDR), different scenarios are given as examples for their configuration. These needs depend on requirements towards availability, redundancy, and additional safety related measures.

1. FS (#1) implementing a 2-out-of-3 redundancy configuration using three compartments. The compartments each need to be run on a distinct piece of physical hardware.
2. FS (#2) implementing the same configuration as above, but also including a fourth compartment with Basic Integrity software only, that has no further need for redudancy and could be deployed on any suitable physical hardware.
3. FS (#3) implementing a 1-out-of-2 redundancy configuration using two compartments with Basic Integrity software. The redundancy goal can only be achieved when the compartments are run on distinct piece of physical hardware.

For FS #1 and #2, an application specific communications interface I0 is used between the two. As this I0 interface is highly specific, it's out of scope for the MPC.

The operational scenarios discussed in the OAS focus on the FS resp. compartment level. As such, the learnings later derived for the MPC are mostly relevant for the internal handling of the compartments (hardware independence approach, see chapter 5.4) and the interface exposed to facilitate the necessary operations and status collection (external interfaces, see chapter 7.4.6.1). The overview of the scenarios as discussed in the OAS is given in the following list. For details, please refer to [15].

Integration Scenarios

SPT2CE-1411	Integration of Functional System FS2 beside Functional System FS1 on already existing Computing Element
SPT2CE-1406	Integration of Functional System FS2 with Functional System FS2, interacting with each other
SPT2CE-1405	Integration of Virtualisation Environment on a new version/type of a physical Computing Element

Deployment Scenarios

SPT2CE-1420	Prepare Physical Computing Element(s)
SPT2CE-1421	Install Virtualisation Environment on Physical Computing Element(s)
SPT2CE-1428	Configure Virtual Computing Elements required for first Functional System
SPT2CE-1431	Deploy Functional System Compartments on Virtual Computing Elements
SPT2CE-1439	Uninstall Functional System deployed on Virtual Computing Element(s)

Update Scenarios

SPT2CE-1448	Replace physical computing element
SPT2CE-1446	Update Virtualization Environment while Functional System is Running (Compatible Update)
SPT2CE-1456	Update Functional System while it is Running (Compatible Update)
SPT2CE-1458	Update Functional System including Stopping of FS (Incompatible Update)

Recovery Scenarios

SPT2CE-1483	Total SW Failure of one FS Compartment
SPT2CE-1499	Failure of all external communication channels regarding I0
SPT2CE-1485	Total SW Failure of all FS Compartments
SPT2CE-1482	Individual SW failure of one virtual computing element
SPT2CE-1489	SW Failure of one complete VE Instance
SPT2CE-1487	SW Failure of all VE Instances
SPT2CE-1496	Individual HW failure within one physical Computing Element
SPT2CE-1490	Total HW failure of one complete physical computing element.
SPT2CE-1492	Disaster scenario - failure of all computing elements
SPT2CE-1501	Failure of one external communication channel regarding I0

3.11 Intended Usage Scenarios

For a discussion around usage scenarios for the MPC, two cases must be differentiated: The usage of the Modular Platform Concept (as described in this deliverable) and the usage of actual platform implementations compatible to the MPC (CPI – compatible platform implementation).

MPC Usage Scenarios

The MPC can be used to implement platforms that are compatible to the principles as outlined in this deliverable. As the deliverable itself is not a directly implementable specification, additional subsequent steps are necessary for full standardization of the relevant interfaces to unlock the potential benefits described in the previous chapters.

Nevertheless, to research the feasibility of the MPC, the information and design proposals from this deliverable can be used to guide future work. Especially for the highly recommended – as per RIS [14] – interfaces for the operation of compartments (I2, I3) and the integration of CPI into a bigger IT/OT landscape (I1), prototypes can be built on this deliverable, enhancing the specification.

The MPC itself can also be used as a basis for the definition of prototypes and demonstrators, such as the work package 36 “On-board Platform Demonstrator”.

CPI Usage Scenarios Examples

As the MPC is one unified concept, various implementations of the MPC (referred to as CPI, see above) are possible and are expected to be tailored to their use case and environment. Example types of CPI are given here, without the list being exhaustive:

- Platforms for small, on-premises trackside data centres that are used to operate interlockings

- Platforms for medium sized, centralized trackside data centres that are used to operate multiple interlockings
- Platforms for larger, centralized trackside data centres that are used to operate multiple interlockings from diverse areas, potentially also offering georedundacy for other data centres
- Platforms for consolidated on-board functionality, especially in CCS areas were frequent updates are to be expected or functionality will be added later in the lifecycle

There are of course smaller scenarios where the MPC in general might not be suitable, e.g., for single object controllers and similar single-function, potentially embedded systems. Here, CPIs are not expected to be available.

3.12 Platform Environment Examples

For the MPC, two platform environment examples are shown in this chapter. These examples are included to try to complement all the information regarding MPC and show a potential mapping of abstract terms like FS and FA. The first one is about wayside and the second one is about on-board. These examples do not include an exhaustive list, neither for the functional systems nor for functional applications, and are not trying to define MPC usage. Both platform environment examples are proposed following the Figure 5 introduced in chapter 3.7.2 (ERJU SP CE domain: Glossary).

The principal concepts concerning the Glossary are Functional Systems (FS) and Functional Application (FA). In the following figures, they are represented as green (FS) and in blue (FA).

Concerning the wayside environment and subsequently to Figure 10, RBC and Interlocking systems are used as examples.

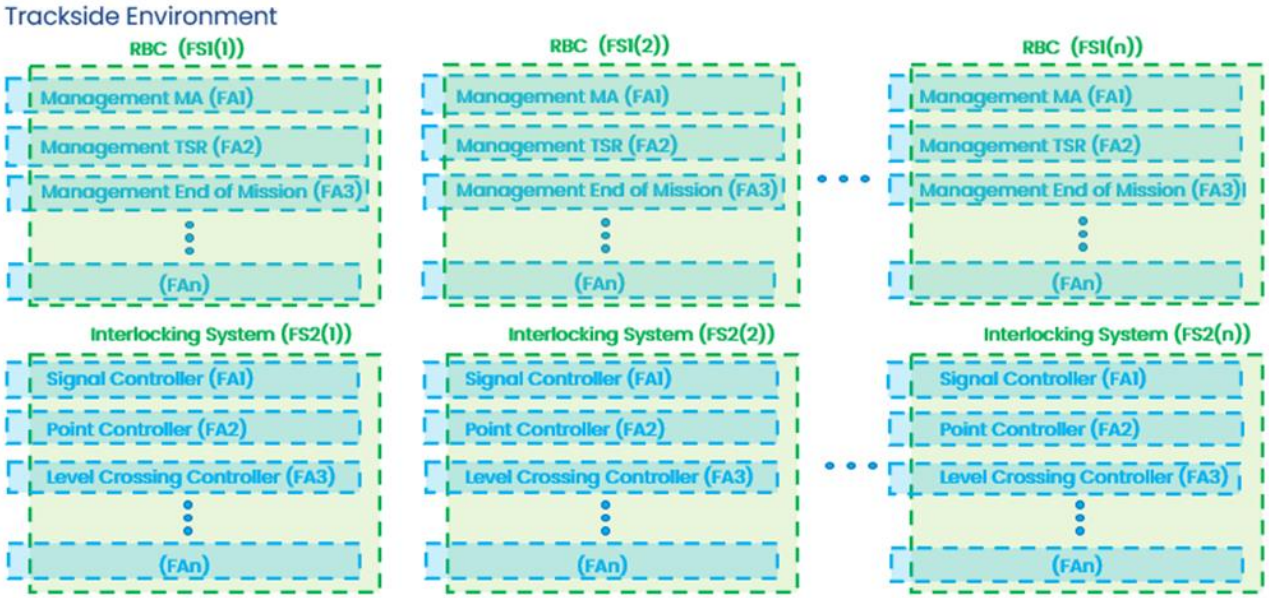


Figure 10: Trackside environment for MPC

The first functional system proposed on a trackside environment is the RBC. In this case it can be distinguished different functional applications such as a management of a MA, a management of TSR or a management of an End of Mission among other. These three functional applications are some examples proposed for this concrete functional system, but there are not the only ones, and it can be more functional applications for a functional system.

Regarding RBC concept, it could be several RBC working as functional systems on different instances just because on a same train route there are different RBC to connect.

As there is a functional system for the RBC it will also another one for the Interlocking System. The functional applications proposed for this functional system are the signal controller, the point controller and level crossing controller among others.

Concerning on-board environment and subsequently to Figure 11, it is proposed another four concepts working as functional systems, ETCS, FRMCS, ATO and National System.

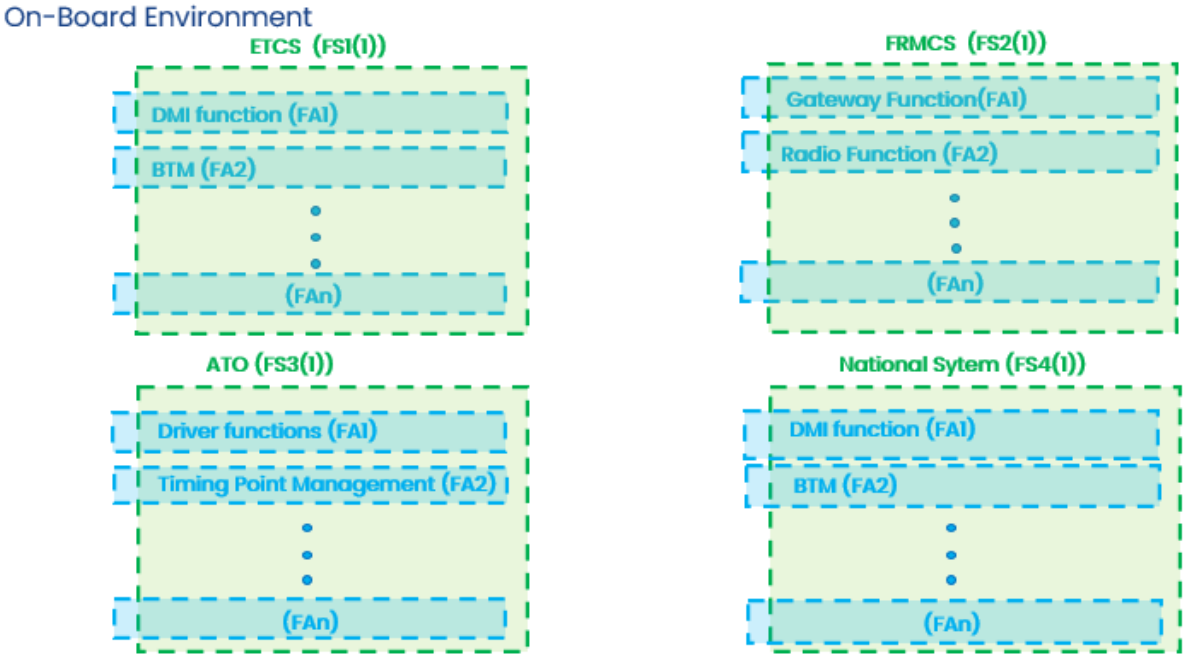


Figure 11: On-board environment for MPC

The first functional system proposed to run in an an on-board environment is the ETCS concept.

According to ETCS concept understood as a functional system, DMI function and BTM are some platforms environment examples for functional application.

The second functional system proposed for the on-board environment is FRMCS. Pursuant to FRMCS functional system, some examples of functional applications can be found. These functional applications are the gateway function and the radio function.

For the on-board domain, another functional system proposed is the ATO concept. Driver functions and timing point management are the examples proposed as functional applications.

The last functional system proposed is the ATP national System. The functional application for this functional system depends on the national system itself, so, as it was for the ETCS concept, the BTM and DMI function shall be examples of functional application for this functional system.

4 MODULAR PLATFORMS REQUIREMENTS

Based on the aspects of the Modular Platform Concept as described and discussed in the previous chapter, the next step is to create a set of relevant requirements. These requirements are the basis for architectural work in the next chapter.

Requirements towards railway-suitable (modular) computing platforms have been defined in the past, as discussed in this work package's first deliverable [18]. Requirements defined in the past usually have a distinction between trackside and on-board systems, depending on their context. For providing a more generic set of requirements, this work package aims to remove this distinction where feasible, potentially only containing some optional and specialized requirements. The following briefly discusses the potential sources for this work.

Project	Discussion
RCA/OCORA	The RCA/OCORA initiative is a comprehensive source for the on-board computing platform requirements. In particular the document OCORA TWS03-020-"Computing Platform Requirements" v. 4.1 [6], part of the OCORA Release 4 (and later), notably all the "approved" requirements MSC-XX, with XX from 01 to 127 (including the optional ones), are suitable for the purpose of MPC.
EULYNX	While EULYNX is looking at several aspects of distributed computing systems, its goal is not to define modular computing platforms. As such, EULYNX is only a source for indirect stakeholder requirements.
SIL 4 Data Center & SIL 4 Cloud Reports	The "SIL4 Data Center" report [4] and the "SIL 4 Cloud" [3] list several requirements in textual form.
ERJU SP CE domain	The already discussed second deliverable of the ERJU SP CE domain (OAS, see chapter 3.7.3) provides several requirements based on the operational scenarios discussed.

Table 3: Previous work related to MPC requirements

The goal of this deliverable is to present selected requirements for the MPC, ideally with little to no differentiation needs for on-board and trackside use. The explanation of the methodology, the detailed list of sources, and the selected respective adapted requirements are found in Appendix A.

5 MODULAR PLATFORMS ARCHITECTURE

The Modular Platform is a computing environment for the execution of mixed-critically workloads, offering the central benefits of allowing portability, flexibility and re-use of business logic captured as application software, the so called "Functional Applications" (for this and other terms' definition, please refer to the glossary in chapter 3.7.2). To enable these benefits, following the previous work and SP CE domain inputs (see chapter 3.7), three distinct domains for the Modular Platforms Concept architecture were derived:

- Application-Level Platform Independence (ALPI)
- Hardware-Level Platform Independence (HLPI)
- Interfaces external to the Platform

Here, the notion of "platform independence" refers to the independence of an actual implementation respectively instantiation of the Modular Platform concept (CPI, see chapter 3.11). The relation between the domains is shown in the following diagram.

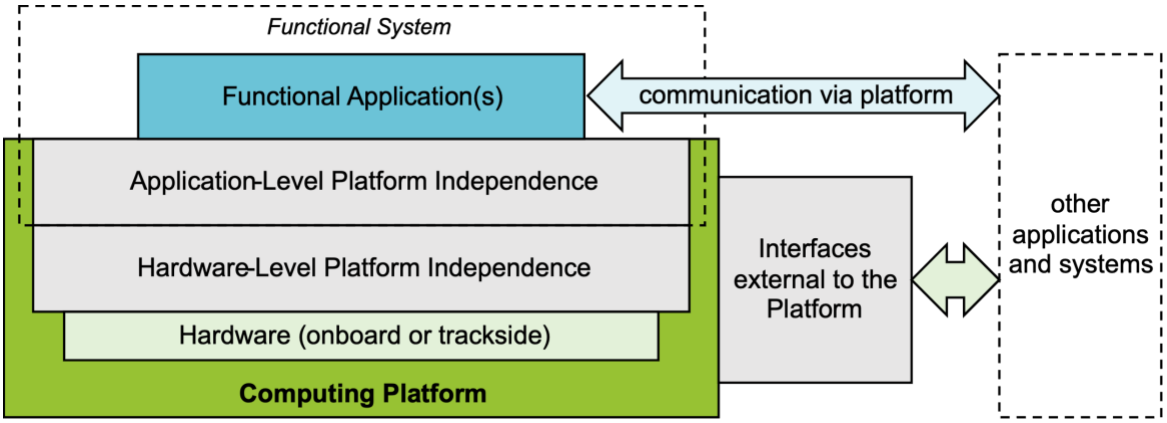


Figure 12: The three Modular Platforms domains embedded into the overall architecture

How these three domains fulfil the goals of the Modular Platform is explained in detail in the three dedicated subsequent main chapters. In this overview chapter, the next figure will show how the Modular Platform domains implement the SP CE domain interface recommendations and enable the known operational scenarios, using an enriched version of the figure above. Here, the division of SP CE domain interfaces to our ALPI, HLPI and external interfaces categories is introduced.

MPC domain	SP CE Interfaces	Details
Interfaces external to the platform	• I1: External Diagnostics, Configuration & Control Interface	Chapter 8
Hardware-Level Application Independence (HLPI)	• I2: Hardware Abstraction Interface • I3: Virtualisation Interface	Chapter 6
Application-Level Platform Independence (ALPI)	• I4: Basic Integrity Platform Independence Interface • I5: Safe Platform Independence Interface	Chapter 7

Table 4: MPC domain mapping to SP CE interfaces

The mapping given in Table 4 is graphically represented in the figure below.

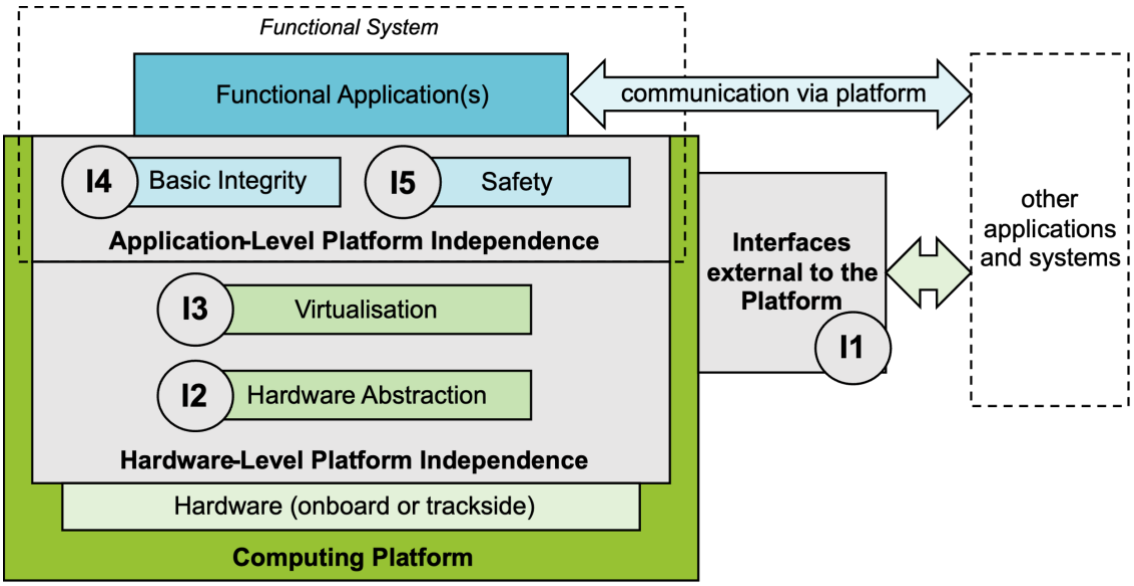


Figure 13: SP CE domain interfaces mapped to the Modular Platform domains

As shown, the interface I1 is encompassing Modular Platform related and relevant interfaces which are external to the platform, for example for update and configuration purposes. Interface I1 is not used for Functional Application communication, e.g., for interfacing to object controllers in an interlocking application. For this business logic relevant communication, the SP CE domain introduced a generic I0 interface [15], which is out of scope for the MPC.

Interfaces I2 and I3 are providing means for hardware abstraction and, if needed, aggregation respectively integration of multiple runtime environments (inside of compartments) running on the same hardware.

Interfaces I4 and I5 are used by a Functional Application to implement its business logic in a platform independent manner. A Functional Application may contain both, safe and non-safe functionality, using I4 and I5, respectively.

Between the SP CE domain interfaces (I1 and I2…I5), common parts are needed for the implementation of an actual computing platform, as shown in Figure 14 below.

These common parts, also called “layers” by the SP CE domain, will be discussed later in the document, and are at this stage only used to illustrate potential usage scenarios for the interfaces I1 to I5. Also, for the functionality shown within I1, the naming from the SP CE domain has not been used here, as there are some differences proposed in chapter 8.

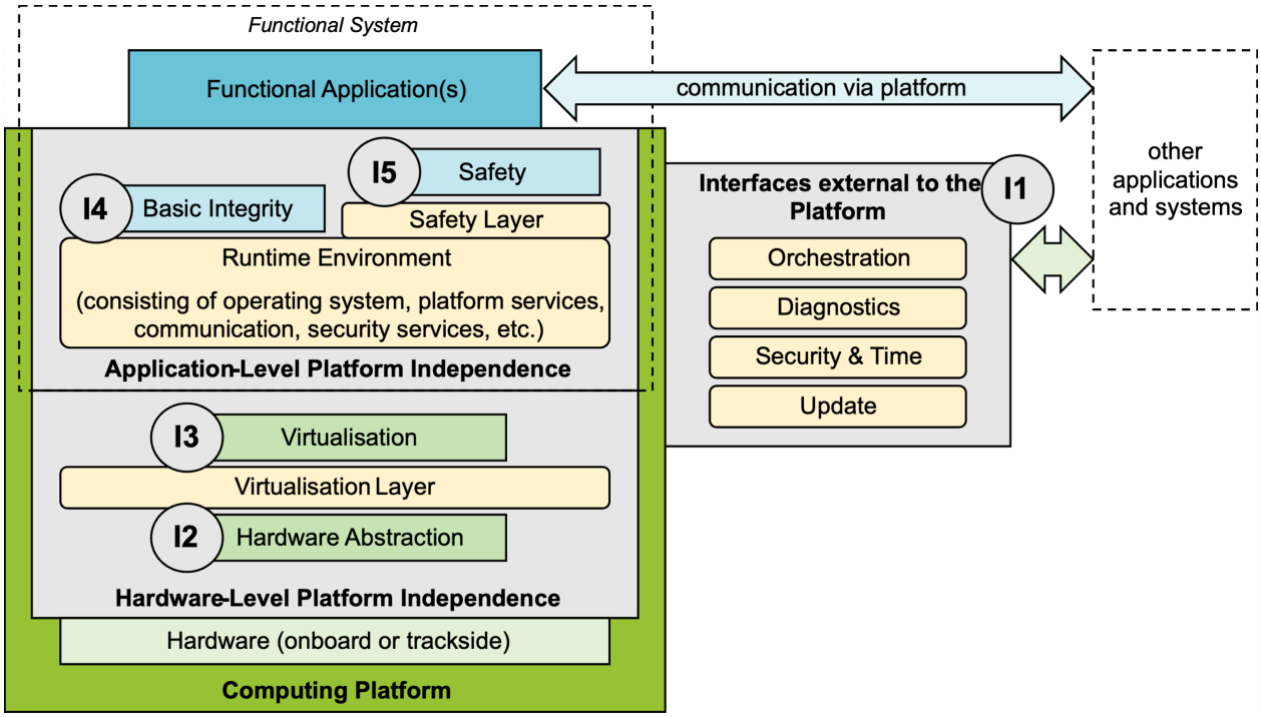


Figure 14: Modular Platform architecture showing key components and interfaces

5.1 MODULARIZATION ARCHITECTURE

The MPC is a truly modular environment when it's agnostic towards multiplicities and tasks of the physical (e.g., hardware) and logical components (e.g., software executing a business logic). Taking the approach from the previous chapter – the distinction of three MPC domains – and focusing on the two providing independence methodologies (HLPI and ALPI), the modular setup of physical and logical components can be introduced using simple examples.

First, in the context of HLPI (Hardware-level Platform Independence), the flexible deployment of Functional Systems onto Physical Computing Elements being part of a Compartment Execution Environment is shown.

Afterwards, the construction of CPs using the ALPI approach is discussed.

This is done using the terms and naming conventions established in chapter 3.7.2.

5.1.1 HLPI Modularization Architecture

Functional Systems are, on a high level, a collection of Compartments that together implement a certain business logic. They can be seen as a deliverable by a supplier that an end-user wants to deploy onto its CPI consisting of one or more Compartment Execution Environments (CEE).

HLPI provides the ability to execute a certain number of Compartments (CP) on a single Physical Computing Element (PCE). Several PCEs can be part of a single CEE. Compartments are belonging to a Functional System (FS) and only are of value when they are deployed as described in its individual FS Deployment Rules (FSDR), in accordance with their approval documents. The generalized view is shown in Figure 15 below.

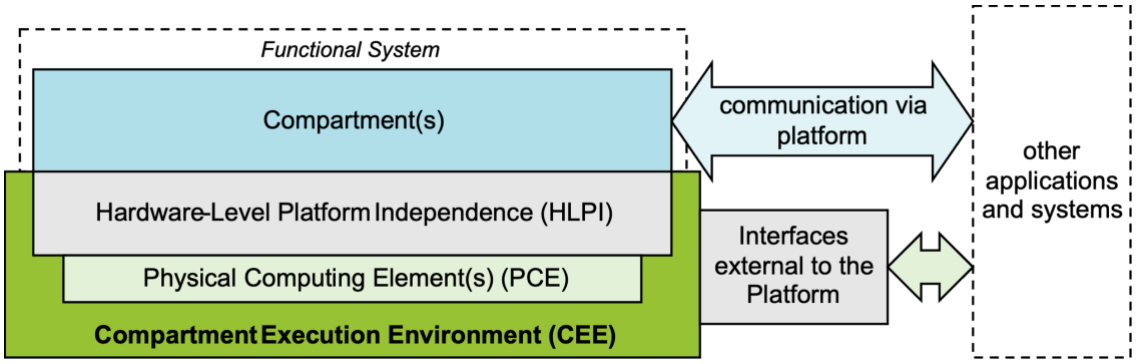


Figure 15: Modularization enabled by FS Compartments on HLPI

For an exemplary FS that employs a 2oo2 (2 out of 2) configuration, the FSDR states that two Compartments need to be deployed on separate PCEs. How this can look like after deployment is show in Figure 16 below. Here, the CEE only hosts the exemplary FS. The Compartments are supported by the HLPI mechanisms present on each PCE.

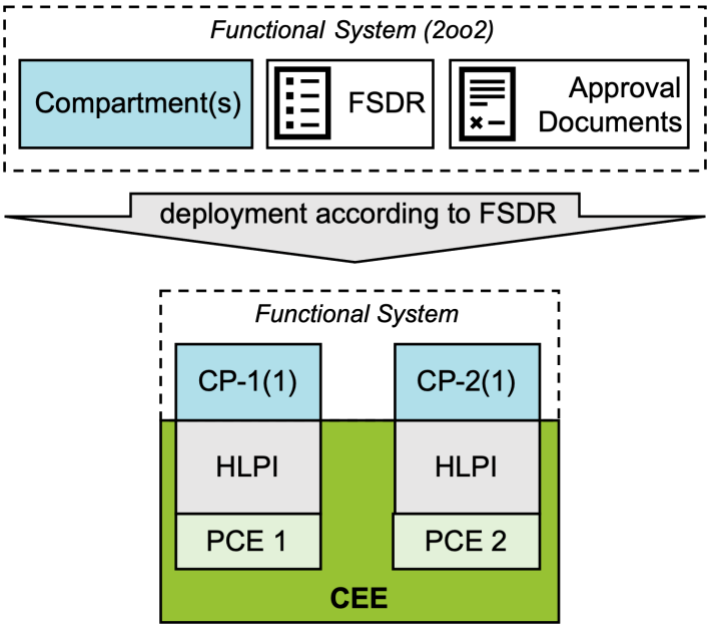


Figure 16: Example deployment of a single 2oo2 FS into one CEE

A more complicated – but hypothetical – setup employing two different FS, one 2oo2 and one 2oo3, is being deployed on a pool of three PCEs in the next example, as shown in Figure 17. Here, MPC-P06 (Consolidating more software on less hardware.) implicates that CPs from different FS can be executed on the same PCE while being free from interference according to MPC-A04. Therefore, only three PCEs are needed to execute five CPs belonging to two FS.

How exactly the CPs are distributed over the available PCEs is not important for the MPC except for conditions defined in the individual FSDRs. With FSDR, FS-specific rules for deployment can be demanded. This would mostly be used to state that CPs belonging to redundancy configurations have to be executed on distinct PCEs, but could also be used to make sure special applications like 2x2oo2 has groups of CPs executed in different locations (within a data centre, e.g. different fire protection zones, or even outside of a single data centre, e.g. for georedundancy).

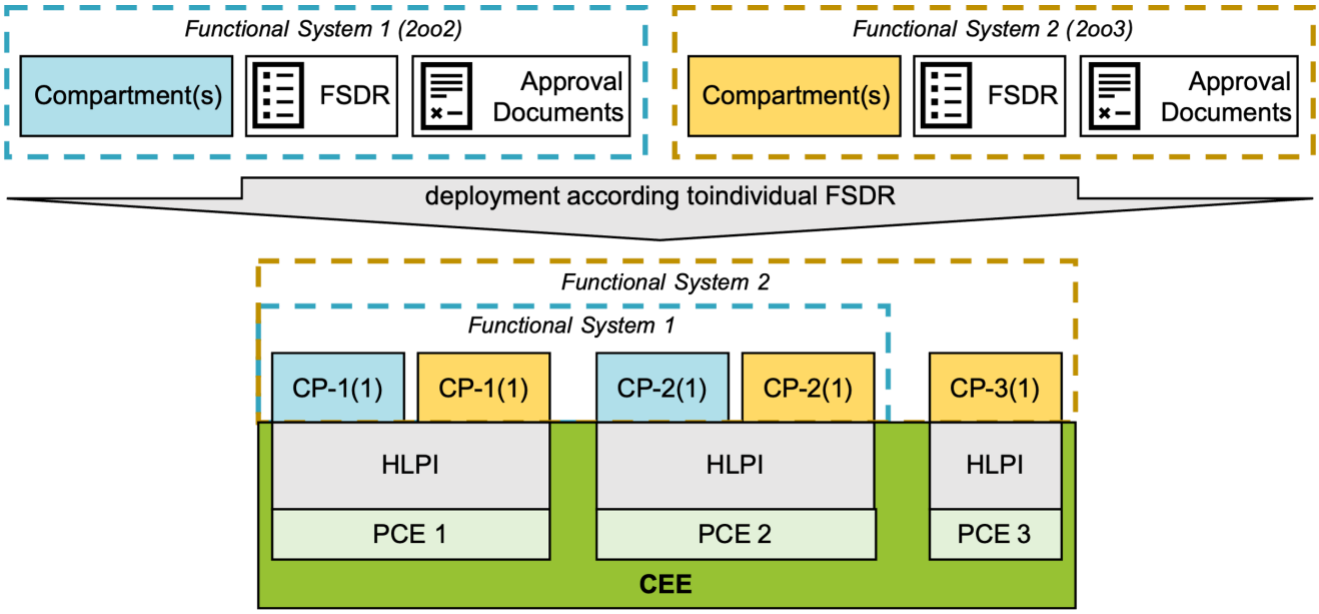


Figure 17: Example deployment of a 2oo2 FS and a 2oo3 FS into one CEE

Further examples can be constructed, especially as there is no limit to the number of CPs and FSDR complexity in a single FS. Of course, also simple single-CP examples for Basic Integrity usage not employing redundancy could be constructed. For the purpose of this description of HLPI modularity for the MPC, the two examples already given should be sufficient.

5.1.2 ALPI Modularization Architecture

Where in the HLPI context, as shown in the chapter before, the deployment of CPs onto a CEE is shown, the ALPI context is concerned with integrating Functional Applications (FA) into the CPs. It's a step that can be seen as a supplier implementing a business logic and packaging it into CPs to be bundled up to a full FS, together with the FSDR and approval documents.

The relevant difference is, however, that there is no mandate to employ ALPI methods to build suitable CPs, in order to allow the packaging of legacy or other software into CPs. For future projects, using the ALPI approach can be beneficial, though, especially when needing access to the services available in the MPC using I4, as introduced later in chapter 5.2.

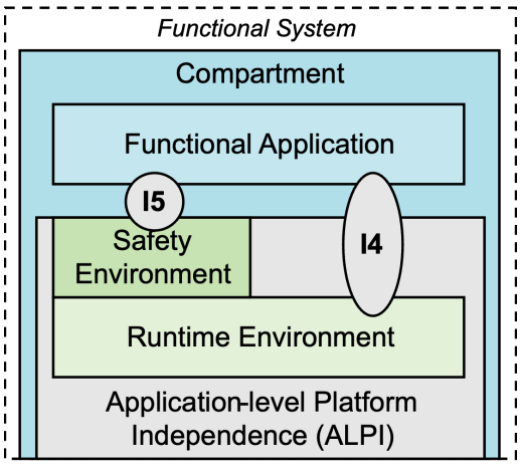


Figure 18: Modularization enabled by FS Compartments using ALPI

When employing ALPI, within a CP the Runtime Environment (RTE) and the Safety Environment (SE) together with their interfaces I4 and I5 are provided for the Functional Application (FA) to be used. A potential use-case could be to use a CP without FA, supplied by a platform vendor, and install a FA into this CP. This kind of "empty" CP – where empty is referring to the missing FA – could come prepared with all necessary services, the RTE and the SE, so that the FA developer can focus on implementing the business logic. Also, updates to any component in such an "empty" CP would be feasible in a generalized way, leading to installing the specific FA into updated (and potentially pre-verified) CPs.

In the following figure, such an "empty" Compartment containing the ALPI components RTE and SE is called ALPI-CP. This also implies that interfaces I4 and, where necessary, I5 are available within this kind of CP to be used for FA development.

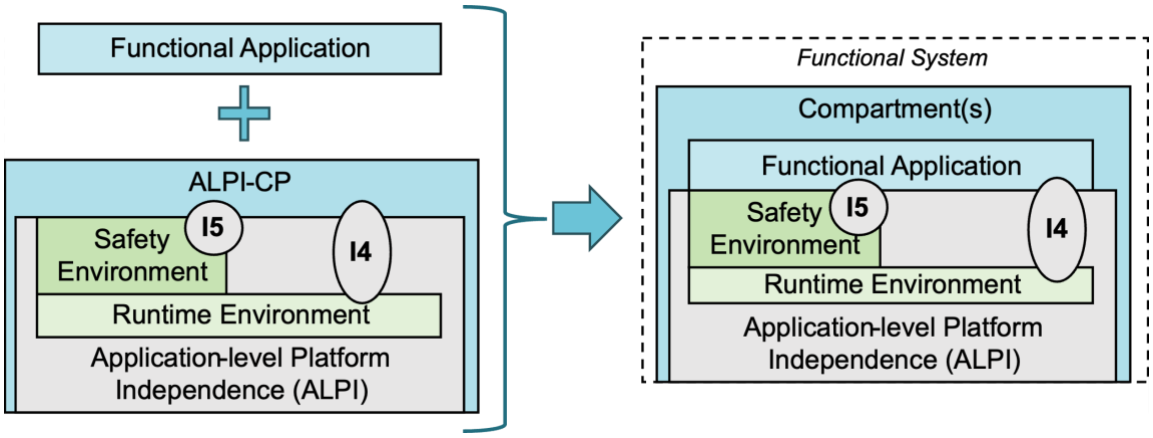


Figure 19: Installing an FA into an ALPI-CP to create a FS CP

Depending on the needs of the FA, multiple CPs with different numbers of FA Tasks (FAT) can be created in this process. The multiplicities for these different redundancy configurations are not shown here. Also, the remaining FS artefacts FSDR and approval documents are not covered here, as are all implications that would result from a full configuration management.

5.2 Service Architecture

While the previous chapter was focusing on how individual Functional Systems are constructed in the MPC, these FS have to rely on different services available to them and within the MPC. Many of the advantages of the MPC are in fact dependent on a multitude of services that provide, together with appropriate external interfaces, the ways to build, deploy and maintain a full MPC system.

As envisioned by the SP CE domain, service and management functionalities as remote update, diagnosis, IT-security are defined and provided separately:

a) by ERJU as one generic rail standard for all aspects relevant for the rail systems

b) by standard IT solutions for the aspects in context of VE, depending on the selected solution of the VE.

This separation has an impact to the service architecture and on the external interfaces of the platform. When the Compartment Execution Environment (CEE) itself cannot be expected to provide a standardized interface – due to allowing multiple options for standard IT solutions, only limited by the fact that these solutions need to fulfil the requirements towards them – additional entities have

to be defined to bridge these differences. These architectural components are introduced on a high level in this chapter and are further detailed and explained, as well as their interfaces, in chapter 8.

5.2.1 High Level Service Architecture

Figure 20 below shows the relationship between components within a CPI (compatible platform implementation, e.g., a product complying to MPC, see also chapter 3.11), and introduces the entities Shared Services, Platform Management, CEE, and their accompanying interfaces. The relationships are shown for an arbitrary number of FS within a CPI. The interfaces are shown in the same colour as the components that are responsible for their respective specification.

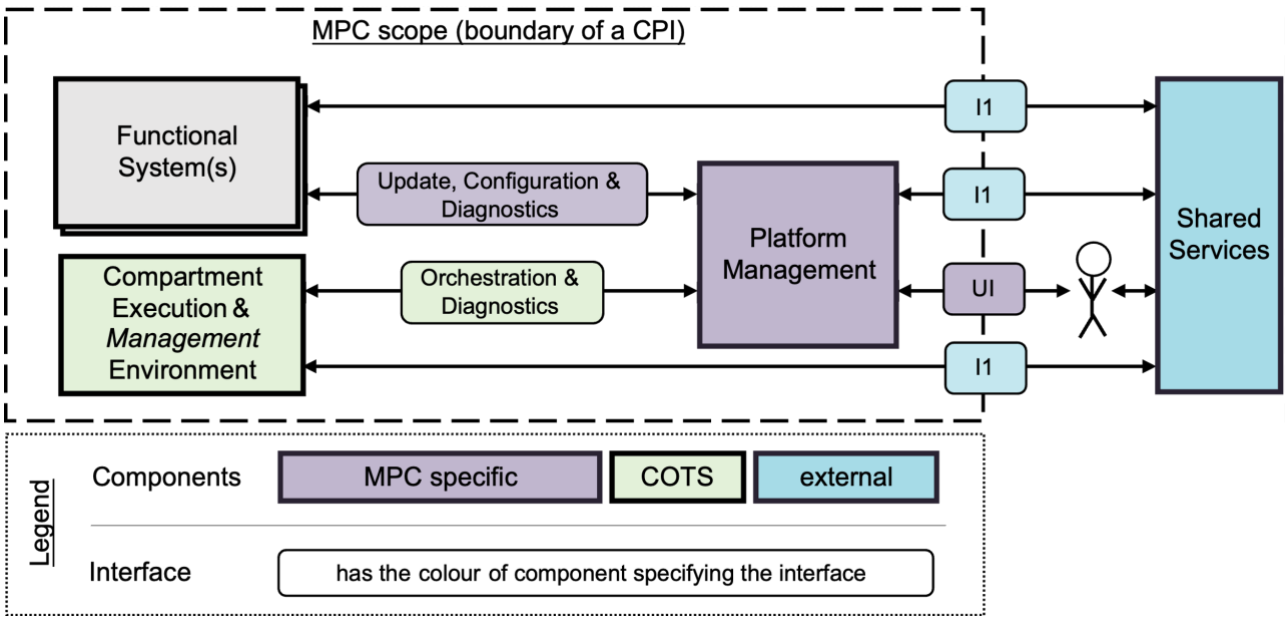


Figure 20: High Level MPC service architecture

Compartment Execution & Management Environment (CEME)

When (exchangeable) standard IT solutions are used for the VE (Virtualization Environment) inside CEE (Compartment Execution Environment), the assumption is that there is a technical management tool respectively interface for the management of the whole CEME, including the PCEs, VEs and VCEs, offered by the chosen COTS VE solution. In addition to the SP CE definition of CEE, the CEME also provides interfaces towards the Platform Management, e.g. for tasks such as on-boarding of new PCEs, setting up the VE and creation of VCEs. The interface, referred to in the figure as "Orchestration & Diagnostics", is expected to be part of the solution used and to be required to expose the necessary functionality for CEE management. Implementation details of interfaces I2 and I3 are specific to the chosen, suitable VE solution, and are not shown here.

Shared Services

The shared services as mentioned here are the combination of several services defined by multiple ERJU SP domains and contain security, update and diagnostics related functionality. Specification of them is out of scope in WP26, but highly relevant for the MPC.

Platform Management

The Platform Management (PM) is a new component added to the input of the ERJU SP CE domain and deemed necessary within the scope of the MPC. The PM controls the specific CEE, interacts

with the Shared Services (via I1) and collects management-related diagnostics data from the Functional Systems while also managing their configuration and updates. Its function within the MPC is to enable the usage of different suitable COTS CEEs while providing fixed interfaces (I1) to the outside. This way, CPIs with different PCE+VE+VCE combinations still expose the same interfaces to the outside. The PM is also providing a User Interface (UI), especially to support the user stories for the MPC as outlined in chapter 3.9. For example, central tasks of the PM are to properly deploy FS according to their FSDR, monitoring the FS's operational state and reacting to any non-regular situation that can occur. There is no guidance from the MPC on the amount of FS managed by a single PM, respectively the number of PCE available in a CPI. The implementation of the PM is expected to fulfil to Basic Integrity requirements only.

I1

The I1 connections shown in the figure are a combination of COTS interface definitions respectively protocols for standard tasks, such as time synchronization, and additional fixed interfaces defined by ERJU, e.g. for FS update. The I1 interacts with all, Function Systems, Platform Management and the CEE. However, not every component has to implement or comply with the whole range of I1 functionality. Especially for the interfacing to the CEE, the assumption is that standard interfaces for task targeting IT security are used by I1 and are available from potential COTS CEE solutions.

Detailed discussion of the interfaces

A more detailed discussion of the relevant internal and external interfaces can be found in chapter 8.

5.2.2 CEME and AEE Service Architecture

To support update and diagnostics within both, the CEME and the AEE areas, at least two non-exclusive variants of endpoints are possible. This is shown in the next figure.

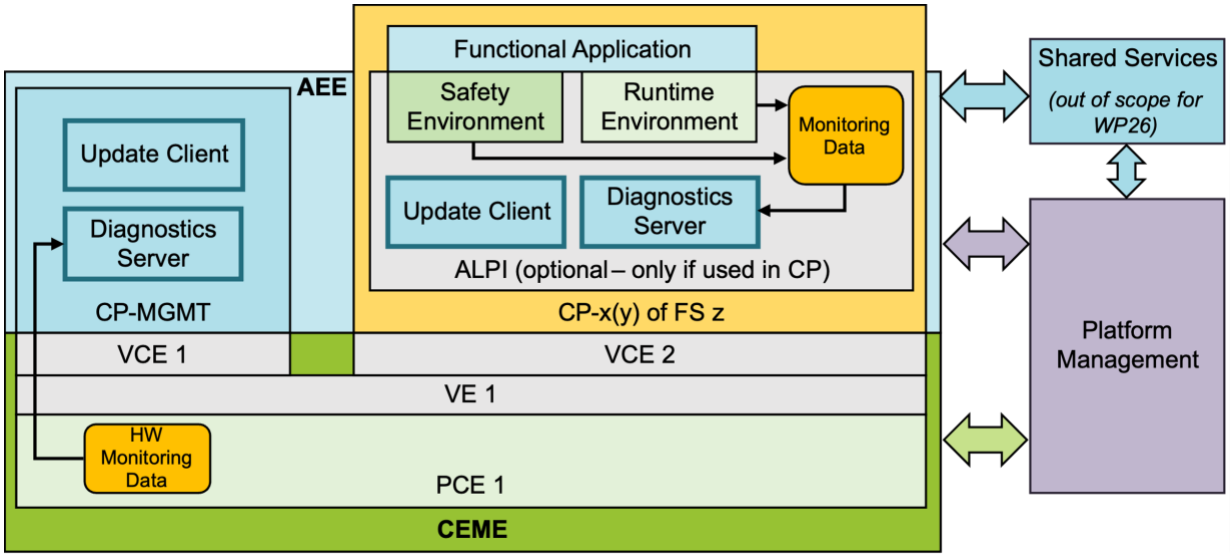


Figure 21: Service architecture endpoints in AEE for FA and PCE data collection

CP-MGMT

The Management Compartment (CP-MGMT) is a way to bridge the standard IT VE environment to the AEE. Here, the example given in Figure 21 is the collection of hardware monitoring data (e.g., temperature measurements of certain components, see chapter 6.7) from the PCE, being

transported through the VE and VCE to an exemplary “Diagnostics Server” in the CP-MGMT. This diagnostics server can collect the data and exchange it with the Shared Services and the PM. Similarly, an update client can be implemented. A dedicated CP-MGMT would be dependent on the VE used and use its methods and interfaces to access PCE, VE and VCE data. The CP-MGMT would be managed by the PM. Methods for achieving the data transfer from PCE to CP are discussed in chapter 6.6

ALPI Services

When I4 or I5 are being used, the data collection can happen within the confines of the ALPI, as shown in an example for VCE2. Here, monitoring data of the FA execution can be collected within the appropriate Compartment and transported to the Shared Services and PM. Similarly, an update client would be located within the CP.

Other services

The update and diagnostics services are only two of the possible services. Security services for APM, PKI and time synchronization are not shown here but would be integrated into the FS CPs.

Combination of CP-MGMT and FS CP

For certain implementations of MPC, it might be feasible to integrate the functionality needed within the CP-MGMT into the normal FS CPs. This can reduce the number of CPs in a given environment, as one CP-MGMT per PCE is necessary. However, it creates a strong dependency on a given VE also in the normal FS CPs. Additionally, only one CP per PCE should read and report data from the VCE, VE and PCE, to avoid exclusive access problems and double reporting issues during operation. This creates dependencies between different FS that might not be aware of each other, leading to further complications.

Detailed discussion of the interfaces

The detailed derivation of the necessary interfaces to facilitate the data and control flows is shown in chapter 8.

5.3 Additional Assumptions

The Modular Platforms Architecture imposes additional assumptions on the definition on the MPC and its components, namely HLPI, ALPI and the external interfaces. They augment the SP CE domain recommendations and assumptions collected so far.

MPC-AA01 I1 shall always be used (includes potentially non-standardized orchestration interfaces, namely the OI, and nested orchestration in future cases where there are containers inside VMs that need to be tended towards).

MPC-AA02 Usage of I2/I3 does not mandate I4/I5, and vice versa.

MPC-AA03 The message content when using I4 and I5 provided communication mechanisms is not specified in/by work package 26 respective the Modular Platform Concept.

MPC-AA04 As a first step, systems that can be orchestrated based on manual human user input via I1, leading to a so-called “static configuration” are in the scope of this work package. The Modular Platform Concept, especially the Platform Management component, however, can in the future be expanded to include dynamic configuration approaches. This requires the safety implications of this approach to be solved.

MPC-AA05 Safety & Reliability: Only the safety layer is expected to be an up to SIL4 grade product. Kernel, orchestrators, etc. are not expected to be SIL grade.

MPC-AA06 Over I1 safety related information gets exchanged but safety is assured by the safety layer of a functional system.

MPC-AA07 I1 can be used to enable remote maintenance.

MPC-AA08 Virtualisation Software and Hardware of the CEME are considered providing non-safety related functions. The used COTS products raise the need to be qualified according to EN50716.

MPC-AA09 HLPI and ALPI use is agnostic to each other.

MPC-AA10 The deployment configuration is checked by the safety layer to ensure restricted deployment (especially to avoid replicas to be deployed on the same physical HW).

MPC-AA11 Updates of components outside of the safe part (e.g. security updates) should not affect the safe part uptime.

5.4 CONCLUSIONS

The Modular Platforms Architecture builds on top of the ERJU SP CE domain input and prior work of work package 26 itself. The previous introduction of ALPI and HLPI layers helps to simplify systems views and the discussion, and also represents concrete areas of expertise and future product development.

The modularization and service architecture approaches shown in this chapter build the basis for the following chapters, detailing how HLPI, ALPI and interfaces can be specified.

From the architecture perspective, one open point remains: A combined modularization architecture proposal showing how the deeper levels of FS (e.g., compartments, RTE, Functional Applications, etc.) interact with the interfaces introduced in the service architecture, as well as with the Platform Management and Shared Services (see Open-011).

6 HARDWARE-LEVEL PLATFORM INDEPENDENCE (HLPI)

6.1 INTRODUCTON

One of the primary objectives of HLPI in this new architecture is to facilitate the integration of multiple Functional Systems, which may originate from different suppliers and possess varying safety integrity levels, onto a shared Physical Computing Element (i.e., hardware). This shall be accomplished through the implementation of a virtualization layer, which ensures maximum hardware independence and facilitates seamless integration.

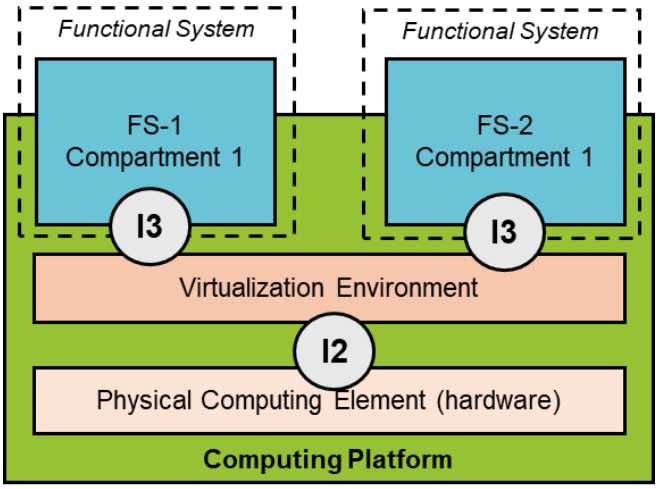


Figure 22: Aggregation of FS Compartments on same Hardware

By utilizing a standardized IT solution as a common virtualization environment, the architecture benefits from a reliable and well-established solution. It includes all the necessary management functionalities for efficient software orchestration. Furthermore, the streamlined management of various commercial off-the-shelf (COTS) hardware versions and types from different suppliers will be significantly simplified. This approach is further enhanced by the availability of qualified hardware from multiple hardware suppliers.

To accomplish this, it is essential to define the specific interfaces between hardware, virtualization layer, and Functional System software in alignment with the safety and security architectures. It is important to note that the virtualization software and hardware are considered to be non-safety components without safety implications, yet they must still meet all relevant security requirements.

The subsequent sub-chapters provide an overview of various aspects related to the layered architecture, including requirements for architecture elements and any outstanding issues that need to be addressed. The primary focus is to identify requirements from the perspectives of safety (see chapter 6.6), security (see chapter 6.8) and availability (see chapter 6.9).

6.2 ASSUMPTIONS

6.2.1 FS without direct I/O interfaces

In scope of Computing Environments / Modular Platforms are FS with communication-based interfaces to other FS.

The communication interfaces between FSs shall be standardized. By this it may not make a difference if connected other FSs are running on the same computing elements or not. Even FSs running on specific hardware (e.g. legacy systems which are already installed) can be connected via the same standardized communication interfaces.

I/O control functionality is out of scope of HLPI. Such I/O control functionalities need use case specific HW solutions for the individual use cases, e.g. point controllers need other specialized physical functionality and interfaces than signal controllers. By this a hardware independent standardization of the system internal architecture of I/O controllers is not easily possible.

Functional Systems that require I/O control functionality are connected via the communication-based interface I0 to use-case specific I/O controllers.

6.2.2 FS internal communication

Each solution of an SE supports communication-based interfaces between the FS compartments without any constraints regarding the usage of the network, as e.g. no kind of constraints as “direct LAN cables between the VCE”.

6.2.3 FS time behavior

Time critical processes in HW related functionalities are realized behind I0 within dedicated I/O controllers.

Note that latency times of communication technology leads to additional delay in message processing. This must be foreseen in the overall architecture and safety case.

The trackside overall architecture (interlocking, radio block centre, object controllers) is already defined in such a way that an interlocking needs to show reaction times in a range of 1−2 seconds.

The new overall architecture within the on-board system must be defined in such a way that the FS can react in time ranges below 1 second, with potential real time constraints depending on the function[5]. Time critical processes in HW related functionalities with faster timing requirements are realized behind I0 within the use-case specific I/O controllers.

6.2.4 VE as non-safe software without safety relevance

The proposed safety architecture is based on a non-safety related virtualization environment. By this the functionality of the virtualization environment such as e.g., task scheduling shall not affect safety, but it may affect availability.

Example:
If the scheduling of SW parts of aggregated FSs is not processed as required by the individual FS, then the FS itself shall identify this problem and react in the needed way from view of safety.

In case of a safety related FS the safety concept of the SE must not require a specific behaviour of the virtualization environment from view of safety. A misbehaviour of the virtualization environment shall never affect safety, only availability. See **REQ-HLPI-1.**

[5] For example, there is a time constraint of 1 second from balise reader to brake in the ETCS on-board context. This constraint needs to be budgeted over all systems in the reaction chain.

6.2.5 Standardization Update Process for FS Compartments

It's assumed that the process for the handling of FS Compartments in context of updating a FS is defined by the System Pillar Transversal Group and hence out of scope for WP26.

6.3 Resource PartiTIoning for FS Compartments

From view of availability (i.e., stable running FS compartments) it's essential to assign physical CPU resources (e.g., cores, cache / shared bus) exclusively to individual FS compartments. Assigning virtual cores to individual SW components within the FS compartments (e.g. Functional Application Replica using one CPU core exclusively) can only happen within the Compartment.

The VE shall provide the mapping of CPU cores exclusively to VCE, see **REQ-HLPI-2.**

The CPU performance provided by the mapped CPU resources must be guaranteed for every timepoint during the runtime of an FS Compartment, see **REQ-HLPI-3.**

Parallel installation of additional FS Compartments (of other FS) in additional VCEs on the same Virtualization Environment Instance must not have any impact on the guaranteed CPU performance (cores) for running FS Compartments, see **REQ-HLPI-4.**

Open point	What kind of further HW architecture aspects will be "bottle necks" in parallel usage by independent FS compartments running aggregated on same computing element? Memory bandwidth? Network bandwidth? See Open-001.

6.4 FS Compartment Configuration of the VE

6.4.1 Modularity and independency of VE Config for FS Compartments

The individual VE configurations of FS compartments shall be modular and independent. Each FS Compartment shall have its own configuration for the VCE. Adding or deleting of FS compartments onto the VE instance must not have any impact on the VCE config of the other FS compartments. See **REQ-HLPI-5.**

6.4.2 Compatibility at VE interface

The virtualization environment shall provide defined and stable interfaces for the configuration of the VCE usage by FS compartments. A new version of the VE may not have any impact onto the VCE Configuration of the FS compartment.

Each change in the user interface for the VCE configuration shall be compatible in such a way that existing VCE configs (of already running system) can be used furthermore. See **REQ-HLPI-6.**

6.5 Interface I3 and VE Architecture

The interface I3 of the virtualization layer describes the basic aspects regarding the virtualization layer in context of aggregation of several systems with their own guest operating systems and

possibly different safety criticality levels and possibly provided by several vendors, running together on the same hardware.

This interface I3 is not an interface in sense of "programming interface" but it's the definition of needed functionalities and features within the virtualization layer from view of the Functional Systems running above.

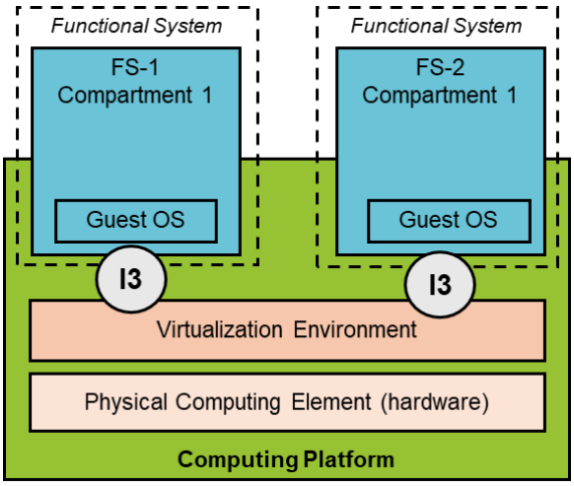


Figure 23: Basic architecture with VE and interface I3

Basic aspects of the different architecture variants as Container Solution (without Hypervisor) and Hypervisor Type-1/2 for the aggregation of FS compartments are discussed here in the following sub-chapters.

For dependencies from the perspective of safety see chapter 6.7, for security see chapter 6.8 and for availability see chapter 6.9.

Open point	Architecture: how to handle the message-based interface of the NHA (see chapter 6.7.1) to FS Compartments above – is this interface a part of I3? See Open-002.

6.5.1 Hardware Independence

One of the main goals is to achieve hardware independence at the interface I3, is to be able to change the used physical hardware without any impact to the FS compartments.

The runtime interface I3 of the VE shall provide HW independence for the FS compartments running above.

The FS related configuration of the VCE shall be independent from the concrete used hardware. Changing the hardware (e.g. replacing HP-servers by Fujitsu-servers) shall not have any impact onto the VE config of the FS compartments running above, meaning it shall be possible to replace a used physical hardware during runtime of the FS without touching the FS related configuration of the VCE.

To avoid resource consuming emulators within the VE the CPU instruction set should be defined for the interface I2, for this see chapter 6.6.

HW related information, which is needed by the SE (see chapter 6.7.1), must be provided in an abstract generic way to achieve HW independence at the interface I3.

6.5.2 Container

Usage of container technology means:

- No flexibility in OS type, all FS must be based on same operating system type (e.g. Linux). This yields the advantage, that only one OS has to be maintained.
- Dependencies between the common operating system kernel and the FS running above.
- Weak “isolation” from view of resource usage, several FS are using same OS kernel resources. Overall integration necessary, but potentially a ressource-saving approach, e.g., for on-board, where rolling stock hardware ressouces might be limited.
- Overall performance testing necessary with different systems involved. By this it is assumed that the integration and qualification of containerised FS Compartments stays in the responsibility of one single vendor or integrator.

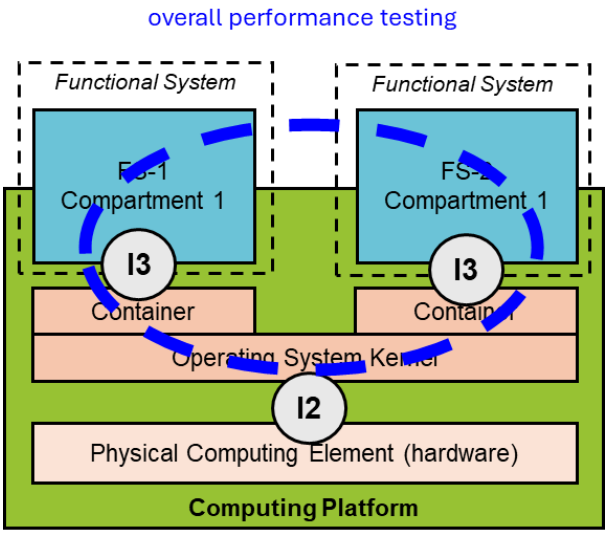


Figure 24 Container as VE

6.5.3 Hypervisor

Usage of hypervisor technology means:

- Flexibility in type of the guest OS, each FS can be based on own operating system type (e.g. Linux, Windows), accompanying additional effort for maintaining the guest OS.
- No direct technical dependencies to the FS compartments running above.
- Best available solution for the “isolation” of the aggregated FS compartments from view of resource usage (cores, memory, communication), coming along with additional ressource demands and a potential higher performance decrease in comparison to containers.

From view of **safety** a technical possibility is needed to implement functionalities running on the host operating system bare metal on the physical computing element, see chapter 6.7 NHA (Native Hardware Access).

From view of **availability** the best possible resource isolation for the FS compartments is essential. By this the common host operating system behaviour and workload shall be as deterministic as possible.

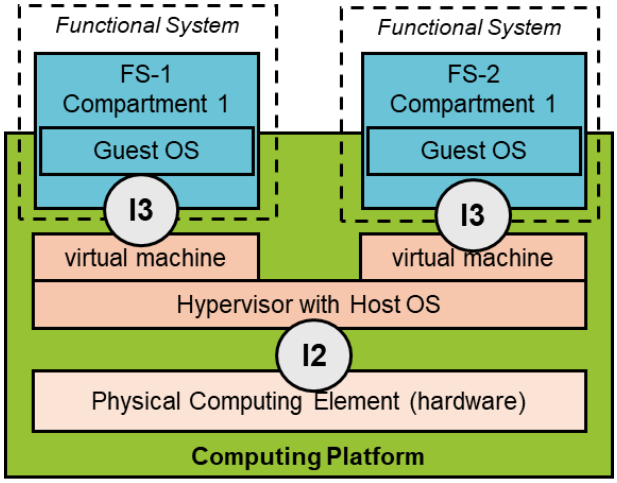


Figure 25: Hypervisor as VE

Exemplary available solutions:

- KVM integrated in Linux
- KVM integrated in Red Hat Enterprise Linux
- VMWare
- Windows Hyper-V

6.5.4 Hypervisor and Container

The combination of Hypervisor and Containers (running within a shared virtual machine) leads to known challenges, increasing the system complexity (e.g., for the orchestration of compartments).

As introduced in 6.5.2 and depicted in Figure 26 below, multiple Functional System compartments running aggregated as containers in one VM raise the need of overall performance testing. Resource isolation isn't established in such a way that compartments of FS-2 and FS-3 can be provided independently, as this is the case for the FS-1 compartment deployed in a separate VM.

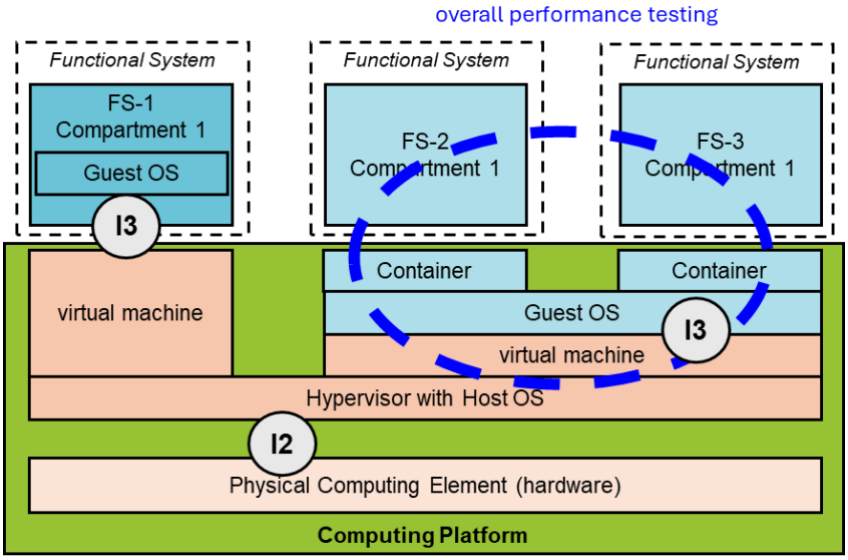


Figure 26: Hypervisor and Container as VE

6.5.5 Summary

6.5.5.1 Trackside use case

For the **trackside use case** of a data centre the isolation and independency of the aggregated FS is highly essential for the individual handling of FS compartments during the lifecycle of the data centre.

If some FS as interlockings or RBCs are already running with full safety responsibility then it must be possible to bring additional FS into the data centre without touching the running FS, even without the need of extra integration tests of the already running FS on the VE.

By this the trackside use case needs best possible isolation of the aggregated FS. Best possible resource partitioning is needed to achieve independency and availability of the FS technical interference between the aggregated FS in context of resource usage.

Such an isolation is achieved in best possible way by the usage of a hypervisor with virtual machines for the FS compartments. From safety view some functionalities must be provided by native running software NHA (see chapter 6.6). This means that a hypervisor is the preferred solution.

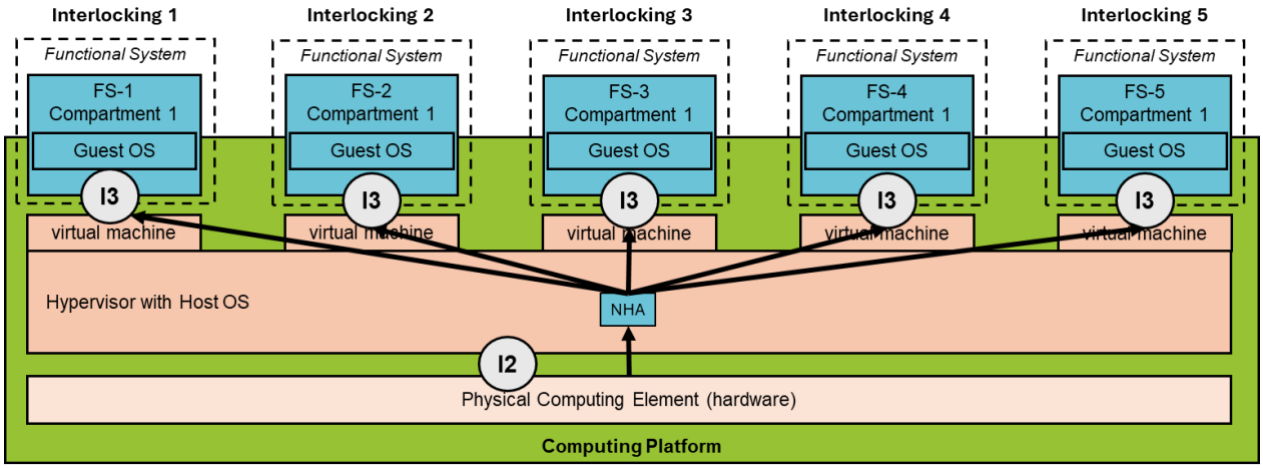


Figure 27: Trackside use case data centre: FS isolation by virtual machines

6.5.5.2 On-board use case

In the **on-board use case** Software changes or enhancements will not be done during the operation of trains. A train will always be stopped for SW maintenance purposes. By this the independent handling of the individual FS in context of software maintenance is not as essential as for trackside.

Additionally, the physical computing elements within the train will not provide the same amount of CPU resources as in a track side data centre, meaning the computing element resources are limited and must be used very efficiently.

For such an on-board use case a container-based architecture will be possible to run basic integrity software, which does not depend on a concrete SE. Such container based aggregated BIL FS must be integrated holistically to ensure the availability of the overall system.

Both, hypervisors or containerised approaches are in principle appropriate as well for on-board safety critical Functional Systems. Depending on concrete needs and existing constraints a trade-off between flexibility, extendibility and available resources may drive a concrete solution decision. Nevertheless, for a multi-vendor setup a hypervisor solution should be much more appropriate.

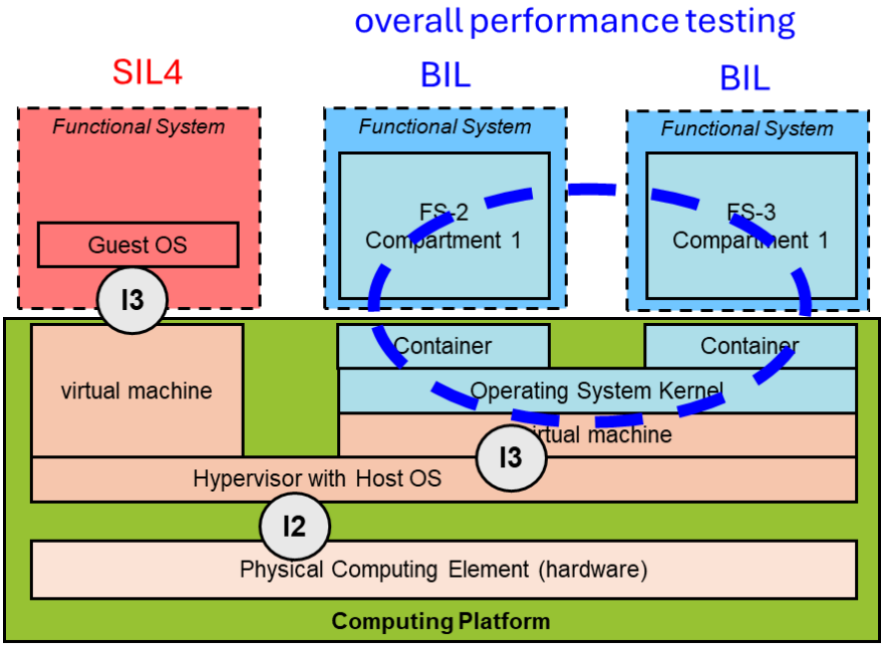


Figure 28: On-board use case: up to SIL4 in virtual machine, BIL in container

6.6 Interface I2 and HW Architecture

The interface I2 of the hardware is not an interface in the sense of a programming interface but rather the definition of required CPU characteristics like processor instruction set, needed CPU features like performance, etc. from the perspective of the Functional System running aggregated on the virtualization layer above.

In this context the topic of "flexible and HW-independent usage of HW" must be analysed regarding the technical details of the HW which need to be defined as "generic standard".

The goal of such a standardized architecture is to achieve full flexibility in replacing the used hardware by another hardware without changing the software of the Functional Systems running above.

Assumptions:

- CPU architecture is specified by I2/I3 and has to match the FSDR of deployed FS.
- No usage of CPU emulation below I3 to avoid potential issues with systematic errors in the emulation layer.

For future proof solutions it's essential that the safety concept of each SE solution is basically independent from the processor instruction set to be able to change the CPU architecture without impact to the safety concept, see **REQ-HLPI-7.**

A first list of potential requirements needed to define this "interface" is following and is subject to further study.

Requirements towards the virtualization layer above I2

- Flexible support of "incompatibilities in detail" in context of hardware spare handling, see **REQ-HLPI-8.**
- Flexible support of differencies in new variants of hardware, see **REQ-HLPI-9.**

Requirements from the VE and Functional Systems running above towards the hardware

- Basic hardware architecture (CPUs, cores)
- Minimum requirements in context of CPU performance, communication, …
- MTBF values of the hardware
- Relevant aspects on interface to hardware vendor (e.g., compatibility of hardware versions)

Open point	The details of the requirements towards the hardware have to be defined, see Open-003.

6.7 SAFETY

6.7.1 HW related information for SE

For the aggregation of several FS compartments of different safety integrity levels, possibly provided by different suppliers, on the same virtualization environment the safety architecture is essential.

Figure 29 shows the proposed safety-architecture for flexible and efficient handling of aggregated FS (on the same computing element) possibly provided by different suppliers. For efficient handling of safety related FS compartments running decoupled in parallel on same VE it's essential that the VE software below the FS compartments does not have safety relevance. Safety relevance of the common VE layer would lead to high effort in overall integration of the FS compartments (provided by different vendors) with the VE due to "safety related resource sharing of the VE". So it's proposed that this VE software is non-safe (possibly provided by a third party).

Each solution of safety environment shall define its own safety concept in a way which allows the usage of non-safe VE, each misbehaviour of the VE has to be identified by the SE and the SE must react safe, see **REQ-HLPI-1.**

VE will serve information to the safety environment above, but safety responsibility is completely on side of the SE.

The figure below shows for the basic safety architecture the SW layers running on one individual physical computing element. This architecture is independent from the details of the safety principle as e.g. 2oo3 or 2x2oo2, meaning it's the same architecture for the other compartments of the up to SIL4 FS running on other physical computing elements (but not shown in the figure).

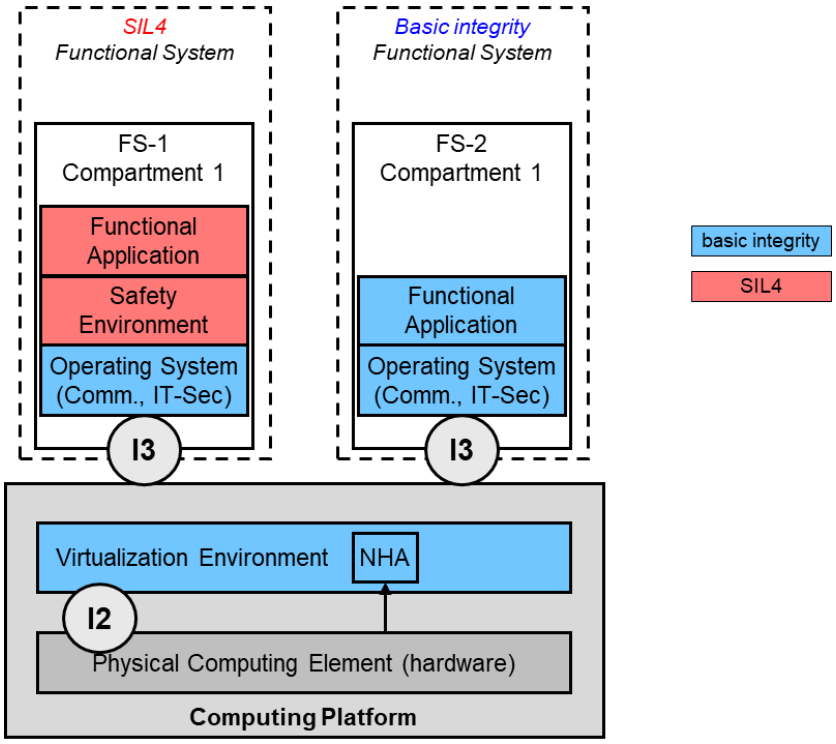


Figure 29: Safety architecture

It depends on the solution specific safety-layers which kind of behaviour and which level of "guarantee" for this behaviour is needed by safety layers to let the safety-layer run on non-safe virtualization layer.

As described later in this chapter, some safety related functionalities within an SE need a reliable access to the **physical** computing elements.

The information cannot be created by FS compartment itself running within a virtual machine because reliable access to underlying physical hardware is not guaranteed for SW running in a virtual machine.

This information cannot be provided by a 3rd party VE because it would not be able to argue the quality of this information (by 3rd party VE) on side of the SE.

As a result, it's necessary to provide this hardware related information by a **native running reliable software** (reliable in the way that the provided information is not influenced systematically). This native running software is called "Native Hardware Access (NHA)", and this software needs direct hardware access without influence by virtualization or emulation. Reference to 50129 is related to ensuring the nominal operating conditions for the hardware and the system. The correct operation can be ensured by directly monitoring the parameters (defined by NHA) or indirectly by monitoring the behaviour of the system. The requirements and example use cases for this are described within the next chapters.

6.7.1.1 Distribution of up to SIL4 FS Comp on different CPUs

The Safety Environment needs a reliable information about the identification of the physical CPU hardware on which the compartment is running. This information can be provided in the required reliable way by a dedicated software component NHA (Native Hardware Access).

See **REQ-HLPI-10, REQ-HLPI-11**

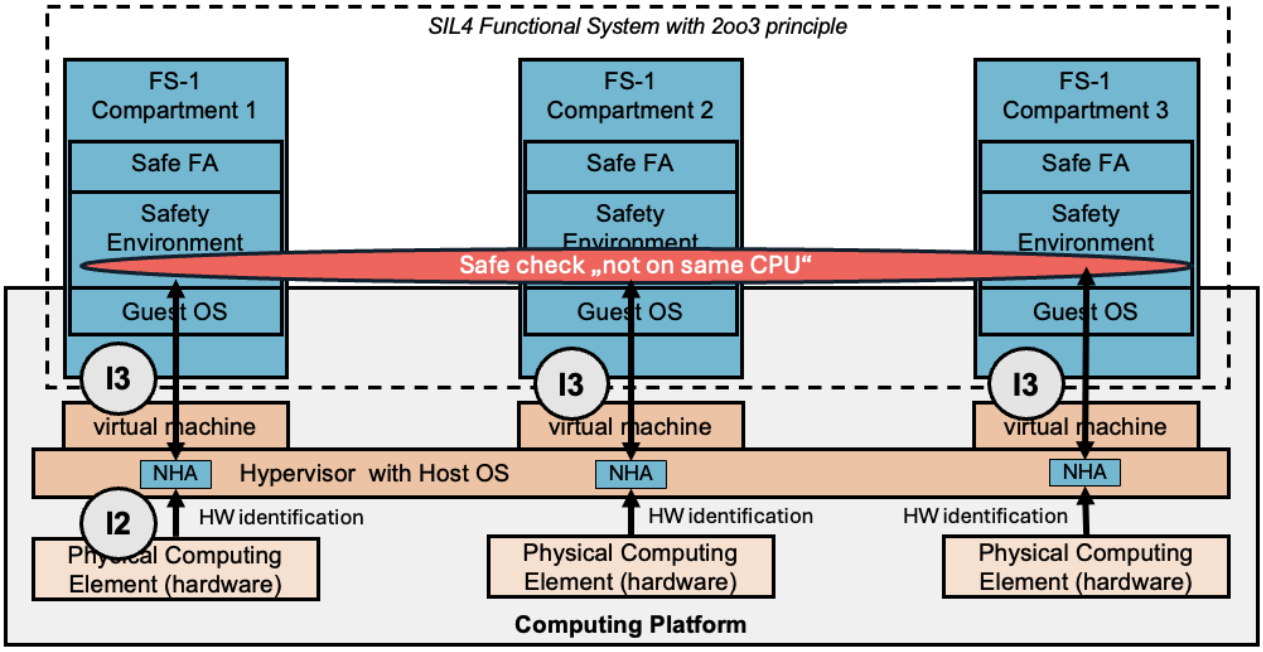


Figure 30: CPU identification provided by NHA

Open point	What is the criteria for unique CPU identification? MAC address? TPM content? See **Open-004**.

6.7.1.2 Core Usage Information

Depending on the safety concept of the SE, an information about the used cores might be needed (to provide the information for safety checks of the SE).

Open point:	How to solve the relationship of used CPU cores (used by the FS compartment within the VCE) and information provided by NHA? See **Open-005**.

6.7.1.3 Independent clock source for the creation of a safe monotonic time

The SE needs to realize time related safety critical services as e.g. cyclic and synchronous replica processing or timer-services for the FA.

To achieve a “safe monotonic time” the SE needs 2 independent monotonic clock input sources for safety, an additional 3rd (in case of 2oo3) or even 4th (in case of 2x2oo2) independent input source for availability. Each of the clock input sources must be provided from a different physical HW and all these input sources may not be influenced in systematic way.

This information can be provided in the required reliable way by a dedicated software component NHA (Native Hardware Access), which is running natively on the hardware, see **REQ-HLPI-10.**

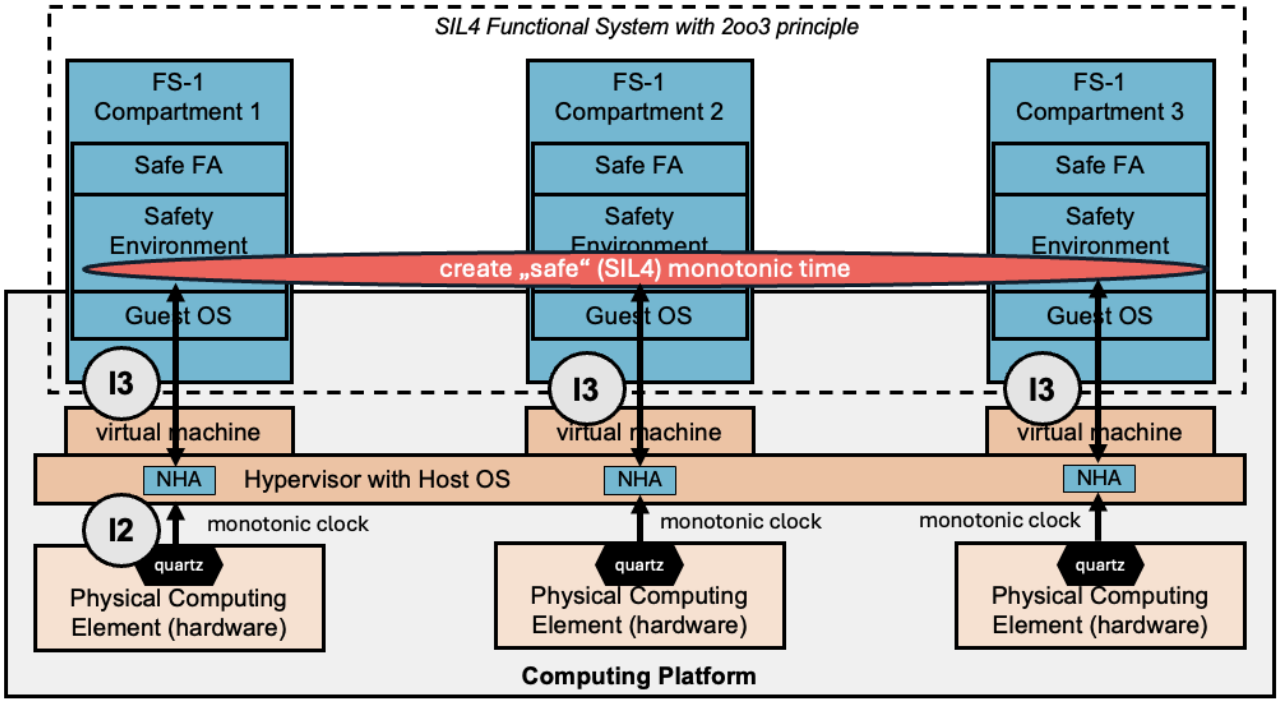


Figure 31: Monotonic clock input source provided by NHA

6.7.1.4 CPU temperature

Depending on the safety concept of the SE an information about the CPU temperature is needed (to provide the information for safety-check on side of SE).

This information can be provided in the required reliable way by a dedicated software component NHA (Native Hardware Access), which is running natively on the hardware, see **REQ-HLPI-10.**

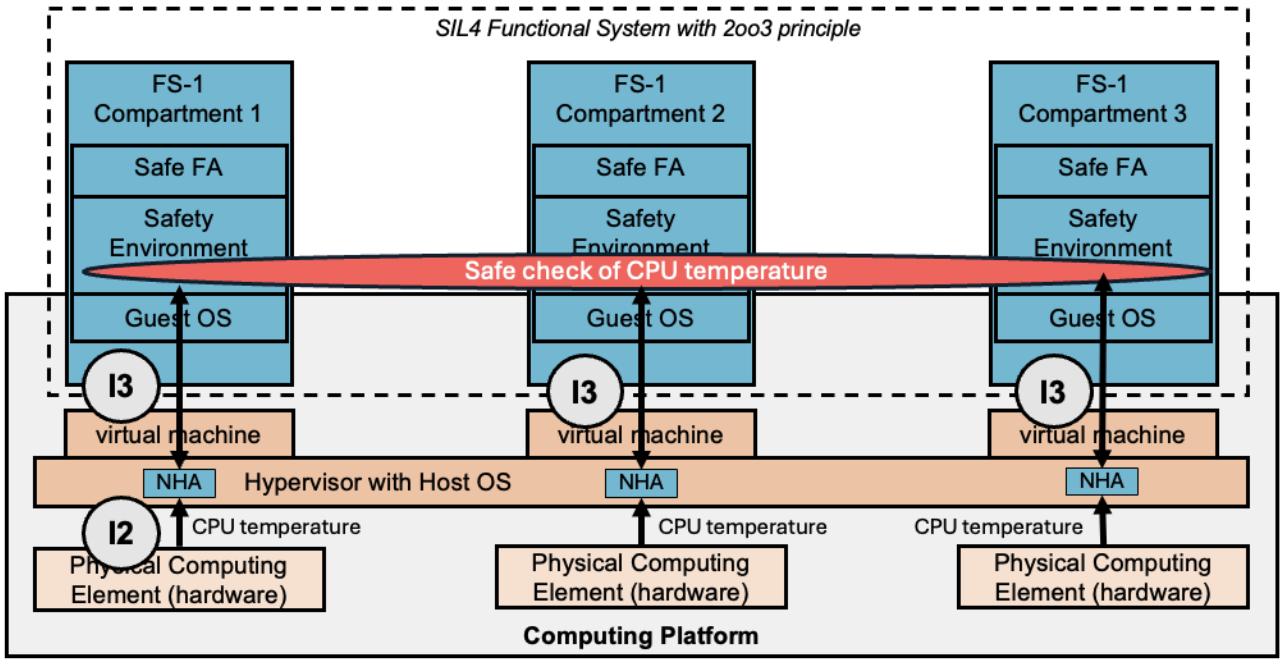


Figure 32: CPU temperature provided by NHA

> Open point: the details regarding sensor information provided by NHA in context of temperature must be clarified, see Open-006.

6.7.1.5 Voltage

SEs may need to monitor the voltage going to different hardware parts of the computing platform the software is running on. This helps detecting any potential device failures or short circuits.

This information can be provided in the required reliable way by a dedicated software component NHA (Native Hardware Access), which is running natively on the hardware, see **REQ-HLPI-10.**

> Open point: the details regarding sensor information provided by NHA in context of voltage must be clarified, see Open-007.

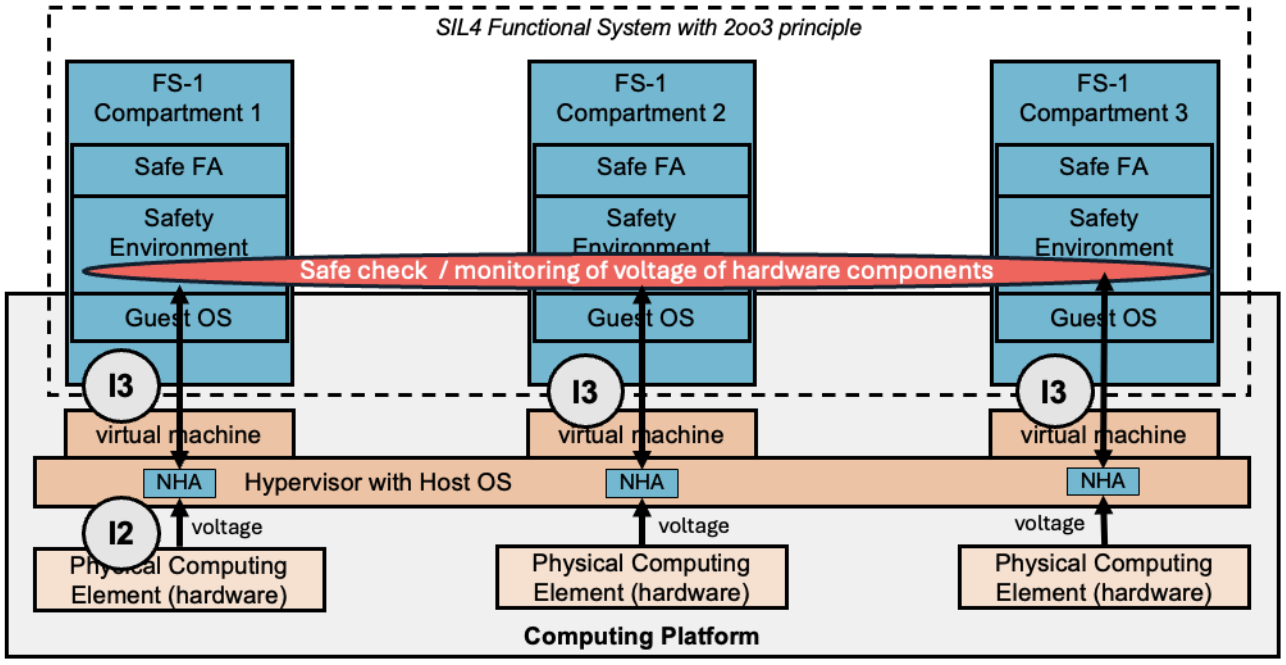


Figure 33: Voltage information provided by NHA

6.7.1.6 Summary

The detailed solution for such a native running software NHA depends on

a) the concrete solution of the VE as e.g. host OS and developer interface of the VE

b) the required hardware information depending on the safety concept of a SE solution

c) the provided data of the concrete hardware via the interface I2.

Since NHA functionality depends on the concrete needs of the SEs, it is challenging to standardise it. Further study is necessary to enable support for diverse SE solutions while not creating limitations on COTS hardware selection.

Different SE solutions with different safety concepts provide different SRACs in context of the required HW related data. For a generic usage of the same COTS hardware for all SE solutions the COTS hardware has to provide all required data via **interface I2** to NHA.

The grade of HW independence of such a NHA functionality depends on the SE solution specific details of the required information at the **interface I2**, meaning the HW related SRACS of the SE solution.

In the context of HW related SRACSs we have to differentiate between:

1) basic hardware information:
   - CPU HW identification (e.g. by MAC address, TPM content)
   - steady system clock
2) specific hardware information which depends on concrete HW details like specific sensors or cores, e.g.:
   - core pinning
   - CPU and/or other temperatures
   - voltages

For **basic hardware data 1)** the access by NHA will be possible without dependency to the concrete interface of the hardware. NHA gets the basic hardware data via OS functionalities of the Host OS of the VE. For this basic hardware data a change of the hardware should not have an impact onto NHA, SRAC fulfilment by NHA should be possible in a generic way.

For **specific hardware data 2)** the access by NHA must probably be adapted specifically for different HW variants. In case access to physical hardware details as sensors is not the same for different hardware variants, this would mean: a change of the hardware may have an impact onto NHA and the SRAC fulfilment by NHA has to be re-validated for the new hardware variant and new version of NHA. Anyhow, standardised access to sensors should be strived for where feasible to avoid such re-validation efforts.

Open point:	It must be clarified, if the required information from the physical hardware can be provided via standardized interface I2 or if the NHA functionality must be adapted for different HW variants. See **Open-008**.
Open point:	the responsibility and technical handling (installation and update) of such a NHA software must be clarified. See **Open-009**.

6.7.2 Safe handling of Software

Handling of safety related software by basic integrity management software running on non-safe operating systems and VE leads to the hazard that handling of the software (starting and stopping, storing and deleting) is not reliable from view of safety.

Example:
It cannot be guaranteed in context of SW update of safety related SW that all old SW components are stopped and deleted and newly installed SW components are started. Inconsistencies due to mixture of old and new versions of the safety related SW components must be identified by the SE.

Hence the SE has to ensure the consistency of all involved safety related SW components for the concrete overall version of the FS, see **REQ-HLPI-12.**

Open point:	the safe handling of safety critical software in context of a non-safe VE with standard orchestration tools must be clarified, as e.g. to avoid unallowed installation and starting of FS duplicates, see Open-010.

6.8 SECURITY

The software packages VE and each FS compartment are each an individual secure component.

3rd party provider of the VE has to consider IEC 62443 to provide certification as needed, see REQ-HLPI-13.

Open point:	overall certification of the secure device needs to be clarified, see **Open-012**.
Open point:	the architecture for access to the TPM of the PCE must be clarified in context of functionality secure boot and certification for IE 62443 SL3, see Open-013.

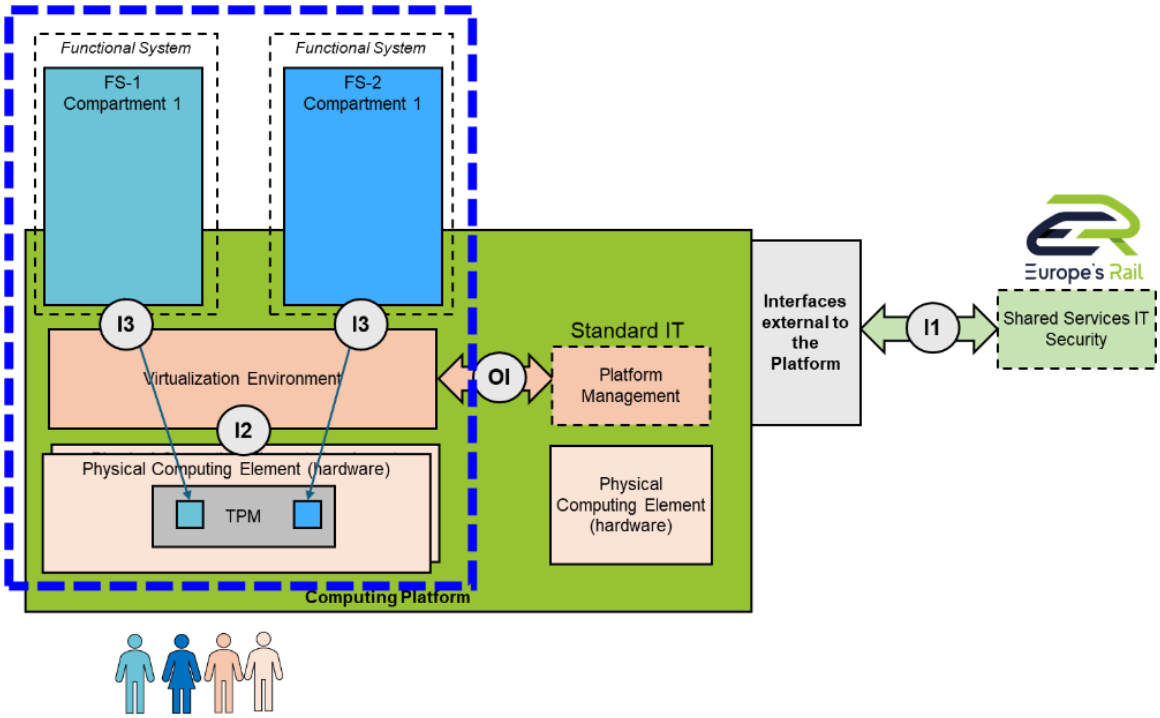


Figure 34: Secure device

6.8.1 ERJU Security within the FS Compartment

Each FS compartment contains its own implemented realization of IT security functionalities.

Workload for IT-security isolated by separated virtual machines (no cross-dependencies between compartments of different FS).

Need for IT sec patching depends on concrete solution, probably several patches necessary (individual patch for each FS compartment and VE).

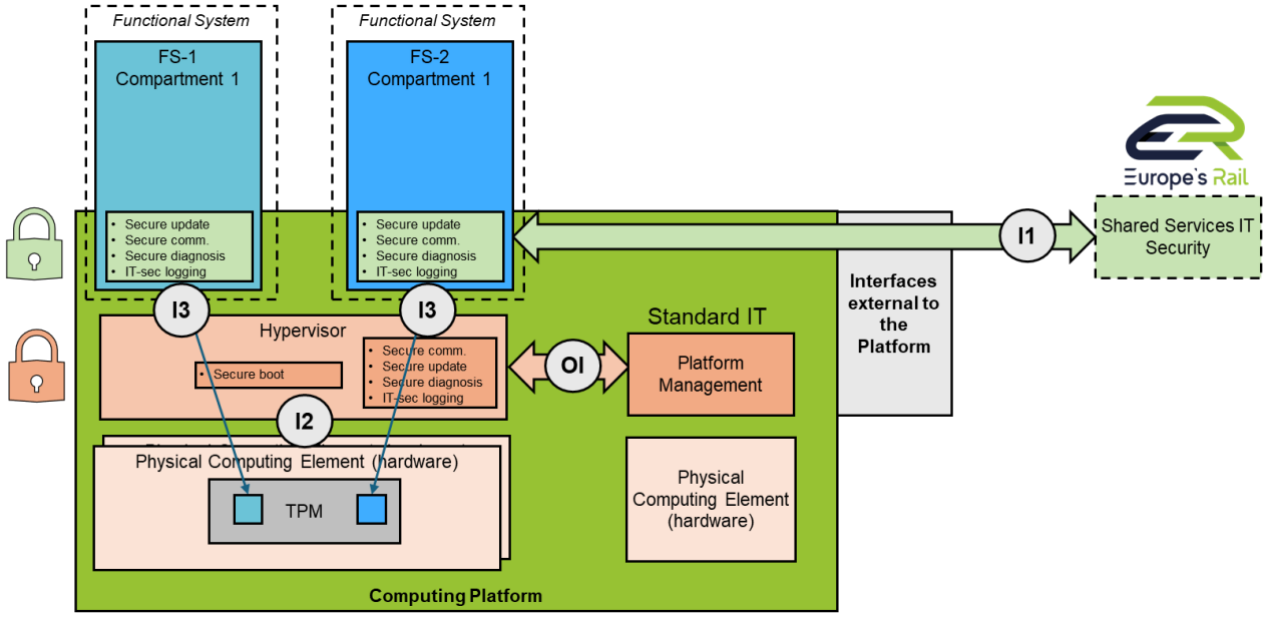


Figure 35: Security within FS Compartments

6.8.2 ERJU Security inside of the CEE

IT-security functionalities are running as addon-functionality native on host OS of the Hypervisor.

FS compartments are not directly affected by IT security, but integration of interface to IT security functionalities is necessary.

Here, workload for IT-security is not isolated by resource partitioning, cross-interference between FS compartments is possible (workload of one FS compartments affects performance of another FS compartment). Thus, this variant is not investigated further.

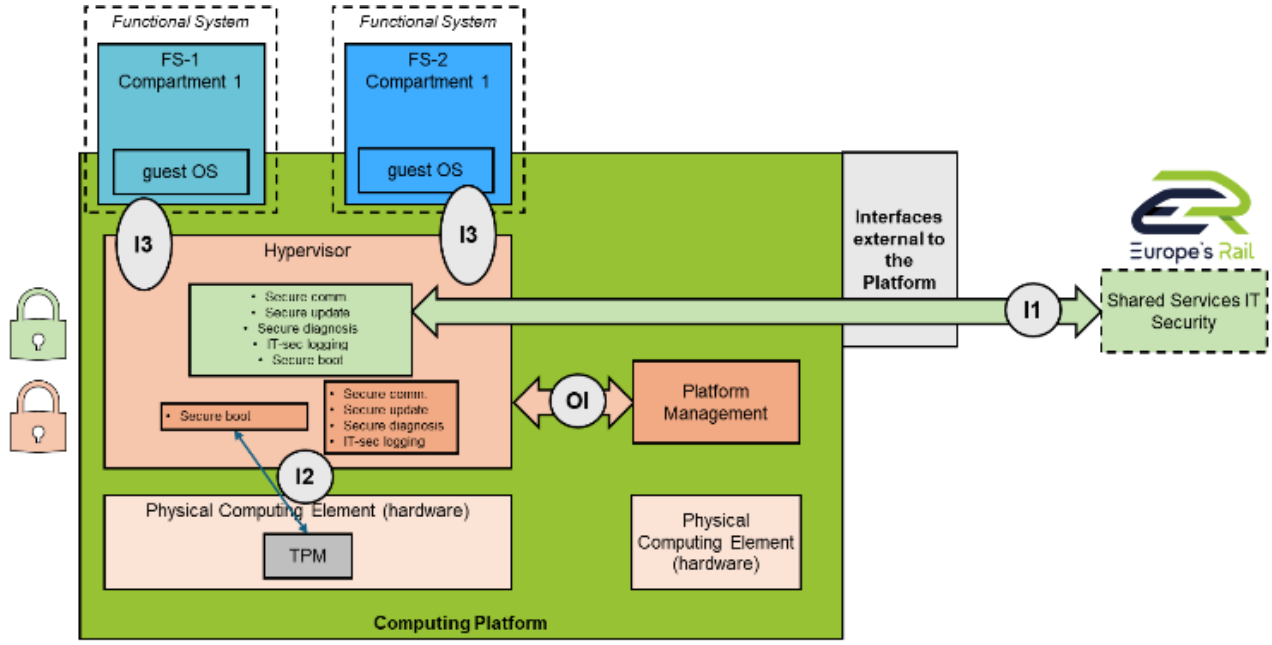


Figure 36: Security native in the CEE

6.8.3 ERJU Security in own VCE as "Soft Crypto Box"

IT-security functionalities are realized as own software running as own compartment in an own virtual machine with message-based communication interface I4 to the FS compartments. It's a kind of "soft crypto box".

Workload for IT-security is isolated by separated virtual machines (no cross-dependencies between compartments of different FS). FS compartments are not directly affected by IT security, but integration of I4 is necessary.

Internal communication between the compartments is not secured. This communication between the virtual machines goes via the hypervisor and it cannot be ensured that this is protected in needed way. Additional latency times are introduced by message-based communication between the functional compartment and the soft crypto box. Re-deploying is complicated because security zones have to be reconsidered. This would be hard to maintain, not exploiting flexibility of VCEs.

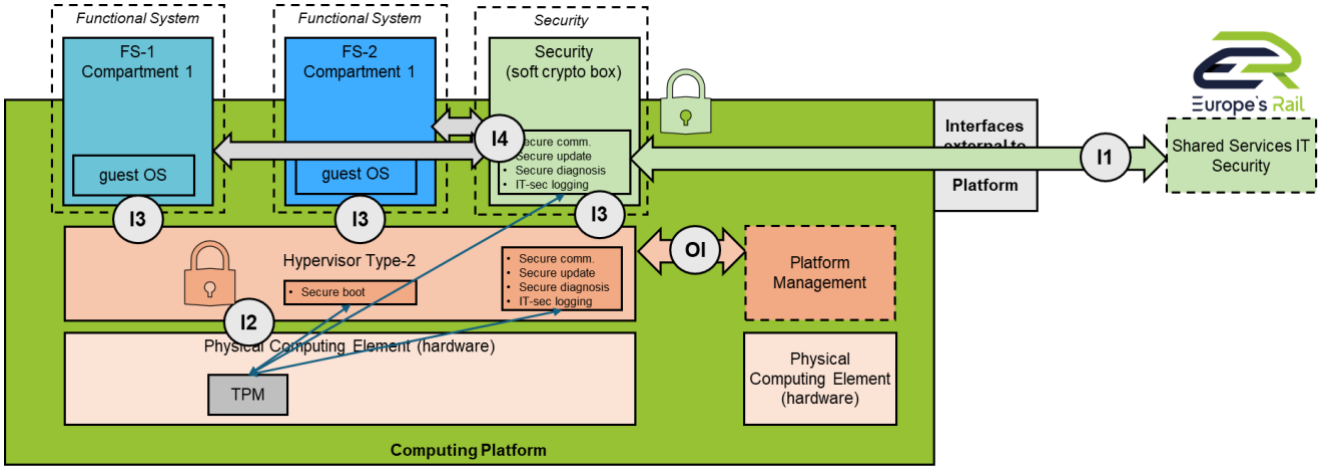


Figure 37: Security by "Soft Crypto Box"

6.8.4 Conclusion

From view of “best possible resource isolation” the ERJU defined IT security functionalities should be implemented for each FS compartment individually. From view of “responsibility for SL3 certification” each FS should be by itself responsible for IT security of own FS. This means that IT security has to he provided as part of the FS compartment.

Preferred solution is architecture with IT security provided by the FS compartments as described in **chapter 6.8.1.**

6.9 Availability of Functional Systems

6.9.1 FS Runtime behavior, reaction time and inter-communication

The basis for high availability of a safety related FS is a nearly perfect deterministic runtime behaviour, reaction time of the involved FS compartments with high-speed communication in-between.

The SE replicas of an up to SIL4 FS, which are running in different FS compartments on different (physical) computing elements, process a time-critical inter-synchronization between the SE replicas as e.g. for processing of a safe voting mechanism.

If an individual process step is not perfect (runtime behaviour, reaction time, inter-communication) the SE identifies this and reacts in a safe way as e.g. by stopping the affected SW parts. This leads to reduced FS availability and could potentially result in the stoppage of all application replicas of the FS.

The VE shall guarantee a stable runtime behaviour and reaction time of all SW parts within a safety critical FS Compartment for each timepoint, see **REQ-HLPI-14.**

6.9.2 Individual failures in hardware or software of the platform

Failures in the hardware or software of the VE lead to failures within the FS compartment(s). To address these failures effectively, redundancy mechanisms for the FS compartments are necessary.

In case of an up to SIL4 FS this redundancy is realized within the SE and depends on the safety principle (as e.g. 2oo3 or 2x2oo2).

In case of BIL FS, the backup functionalities of VE standard IT solutions may be usable as redundancy mechanism. It depends on detail of the concrete FS if such standard solutions can be used or not.

Requirement addressed to the SE of an up to SIL4 FS: The SE shall support the automated repair of a failed FS compartment, see **REQ-HLPI-15.**

6.9.3 Individual failures in communication

For availability of external communication to connected systems, each FS uses at least 2 redundant communication channels. Basically, each individual FS compartment can provide individual communication channels of a redundant communication. For flexible usage on the side of the FS compartment for automated repair of broken network communication channels its necessary to consider all FS compartments in the configuration and physical installation of the network.

The VE shall support the mapping of FS compartments to virtualized Ethernet adapters and to tie a virtualized Ethernet adapter to a specific physical ethernet card. See **REQ-HLPI-16.**

6.9.4 Availability in context of SW maintenance

SW maintenance of non-safe software as e.g. for IT security patching shall be possible during the operational phase of the FS for trackside use cases.

Example:

For an up to SIL4 FS with 2oo3 principle, the process for an IT security patch shall look like this:

- Update of the FS compartment 1 with new version for IT-security mechanism. For this the FS keeps running as 2oo2 (FS compartment 2 and 3) during the update phase and achieves 2oo3 mode again after synchronization of the updated FS compartment 1 with the other FS compartments 2 and 3.
- Repeat this for FS compartment 2.
- Repeat this for FS compartment 3.

Requirement addressed to the FS: update of the IT security components within the FS compartments shall be possible "VCE-wise one after the other", see **REQ-HLPI-18.**

Update of the VE shall be possible in same way "PCE-wise one after the other", see **REQ-HLPI-19.**

Requirement addressed to the Platform Management: The dependency to update "one after the other" must be considered, see **REQ-HLPI-20.**

> Open point: The architecture for updating FS by Shared Services and Platform Management has to be clarified. See Open-014.

6.9.5 Geographical redundancy

For trackside use cases of the Modular Platform, the aspect of "high grade of centralization" leads to increased availability requirements, such as geographical redundancy.

Geographical redundancy means that the FS is running distributed in different geographical locations to provide best possible availability in case of a disaster scenario (as e.g. blackout, terror attack, ...).

The main challenge in context of geographical redundancy is the handling of the split-brain problem for an up to SIL4 FS. See CAP theorem - Wikipedia:

When a *network partition* failure happens, it must be decided whether to do one of the following:

- cancel the operation and thus decrease the availability but ensure consistency
- proceed with the operation and thus provide availability but risk inconsistency. Note this doesn't necessarily mean that system is highly available to its users.

For up to SIL4 FS the consistency must be ensured, and this means that a network partition failure must be handled in a safe way. Additionally, the safe communication to connected systems as decentralized object controllers must be considered. Safety protocols as RaSTA require fully synchronized communication channels, which means the switch-over of a centralized interlocking

logic from one location to another geographical location is not possible without interruption of the safe communication to object controllers.

The safety related handling of split-brain topic must be solved on the functional level of the SE in the FS, even as e.g. for safety principle 2x2oo2 without geographical distribution. The VE does not know details about redundancy mechanism and handling of geographical redundancy by the up to SIL4 FS.

> Open point: the safety related architecture for georedundant FS with safe handling of split-brain problem is not yet defined, see **Open-015**.

6.10 SCALABILITY

The modular platform architecture shall allow to increase the number of FS Compartments running on the VE as long as the necessary physical HW resources are available.

> Open point: how to handle the scalable usage of CPU resources of the physical hardware (cores, memory, network cards) for flexible usage by independent FS compartments running on same PCE. See **Open-016**.

6.11 DIAGNOSIS

Each diagnosis data must be provided via the interface **I1 Diagnosis** to the Shared Services Diagnosis as central data sink. In this it will be necessary to transform the data within the Platform Management into I1 compatible format, see **REQ-HLPI-21.**

Maintenance activities to repair failures shall be automated as good as possible by the Platform Management. For this it's necessary to provide relevant diagnostic data to the Platform Management for root cause analysis and automated initiation of maintenance activities, see **REQ-HLPI-22.**

Relevant diagnostic data for the Platform Management is, without this list being exhaustive:

- state of the VE instance on the PCE
- state of each individual VCE
- state of the FS regarding availability of the individual FS compartments
- state of the network which is used for FS internal communication between FS compartments

> Open point: The details of the diagnosis architecture to process a root cause analysis and realize automated repairs are not yet clarified. See Open-017.

6.11.1 Diagnosis of the Functional Application (FA)

Diagnosis of the FA is related to logical states within the running application. This diagnosis data is provided by the FA itself and provided from the FS compartment via the interface I1 to the Shared Service Diagnosis. This diagnosis data does not have any relevance for the VE or the Platform Management.

6.11.2 Diagnosis of the FS

Each FS is responsible to provide diagnosis data about its own health state (e.g., about the health state of running SE replicas, FA replicas, …). The FS diagnosis data must be provided by the FS via **I1 Diagnosis** (as defined by TCCS) to the Shared Services as central data sink, see **REQ-HLPI-23.**

Additionally, the diagnosis data must be provided to the Platform Management as data sink for the handling of FS compartments, see **REQ-HLPI-24.**

A FS which is running in several parallel FS compartments provides diagnosis data by each FS compartment individually. This is based on redundancy principle for availability and additionally by individual diagnosis states in the individual FS compartments. It may happen that individual failures happen within individual compartments, as e.g. one FS compartment may have an individual SW failure inside.

The Platform Management must handle this relationship of the individual FS compartments of the FS, meaning that one FS provides diagnosis data of several FS compartments, see **REQ-ALPI-025.**

6.11.3 Diagnosis of the VE

Dedicated diagnosis about the VE and virtual computing elements must be provided from the VE to the Platform Management, see **REQ-HLPI-26.**

The Platform Management must forward this diagnosis data about the VE and VCEs via **I1 Diagnosis** to the Shared Services for diagnosis, see **REQ-HLPI-27.**

6.11.4 Diagnosis of the COTS Hardware

Diagnosis about the physical computing elements should be provided by a dedicated diagnosis software possibly provided as 3rd party software (as e.g. Prometheus (software) - Wikipedia, SevOne - Wikipedia) running within its own compartment independent from the rail FS, see **REQ-HLPI-28.**

This diagnosis data shall be provided to the Platform Management, see **REQ-HLPI-29.**

The Platform Management must forward this diagnosis data about the PCE via **I1 Diagnosis** to the Shared Services for diagnosis, see **REQ-HLPI-30.**

6.11.5 Diagnosis of the Network

Due to FS architectures with FS compartments running in parallel on different VCEs with a message-based communication in between, the topic of network diagnosis is relevant for the FS state.

The network diagnosis shall provide the diagnosis data to the Platform Management, see **REQ-HLPI-31.** The Platform Management must evaluate this data in context of “root cause analysis” for the related FS, see **REQ-HLPI-32.**

Open point: architecture for network diagnosis, see Open-018.

6.12 MAINTENANCE

6.12.1 System Maintenance

A modular handling of non-safe parts during runtime is necessary to enable the maintenance of systems if they are supposed to stay in operation with high availability requirements.

Relevant maintenance scenarios during FS runtime are for instance:

- IT-Security patch in FS
  FS compartment-wise "one-after-the-other", see **REQ-HLPI-33**
- IT-Security patch in VE
  hardware-wise "one-after-the-other", see **REQ-HLPI-34.**
- HW replacement
  of an individual physical computing element during runtime of the FS, see **REQ-HLPI-35**

Maintenance activities for **safety related software parts** are not easily possible during runtime of the up to SIL4 FS due to safety related dependencies between the parallel running safety related replicas.

It's state-of-the-art for safety approvals that the safety related software within the individual compartments of an up to SIL4 FS belong to the same version. By this an exchange of individual safety related SW must be done in all involved compartments and this means a FS stop in between = stop of running safety related software (with old version) and start of new safety related software (with new version).

> Open point: architecture and process for installation of SW on new hardware, see Open-020.

6.13 AUTOMATED REPAIRS

Maintenance activities for the repair of failures shall be automated as much as possible by the Platform Management.

Example: Automated repair of a failed VCE (used by an up to SIL4 FS with 2oo3 principle)

In case of a failure of an individual VCE this failure is

- directly identified by the VE, VE provides diagnosis data to the Platform Management.
- indirectly identified by the SE of the belonging FS, the FS running mode is reduced from 2oo3 to 2oo2 (because one FS compartment has failed). FS provides diagnosis data to the Platform Management.

The Platform Management must process this diagnosis data to

- identify the failed VCE as root cause
- initiate the repair (new start) of the VCE automatically

If such a repair (new start) of the VCE is not successful, the belonging FS compartment must be installed in a newly created VCE. The newly created VCE may be on the same PCE or on another PCE.

In case of VCE creation on another PCE the aspect „FS compartments of an up to SIL4 FS must not be installed on same PCE“ must be considered from view availability to avoid a safe reaction by the SE which identifies an unallowed SW installation of FS compartments on same hardware.

> Open point: such an automated installation of safety critical FS compartments by a basic integrity Platform Management must be evaluated from view of safety. See Open-021.

6.13.1 Lifecycle management for the VE

By usage of existing standard IT solutions as VE, a lifecycle management for the usage in context of rails systems is necessary. Each new version of the VE must be qualified for usage in context of rail systems.

> Open point: What exactly is necessary in context hardening of the VE? What kind of VE functionalities must be deactivated or even removed to ensure that the handling of rail systems running on VE is possible in way as needed (efficient handling and available running FS)?, see **Open-022**.

The VE shall provide backwards compatibility of the VE configuration interface for FS configuration. FS related VE configuration of old version of VE shall not be affected in context of FS migration onto new version of the VE. See **REQ-HLPI-17.**

> Open point: Is a kind of “generic” testing possible for performance and runtime behaviour of a new VE version to avoid the need for integration of each individual FS compartment version with a new VE Version? See Open-023.

A new version of VE shall not have an impact on safety but may have an impact on availability.

6.13.2 Spare handling of COTS Hardware

For the usage of COTS hardware spares it has to be considered that HW providers usually do not guarantee that newly ordered hardware is 100% identical to previous deliveries of the same type.

The detailed HW related dependencies between the COTS hardware and the VE at the interface I2 must be identified and “managed” in context of maintenance, for this see chapter 6.6.

6.14 Public Cloud

With “public cloud” we mean computing services offered by third-party providers over the public Internet, making them available to anyone who wants to purchase them. Computing services are sold on-demand, allowing customers to pay only per usage for the CPU cycles, storage, or bandwidth

they consume. The cloud provider is responsible for the management and maintenance of the services. Access is done via the public internet.

Examples:

- Amazon Web Services (AWS)
- Google Compute Engine (GCE)
- IBM Cloud
- Microsoft Azure

We do not see “public cloud” as a realistic use-case for the moment and within the next 5 years for safety-related applications. A Data Center (“Private Cloud”) with defined overall responsibility will be possible. Basic challenges for “public cloud” are described in the following chapters.

6.14.1 Safety architecture

Basically, FS compartments of rails systems could technically run in a public cloud. Even safety critical FS Compartments could run in a public cloud under the condition that the needed information about the physical hardware can be provided in needed reliable way, see chapter 6.8.

For the FS compartments itself it should not make a difference if the virtual machine is in a private cloud or public cloud. Even protocol drivers for safety relevant communication protocols as RaSTA can run in a public cloud due to the safety architecture “underlying VE is not safety relevant”.

The topic of “native hardware access” (NHA) is not solved for public cloud.

6.14.2 Security architecture

Handling of public cloud as a secure device in context of rail infrastructure is not solved.

Public cloud providers typically do not allow to install own IT security solutions.

6.14.3 Performance, reaction time and availability

The fulfilment of requirements of especially safety-relevant FS compartments to the underlying software from view of **performance, reaction time and reliability** in the runtime behaviour for each timepoint over long periods of several years is not yet experienced in the context of public clouds. Such requirements (performance, reaction time) do not affect the safety but the availability of the FS.

Example:
If a SW component does not wake up and react in the required time-range of 20-50ms this misbehaviour would be identified by the SE and would lead to safe reactions and perhaps to a stop of the safety critical FS compartment(s).

6.14.4 Integration and maintenance

Public cloud providers are not able to integrate rail systems with an updated version of the public cloud. Handling of changes in the public cloud in context of integration with the rail systems is not solved.

6.14.5 Business Case

Comparison of costs for running rail systems over decades within a public cloud with costs for private cloud has not been done.

6.14.6 Responsibility

How to handle the situation that changes or instabilities of the public cloud lead to reactions on side of the rail systems and in consequence to impact onto the rail operation (e.g. as stopping of trains due to safe reaction of failed FS compartments)?

6.15 CERTIFICATION

In context of certification the main goal is to define the safety- and security-architecture of the different layers within the computing platform in such a way that changes in non-safe parts like the COTS based hardware or in common SW layers like the virtualization can be handled without impact onto safety- and security-certification of the Functional Systems running above. This is in the scope of the SP PRAMS domain that is currently working on a document “Evolution Management of safety related systems” [17]. Further findings and recommendations from study on modular certification and homologation can as well be expected in the upcoming R2DATO Deliverable D26.4 [20].

6.16 CONCLUSION AND OUTLOOK

The situation “various Functional Systems running aggregated on same computing element” has a lot of new challenges affecting the architecture, interfaces and processes which have not been addressed so far by standardisation as, e.g., EULYNX, which is up to now focused only on the trackside object controller.

The basic results of our investigations show that essential architecture aspects depend on the specific solutions for SE and VE:

1) The grade of HW independence is SE solution specific and depends on the safety concept of the SE. Different HW related SRACs of different SE solutions require different NHA variants with even specific direct dependencies to the interface I2 to the physical hardware and interface I3 extension to the SE running above I3.
A generic HW independent solution of NHA is not possible for different SE solutions due to **SE solution specific dependencies at the interfaces I2 and I3 extension.**

2) The rules for handling of FS compartments in context of installation and update on side of the Platform Management depend on the **specific SE solution** (as e.g. 2oo3, 2x2oo2, ..).

3) The detailed solution for the SW orchestration on side of the Platform Management depends on the **specific VE solution.**

4) The technical SW architecture for realization of NHA depends on the **specific VE solution** and belonging Host OS.

5) The details for clarification of the overall architecture for IT-security according to IEC 62443 depend on the **specific VE solution.**

6) The technical quality of guaranteed resource isolation and performance depends on the **specific VE solution** and must be proven individually for each **specific SE solution**.

Summary:

The depicted architecture and principles of interfaces I2 and I3 are generally independent of the specific virtualization environment and specific hardware and safety layer implementations (as long as certain requirements are met as detailed in Section HLPI Requirements). However, specific safety layer realizations may impose specific requirements on the NHA. Hence there will possibly not be one single NHA realization. Further studies are needed to explore how the NHA requirements for different safety solutions can be harmonized or potentially omitted altogether.

If several different SE solutions shall run on the same VE solution, it's necessary to handle the SE solution specific aspects in context of "HW related SRACS to fulfill by specific NHA functionality" and HW dependency individually for each SE solution.

The figure below shows the specifics in the architecture according to interfaces I2, I3 and to the Platform Management.

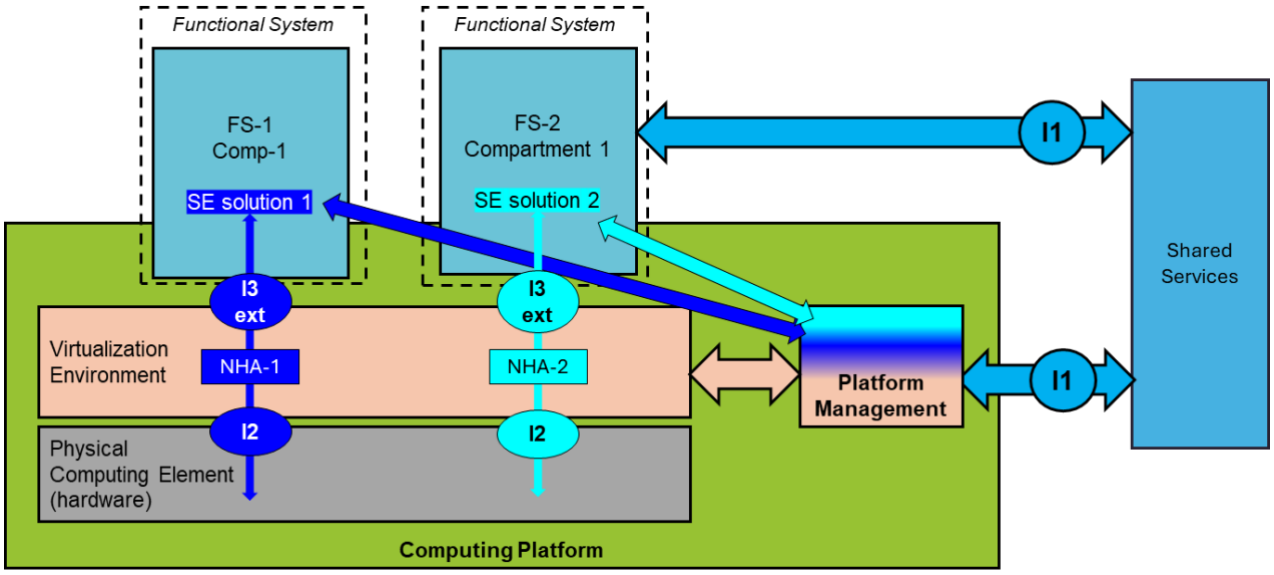


Figure 38: Specifics in architecture

Even if the interfaces I2 and I3 depend in detail on specifics of the SE solution it's necessary and possible to define the functional overall architecture for the Shared Services, Platform Management and Functional Systems.

And the topic of "best quality in resource isolation" needs practical investigation to identify feasible solutions for VE.

7 APPLICATION-LEVEL PLATFORM INDEPENDENCE (ALPI)

7.1 INTRODUCTION

The main objective of the ALPI Interface, used in the MPC modular platform, is to simplify the design of a generic "functional application" that realizes a Business Logic in the railway sector.

The purpose of the interface is to allow application engineers to focus on solving problems closely related to Railway Functional Applications, delegating part of the solutions to problems concerning safety, security, HW/SW integration, deployment, and maintenance to the underlying layers of the MPC architecture. The interface also aims to allow the use of standard COTS solutions as much as possible, to facilitate development of Business Logic by different suppliers.

The proposed ALPI interface shall minimize the impact on the development of runtime services, providing the characterization of the services through configuration values (see REQ-ALPI-016), that define the characteristics of safety, security, and management functionalities for functional application SW orchestration. This simplifies the interface on the functional application user side, hiding the complexity of the solutions adopted in the MPC architecture to ensure the required levels of safety and security (see REQ-ALPI-04).

The ALPI interface makes it possible to use different MPC platforms from different vendors, enabling extensive use of standard IT solutions. reuse of COTS products, adoption of innovative solutions provided by the market, especially in relation to developments in the fields of Operational Technologies (OT), Information and Communication Technologies (ICT), Safety and Security Management.

This chapter deals with basic assumptions, preconditions, and cornerstones necessary to build and specify an API used by applications with safety relevance from CENELEC Basic Integrity up to SIL4.

It reflects possibilities and ideas but does not intend to head towards a specific direction, as this shall be left to the vendors of the API. However, the suggest approach is to promote the adoption of a common high level API set, providing the same functionalities, eventually with different semantics harmonized with the introduction of adapters.

The discussions presented in this chapter are based on the inputs from the SP CE domain (chapter 3.7), the architecture proposal in WP26's first deliverable [18] and the ongoing work in WP26 itself.

The following description includes aspects related to the "programming interfaces" and related to the definitions of needed functionalities from the view of a Functional Application. The descriptions cover both runtime services and off-line configuration.

With reference to safety, the descriptions refer to the I4 interface in the case of Functional Application with Basic Integrity Level, while they refer to the I5 interface in the case of safety-related functions. It is assumed that all I4 services are provided by the RTE, so I4 coincides with RTE. The Safety Related I5 interface contains the restricted subset of I4 services compliant to CENELEC rules and additional safety services provided by the Safety Layer, needed to tranparently manage safety requirements.

The ALPI I5 interface cointains the transparent safety management of the replicas when needed. The off-line configuration allows the proper selection of the safety services. The configuration knows the nature of the FA: if it requires safety services or not, OR if it requires I5 or not.

7.2 CORNERSTONES OF ALPI

To achieve platform independence on application level, the API serving the application must provide all necessary safety-related and non-safety-related interfaces and resources for fulfilling the application functions including diagnosis, logging, and monitoring.

Another basic aspect for platform independence is, that certain architectural principles are shared. Otherwise, standardisation would be hard, not to say impossible. As a result of this, also the behaviour on the other platform interfaces must be specified with respect to the goal since this is a necessary pre-requisite for full interoperability.

Furthermore, Platform independence requires a standardised language to specify the application's deployment-configuration in a platform agnostic way. During integration of the application with a specific platform, the platform-agnostic application deployment-configuration is then translated/converted into a corresponding platform specific application deployment-configuration.

The process related to the deployment and configuration should be managed using a "model driven" approach. The model must provide the rules and procedures for the aggregation of the elementary basic elements necessary for the incremental construction of the subsystem layers. For each level, the minimum constituent elements, i.e. the "basic components" are identified and their use and integration is described via data structures. Note that, this translation or conversion is not an easy, automatic task as it needs to deal with technical functionality as well as with safety principles and fulfilment. Common standard tools must be used during the process to configure components and their aggregation.

7.2.1 Main principles followed for the ALPI's definition

The three guidelines that are adopted in this document are listed here:

- The implementation of Business Logic should be as **easy** as possible.
- The implementation of Business Logic should be as **standard** as possible (see REQ-ALPI-02).
- The Business Logic should be easily integrated and reused.

7.2.1.1 Functional Interface goals

These goals are achieved through a **functional interface** that ensures:

a. Independence: ALPI shall allow developing an application regardless of the SW/HW of the underlying layers (see REQ-ALPI-01)

b. Transparency: Transparency of the complexities related to safety and security architectures/mechanisms (see REQ-ALPI-06)

c. Compatibility and Portability: ALPI shall allow running the same application on different commercial of-the-shelf platforms, using standard HW/SW basic components (see REQ-ALPI-05)

d. Flexibility: maximum flexibility shall be allowed in case of Non-Safety Related Functional Application to take full advantage of the evolution of ICT and OT technologies. Constraints that limit the use of products with new technologies developed in the COTS environment should be avoided; however, solutions/products that favour a high level of

standardisation in the development of BL should be recommended. For Safety Related Functional Application restrictions are to be considered because of the compliance to safety standard (see REQ-ALPI-03).

7.2.2 Previous Work as discussed in D26.1

With reference to section 2.3, several pre-existing documents expressing previous thoughts on the topic influenced this document, especially these documents:

- PI-API DB/Thales/Sysgo/Fraunhofer/... [3]
- PI-API DB/SMO [4]
- OCORA papers [5], [6], [7], [8], [9], [10], [11], [12]

Note that the referenced documents were taken as an inspiration, and not as an ultimate truth.

7.3 Structure Overview

Figure 39: High Level Process of Application Development

Chapter 7.3 tackles the endeavour in 3 different stages:

- "Common Basic Assumption" (see chapter 7.3.1)
- "Application-Level Platform Components" (see chapter 7.3.2)
- "Set of Deliverables for the Integrator" (see chapter 7.3.3)

Chapter 7.4 contains a description of the models to be adopted and the evaluations of the impact on the ALPI interface regarding safety, security, maintainability, orchestration.

7.3.1 Common Basic Assumptions

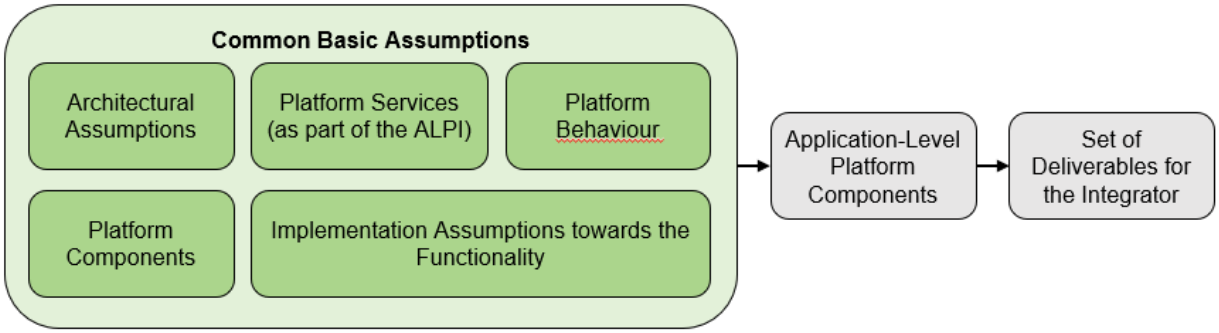


Figure 40: Common Basic Assumptions Overview

This paragraph summarizes basic and agreed assumptions. As a goal, a general direction of the result should be imaginable after reading this section. For each assumption an impact evaluation on ALPI and on Functional Application is carried out in advance.

7.3.1.1 Architectural Assumptions

Following an API usually also means to adhere to a certain architecture or, at least, to some basic architectural concepts. This paragraph tries to summaries exactly those concepts in a high-level description.

The main goal of ALPI is the standardization of the development of a Business Logic. It is assumed that a Business Logic consists in the implementation of one or more Functional Applications based on services provided by ALPI.

1. Each Functional Application consists of one or more “processes" as defined in the glossary.(see 3.7.2 and REQ-ALPI-022, REQ-ALPI-025, REQ-ALPI-026).
2. The platform can be assessed against relevant standards (as at least CENELEC EN50126 / EN50716 / EN50129 and additional ones as EN50155 for on-board). These standards influence architectural decisions for the modular platform concept. Functional Application and ALPI must be compliant to relevant CENELEC standards. (see REQ-ALPI-024)
3. Redundancy for safety is supported, depending on and controlled via configuration. ALPI I5 provides services that allow a Safety Related Functional Application to transparently manage redundancy for safety.
4. Redundancy for availability is supported, depending on and controlled via configuration. ALPI I4 provides services that allow a Functional Application to manage redundancy for availability
5. Communication is message-based. ALPI provides services that enable communication in a single, standardized, common way.
6. Run-To-Completion scheme within a (sub)process. ALPI provides services that allow the Functional Application to adopt the RTC scheme. (see REQ-ALPI-027)
7. Replica deterministic behaviour. Deterministic behaviour of a Functional Application is transparently assured inside ALPI services, additional services allow a Functional Application to do checks.
8. Clear application lifecycle, minimum: Start/Init → Operate → Stop/Shutdown. ALPI provides services that allow the Functiona Application to adopt the lifecycle scheme.

7.3.1.2 Platform Components

Platform components are the basic components, tools and libraries needed to implement, test and run applications, such as:

- Toolchain (e.g. validated compiler, linker, diagnosis, tracing) (see REQ-ALPI-029)
- System Libraries (e.g. glibc, crypto-libs)
- Coverage- and Test-Tools

These components are provided by the Run Time Environment and may be specific, depending on the RTE. ALPI provides a standard “model driven” description that allows the integration of the Functional Application with the needed ingredients in a common standard way.

7.3.1.3 Platform Services

The platform services discussed here are to be seen as part of the services provided to an application by the “Independence API”. These include but are not limited to:

- Logging (e.g. syslog) (see REQ-ALPI-019, REQ-ALPI-036)
- Process control (e.g. create, clone, delete)
- Memory management (e.g. malloc, free)
- Timing (e.g. time of day, sleep, different clocks) (see REQ-ALPI-030)
- Communication (e.g. sockets, message queues)
- Maintenance-related diagnostics and errorcodes (see REQ-ALPI-038)
- Security functions (see REQ-ALPI-014)
- Persitency functions

Note that parts of these services could be safety relevant. For the correct usage of these services both in the BIL and SIL cases, ALPI will allow to specify the Safety Integrity Level to be used through the configuration, keeping the runtime interfaces as unchanged as possible,.

Due to the trade-off between the maximum flexibility for non-Safety Related Functional Application and the constraints imposed by the CENELEC standard, ALPI will provide the Safety Related Functional Application with a restricted subset of the available RTE services.

7.3.1.4 Functionality Implementation Assumptions

In this section, assumptions are proposed on the functionalities that ALPI must provide for the development and implementation of an application. These functionalities must also take into account any constraints deriving from the use of the modular platform.

The assumptions considered concern architecture, life cycle, communications, diagnosis.

ALPI makes assumptions regarding how Applications will work and to how they are structured and may favor certain implementation concepts that doesn’t relate to MPC.

For example, the usage of a gateway concept for the implementation of safe protocols and specialized data services for diagnostics are handled here.

Note that these kinds of assumptions and rules need to be well defined within the user documentation.

It is assumed that ALPI will provide a Functional Application with the services based on the following functionalities provided by the modular platform:

- Configuration for runtime behavior with reference to execution environment, Redundancy Safety Integrity Level, Security Integrity Level, Communication.
- Lifecycle: Start/Operate/Stop phases. (see REQ-ALPI-031)

- Non-Safety Related Communication (i.e. OPC/UA or SNMP communication).(see REQ-ALPI-032)
- Safety-Related Communication (with different protocol implementations).
- Logging scheme.
- Diagnosis.

7.3.1.5 Platform Behaviour

The behaviour of the modular platform needs to be well defined within the user documentation. Even if the implementation details do not need to be known, basic behaviour and maybe also limitations need to be known from the application developer.

ALPI will provide a Functional Application with services that allow the characterization of the platform behaviour in relation to:

- Replica management
- Communication synchronization
- Timing/synchronisation rounds
- Platform health indication and management data available
- Logging/diagnosis environment & behaviour
- External interfaces (see also chapter 8)

7.3.2 Application-Level Platform Components

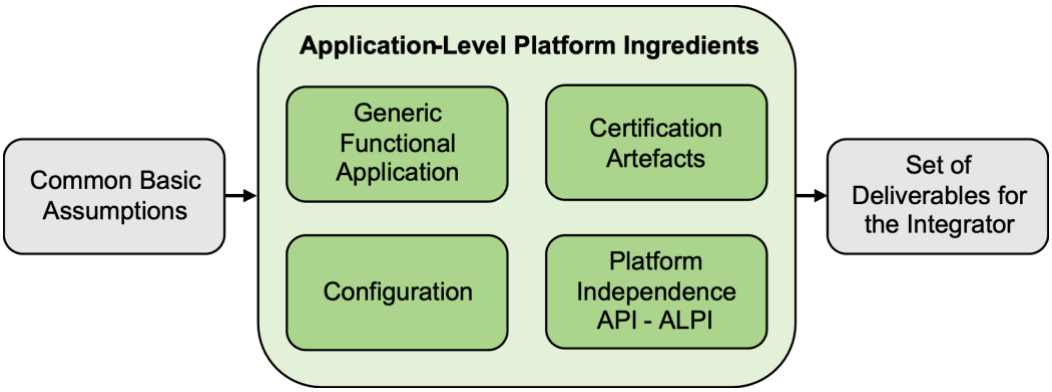


Figure 41: Application-Level Platform Ingredients Overview

An application consists of different parts, which are drafted in Figure 41 above.

The main parts are:

- Generic Functional Application
- Configuration
- Certification Artefacts

Note that this description shall be explanatory only, a real realisation of such a platform can also add other parts, if needed.

7.3.2.1 Generic Functional Application

The Generic Functional Application implements a certain Business Logic (BL) into a piece of software, that might be accompanied by SRACs (from RTE) (see REQ-ALPI-020), if necessary. "Generic" refers to the fact, that the goal is to enable development of different applications independent of a concrete execution platform (see REQ-ALPI-01). Additionally, the same application (at least source code) could also be used on platforms of different vendors if this is beneficial and necessary.

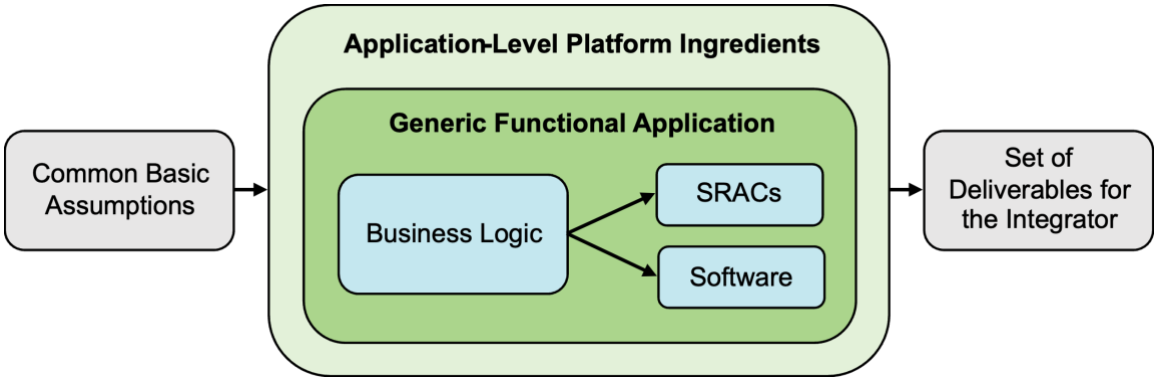


Figure 42: Generic Functional Application Overview

As a general concept, the platform shall allow to run applications with different levels of criticality in parallel. This so-called "mixed criticality" approach decouples the lifecycle of applications of different SIL levels from each other and shall ease the process of changing, especially for basic integrity applications (see REQ-ALPI-013). Of course, this goal can only be reached if the safety solution supports it.

Note: changing of functionality and using same functions on a different RTE without doing anything regarding safety assessment is currently not possible and needs further study.

7.3.2.1.1 Business Logic software – ALPI services

A Business Logic SW is composed of one or more specific Functional Application SW (FA), located and executed in HW independent Functional System Compartments.

ALPI provides each Functional Application with services to communicate with

- external entities via a group of interfaces referred to as "I1". I1 contains the "OI" interface that is used to manage (monitor, control, diagnose, configure) the computing environments.
- other remote/local Functional Systems to manage safe/non-safe functional application data necessary to realize an application. This communication-based interface is referred as I0.

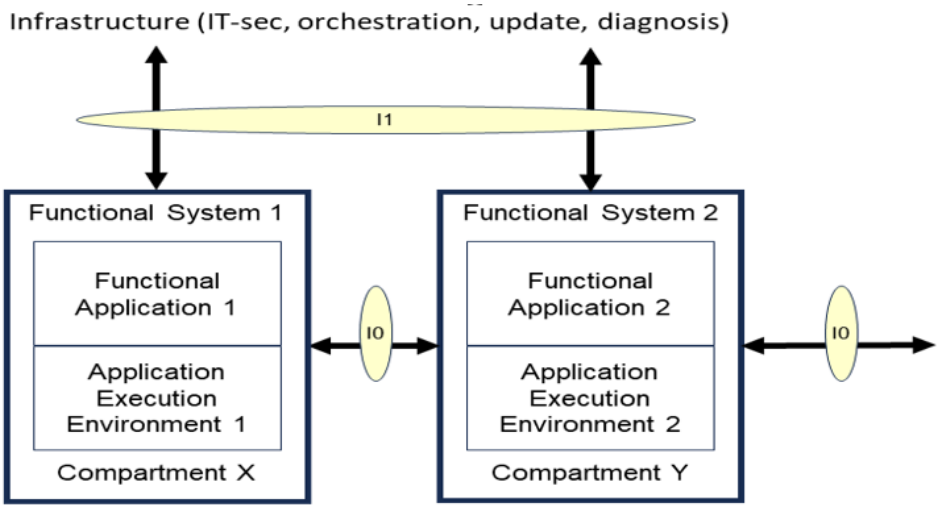


Figure 43: Functional Application Interactions

It is to be noted that ALPI shall standardize the communication interface, providing basic standard functions that allow communication with both external entities, local/remote systems using always the same common services. The type of the runtime communication is selected via a configuration specification.

A Functional Application SW consist of basic components that implement a Functional Application Task (FAT) (see REQ-ALPI-07). FATs use ALPI's services for communication.

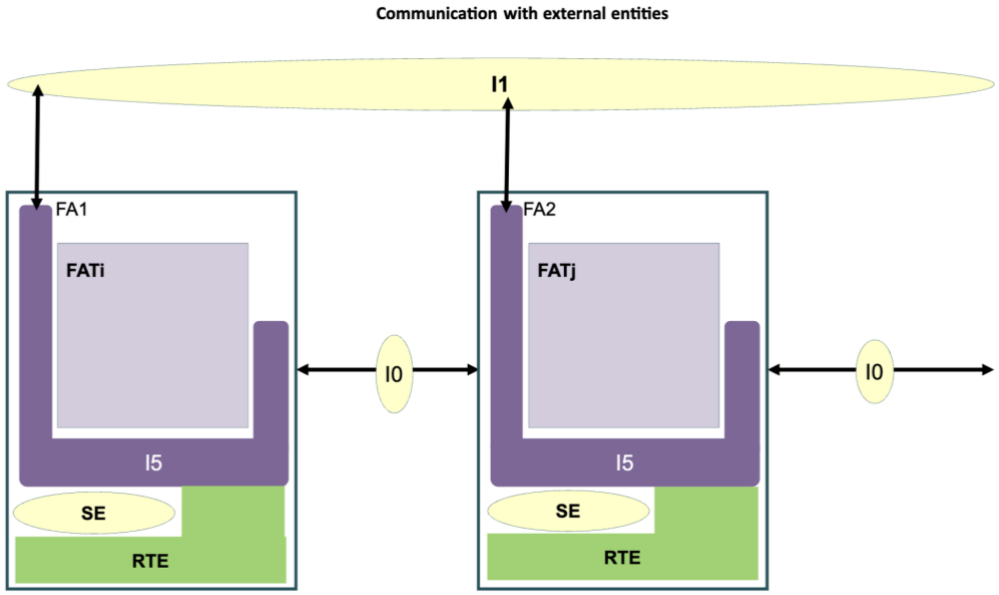


Figure 44: Functional Application Task interactions

ALPI provide each Functional Application Task also with all RTE services necessary for task management and transparently interacts with SE in case of Safety Related Functional Application.

ALPI services are available as Function Call and as Configuration data.

Depending on the type of the services, they can be classified into categories.

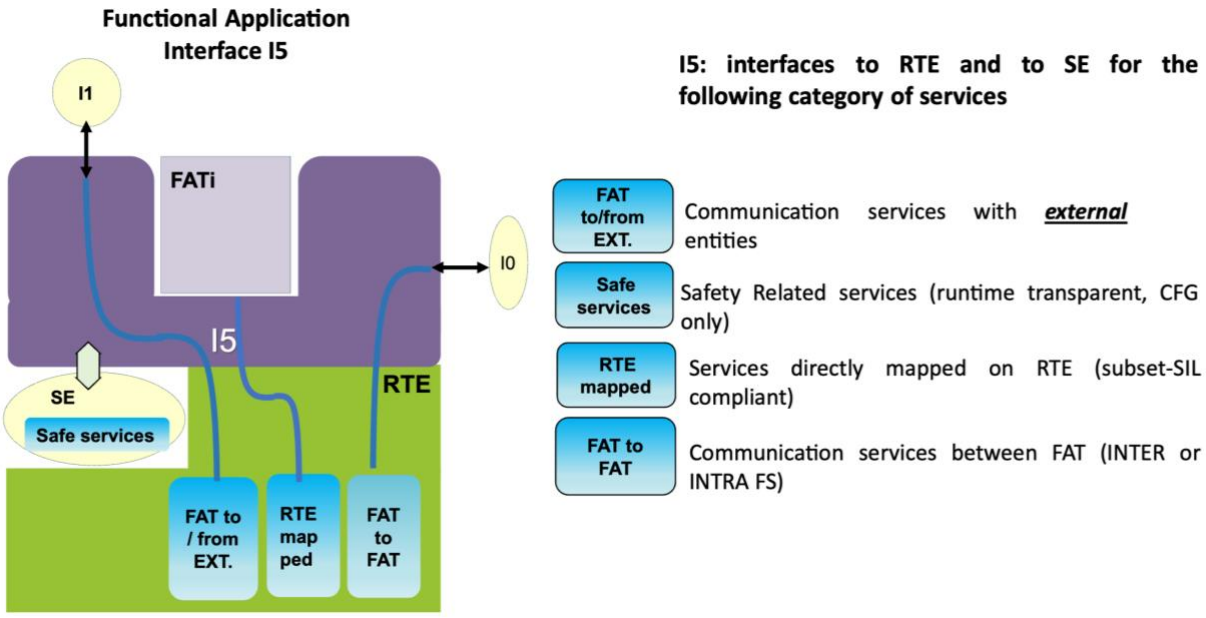


Figure 45: ALPI categories of services

7.3.2.2 Configuration

Configuration data is consisting of at least two fundamental parts:

- engineering data: needed for the business logic to work. Often the engineering data itself is again partitioned into
  - market/customer specific data
  - into market/customer specific data and product generic data
- RTE configuration data for software execution (see REQ-ALPI-033).

For all RTE specific configuration a strong requirement would be, that all platforms from different vendors share a common RTE syntax and semantic, so that running the same application on different platforms is as easy as possible.

Examples for engineering data configuration are:

- Data Base of IXL rules to be used/checked during BL runtime execution
- Logical endpoints that identify all (external/internal) entities that are involved in the command/control process implemented by a FA
- safety mechanism (integrity check, version control/congruency) to ensure that the BL and the safe system is consistent

Examples for RTE specific configuration are:

- Scheduling, execution budget
- Communication endpoint configuration
- Communication protocol details
- Redundancy configuration
- Voting algorithms

- Security algorithms
- …

7.3.2.3 Certification Artefacts

For a successful certification, a set of artefacts is needed:

- RTE artefacts
  - RTE SRACs, the Safety Related Application Conditions
    - Need to be fulfilled by the application, or “transferred” to the customer (see REQ-ALPI-020, REQ-ALPI-021)
  - RTE Security Conditions
    - Have to be defined and implemented from the RTE
    - But need to be fulfilled by the application (according to definition) to achieve a certain security level assessment
  - RTE certification + Safety case
    - On system level, so that it can be treated as “black-box” from the point of view of the Functional Application
  - RTE rules
    - Describes details about what to do and not to do on application level
- Application artefacts
  - An application specific safety case
  - Proof of application to follow the “RTE rules”
  - Proof of application to follow the SRACS coming with the RTE
  - Proof of application to follow the Security conditions

7.3.2.4 Platform Independence API - ALPI Interface

The ALPI interface (Application-level Platform Independence, previously: PI API – Platform Independent Application Programming Interface) is the concrete implementation how the Functional Application utilizes the underlying platform. It contains syntax and semantic details for each and every purpose needed from the application.

As outlined in the chapters above, more than only this ALPI is needed to realize the goal of portable applications, such as architectural preconditions and concrete platform behaviour. All that kind of necessities, including ALPI, need to be described in detail within the user documentation. For all safety relevant parts, additionally so-called SRACs (Safety Related Application Conditions) need to be clearly defined and explained.

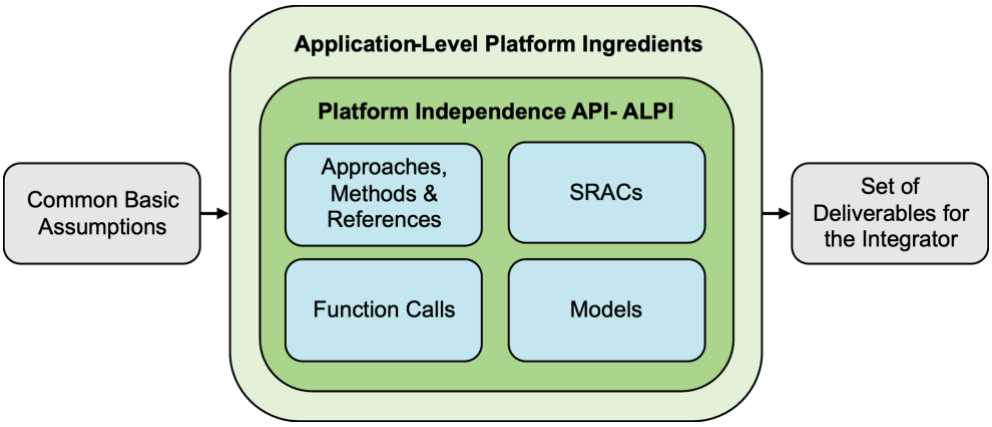


Figure 46: ALPI (PI API) Overview

In order to have a consistent top-level description, ALPI shall provide a detailed definition of the following topics:

- SRACs in different contexts, e.g.
  - towards the RTE
  - towards the HW
  - towards the application developer
  - *Note that it is very likely, that different platform variants come with additional, specific SRACS (hopefully, just a few...)*
- Models (see REQ-ALPI-017)
  - Programming Model
  - Communication Model
  - Configuration Model
  - Security Model
  - Maintenance/Diagnosis Model
- Approaches, Methods and References
  - Testing and Integration Approach
  - Testing Suites, e.g.
    - generic reference for a modular platform
    - ALPI/“PI API” test suite
- Function Calls (syntax and semantics, variants for different SIL-targets)
  - Memory management
  - Process management & lifecycle
  - Timing
  - communication
  - diagnostics

- HW IO (for embedded HW, e.g. on-board)
- Persitence and security functions
- ...

7.3.3 Set of Deliverables for Integrator

After the development and testing is finished, a set of deliverables is bundled for the integrator. This set is suitable to enable the integrator to enable the application on a modular platform instance.

- Application
  - Depending on the delivery model, in source-or binary form
- Configuration Data
  - Engineering Data
  - Application-specific RTE configuration data
- Certification Artefacts
  - From RTE vendor and from application vendor
- Integration of application with modular platform components
  - Depending on the concrete case: RTE, OS, HW, Virtualization, etc.
  - Test specs/cases to perform integration wherever possible

7.4 ALPI Details

Based on what is reported in the chapter 7.3, this section provides further details about the architecture, layers, and models used to describe the ALPI interface. Descriptions are based on the inputs from the SP CE domain (chapter 3.7), the architecture proposal in WP26's deliverable [18], [19] and the ongoing work in WP26 itself.

7.4.1 Assumptions

Application developers should be able to focus on implementing the application logic. All safety and fault tolerance mechanisms not inherent to the application's logic – specifically redundancy, voting and persistence – shall be implemented in, and transparently handled by the platform.

The application-level platform independence (ALPI) interface provides the standardised abstraction of all platform's specific hardware and software – allowing for portable applications.

ALPI should also allow the aggregation and the interaction of Functional Application with different Safety Integrity Level into a single Functional System.

7.4.2 ALPI architecture and layers

From a functional point of view, ALPI interfaces I4 and I5 are part of RTE and SE respectively.

ALPI provides the I4 interface in the case of Basic Integrity Level FA. (see REQ-ALPI-011)

ALPI provides the I5 interface in the case of Safety Integrity Level FA that require the use of a Safety Environment that ensures SIL using composite safety mechanisms. (see REQ-ALPI-012)

ALPI also, through I4 and I5, allows:

- the interaction of FAs with the communication system, via platform, towards the I0[6] interface that manages communication with other FAs and external systems
- the interaction of FAs with the I1 interface and the entities outside the platform, for the management of activities related to orchestration, diagnosis, security and time, update. These activities are mainly implemented in RTE layer and are transparent to the ALPI runtime interface.

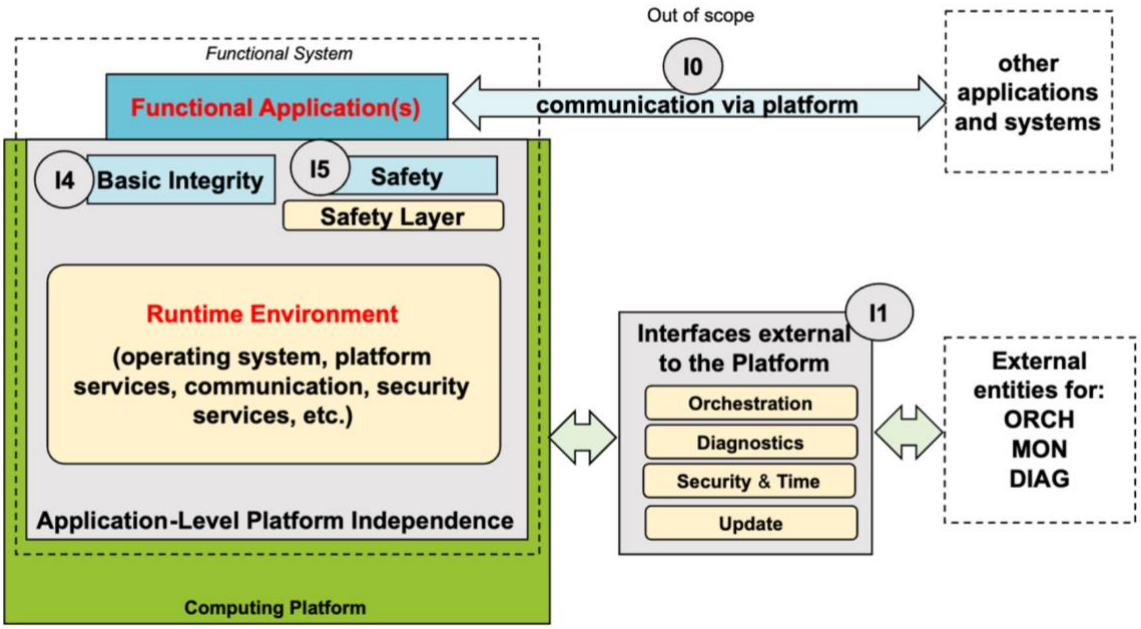


Figure 47: Functional Application, ALPI, RTE

7.4.3 Generic Functional Application

Functional Applications realize a Business Logic. They are part of a Functional System and are executed on the Runtime Environment inside a computing platform. FA use runtime services provided by ALPI interfaces I4 and I5.

[6] I0 is not in scope of this deliverable

7.4.4 Interface I4 and RTE

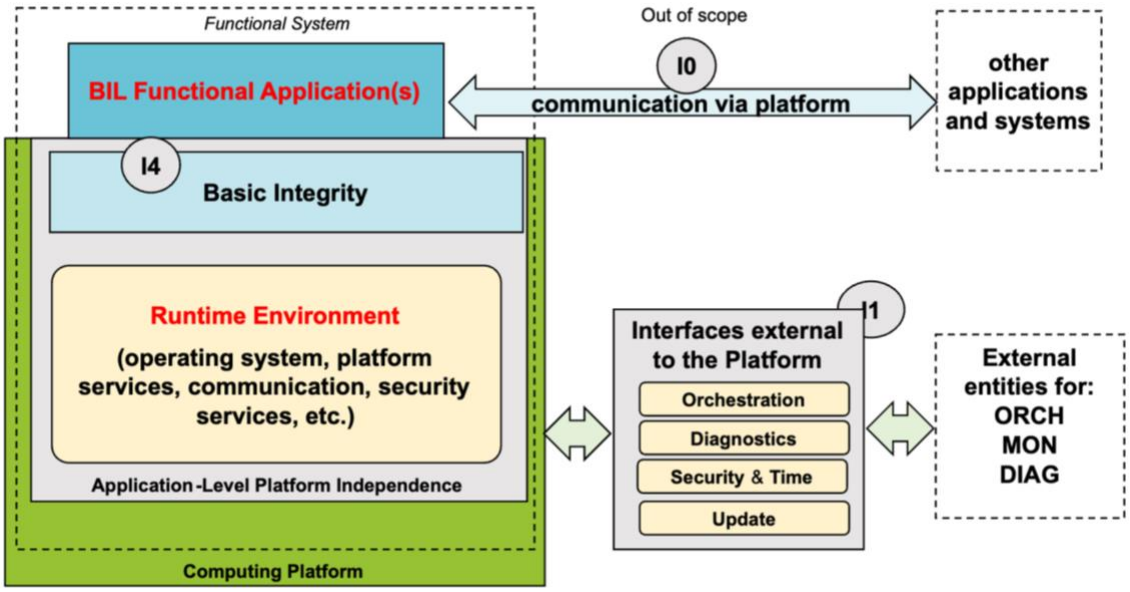


Figure 48: Interface I4

I4 goals:

- I4 shall standardize the access to RTE services, allowing FA developers a simpler and common way to develop FAs.
- I4 is a “runtime” interface, and it is assumed not to have “deployment interfaces”. The basic deployment of an FS, and therefor a FA, is handled by the RTE and does not affect the application.
- I4 provides FAs with all standard services for process management inside a RTE in relation to memory and time.
- I4 provides FAs standard services to communicate with other applications and system. The communication is realized via platform with I0 interface that is out of scope.
- I4 provides FAs a configuration structure to define specific behaviour of all services
- I4 provides FAs a configuration structure to identify every communication node of other application or system needed to realize a Business Logic.

7.4.5 Interface I5 and SL

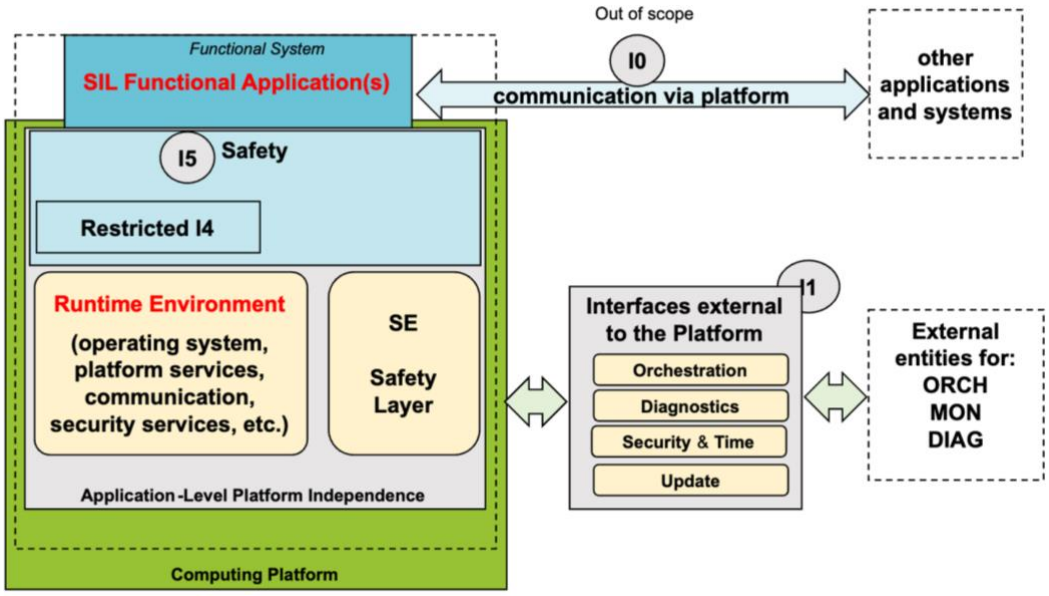


Figure 49: Safety Integrity ALPI interface I5

I5 provides runtime services to SIL FA. (Ref.Figure 49)

I5 goals:

- I5 shall standardize the access to SE services, allowing to FA developer a simpler and common way to develop SIL FA.
- I5 is a “runtime” interface, and it is assumed not having “deploy interfaces”. The basic deployment of an FS, and so of a FA, is handled by the RTE and does not affect the application.
- I5 provides FAs with all standard services for process management in relation to safety related function that implement safety mechanisms to realize composite safety, such as replica synchronization and execution.
- I5 provides FAs a configuration structure to define SIL of services that transparently manage the safety mechanisms.

7.4.6 Implementation models

7.4.6.1 Functional Applications, Tasks and Deployment Configuration

Functional Applications implement the logic of typical railway functions. They consist of one or multiple Tasks, each having distinct functions. Depending on a Task’s function in the system, it may be restricted to use the corresponding limited subset of the ALPI and must comply with the applicable defined set of standardised safety related application conditions.

To achieve deployment independence, every Functional Application shall include a platform-agnostic deployment configuration (see REQ-ALPI-018) that defines, for each Task, in a standardised and abstracted way, its safety, resource (e.g., timing, memory, etc.) and communication requirements.

When integrating a Functional Application with a specific platform instance, the platform-agnostic deployment configuration shall be translated to a platform specific application configuration. A functional independence is required between different tasks (FAT-n) and shall be demonstrated.

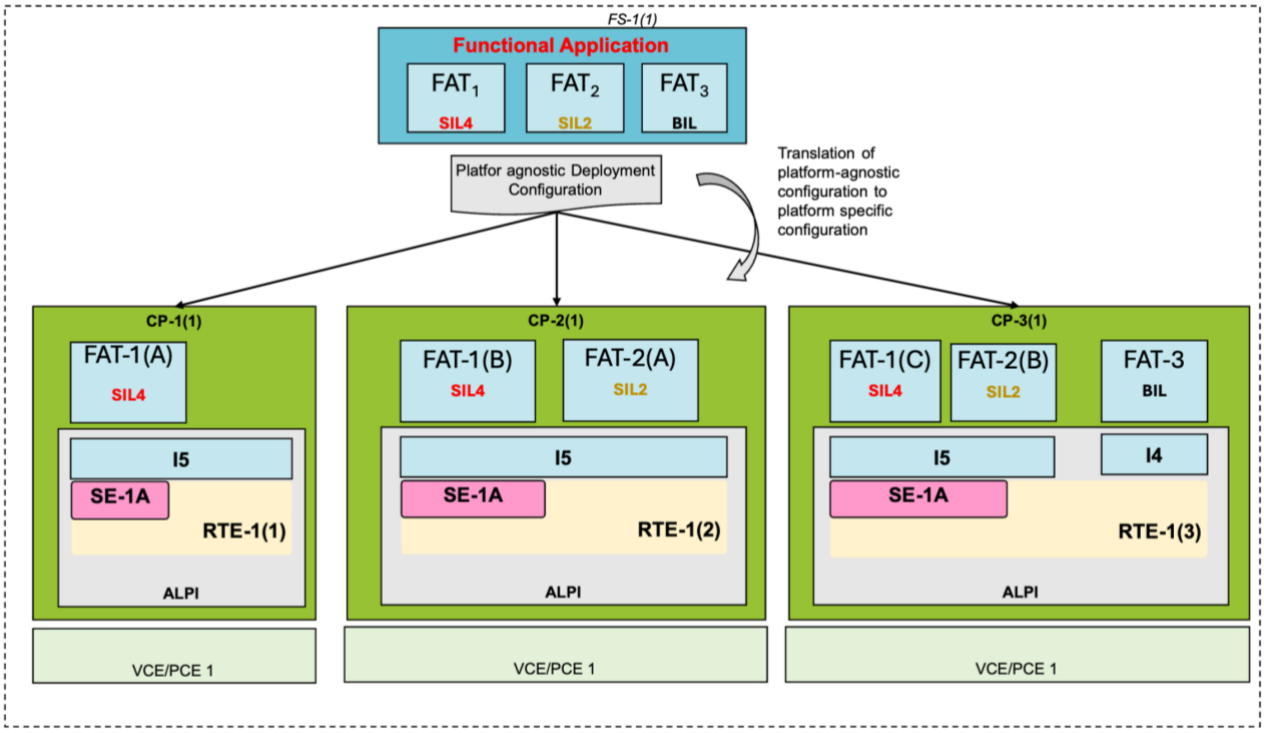


Figure 50: Functional Applications, Tasks and Deployment Configuration

7.4.6.2 Messaging

Exchanging information via messages is a key service of the platform. The messaging concept shall follow the below key paradigms:

- Location transparency: It shall be transparent to a Task of a Functional Application whether it is communicating to a local entity (i.e., residing on the same local platform instance) or a remote entity (i.e., residing on a remote platform instance);
- Replication transparency: It shall be transparent to Tasks of a Functional Application whether they themselves, and the Tasks of the Functional Application they are exchanging messages with, are replicated or not;
- Authentication and authorization transparency: Authentication and authorization of entities shall be transparent to Tasks of a Functional Application, so that Tasks of a Functional Application can trust that the entities they are receiving messages from or transmitting messages to are the entities they claim to be;

Messages between Tasks of a Functional Application or with Tasks of another Functional Application shall be exchanged via Messaging Relations between the respective Tasks. Messaging Relations shall be managed by the platform. They can be joined, or disjoined, registered or subscribed to. Once a Messaging Relation between two entities is established it can be used to exchange messages.

A Messaging Relation shall have various properties related to the usage of voting, the usage of specific Safe Communication Protocols, quality of service, etc.

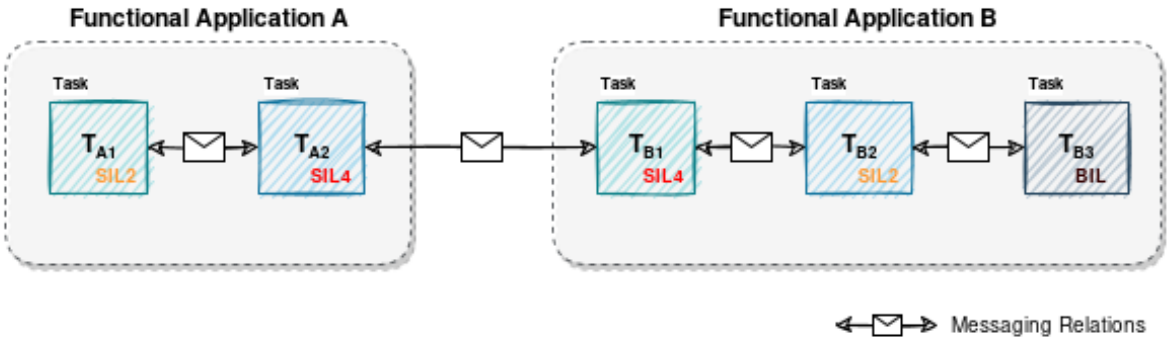


Figure 51: Messaging Relations between Tasks

Depending on the replication of the entities involved in a Messaging Relation, the message exchange may involve message voting and/or distribution – both shall be transparently handled by the platform.

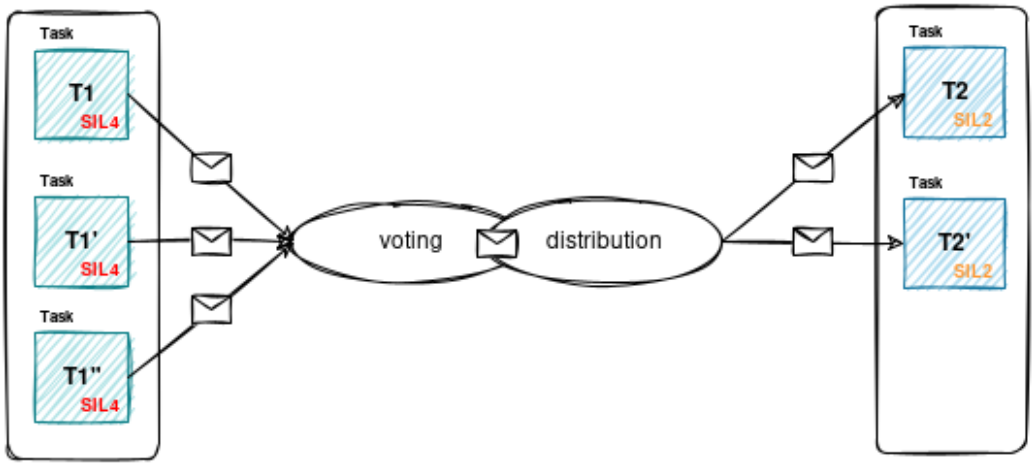


Figure 52: Message voting and distribution

Uni-directional Messaging Relation (publish/subscribe): the transmission of messages from one or multiple publishing Tasks to one or multiple subscribing Tasks without implicit message acknowledgement from the receiving side. Uni-directional Message Relations may have exactly one publisher or multiple publishers.

Key characteristics:

- Posted messages (on the same Messaging Relation and by the same publisher) shall be delivered to all subscribers in the exact same order as they have been published;
- Missing messages shall be identified by the platform (e.g., through the usage of message sequence numbers or some other platform-specific mechanism). The subscribed entities shall be notified by the Platform whenever there are missing messages;
- Messages shall be time-stamped by the platform, so that subscribers are able to determine how old messages are, and whether they should still be processed or discarded, etc.

Bi-directional Messaging Relation (request/respond): the transmission of messages from exactly one requesting Task to exactly one responding Task, with an explicit response message to each request

message. A Bi-directional Messaging Relation can be used for requests from the requesting Task once both sides have joined the Messaging Relation.

Key characteristics:

- Posted messages shall be received by the receiver in the exact same order as they have been sent. This applies to both messages sent by the requester, and the response messages sent by the responder;
- The platform shall deliver messages (both requests and responses) exactly once depending on the performance achievable by the MPC implementation[7] . It is to be investigated the impact of this need on the design and architecture, and the possibility to adopt less stringent constraints, consistent with safety requirements
- Messages shall be time-stamped, so that the involved Tasks are able to determine how old messages are, and whether they should still process or discard them, etc
- The platform shall inform the requesting Task when the responding Task has joined the Messaging Relation (for the first time, or, e.g., after a crash)

7.4.6.3 Task and Thread Scheduling

Platform implementations shall have the maximum freedom regarding the scheduling of Tasks of Functional Applications, as long as a minimum set of design principles are met:

- Task replicas and their threads shall be scheduled based on the following kinds of triggers (or combinations of theses):
  - timer-based, i.e., in configured regular intervals, or in the form of one-shot timers;
  - event-based, i.e., upon receipt of (certain types of messages);
  - timer- and event-based, i.e., the Task obtains execution time in regular intervals, or in the form of one-shot timers, only if (certain types of) messages have (or have not) been received.

(see REQ-ALPI-028)

7.4.6.4 Time

7.4.6.4.1 *Timestamps and Task replication*

The platform shall be able to provide timestamps with the following three different quality attributes:

Unsynchronized Timestamp: corresponds to the time at the point when the replica requests this (and for which different replicas of the same Task may obtain a different result).

Synchronized Timestamp: the exact same time for all replicas of the same Task requesting this (even if there is a time lag between the different replicas in when this

[7] The exact delivery of messages is hard to achieve. They are very costly and require a lot of coordination. Particularly when low latency is required

is requested). This is especially important if the timestamp is used in any voted output message.

Precision: The precision of timestamp is application dependent and accuracy varies accordingly. Precise timestamps are computationally much more expensive than more imprecise ones. This has an impact on requirements and development.

Tasks must ensure that they only use the unsynchronized timestamp in cases where it doesn't impact any potentially voted output. The synchronized time may have a lower resolution than the unsynchronized time.

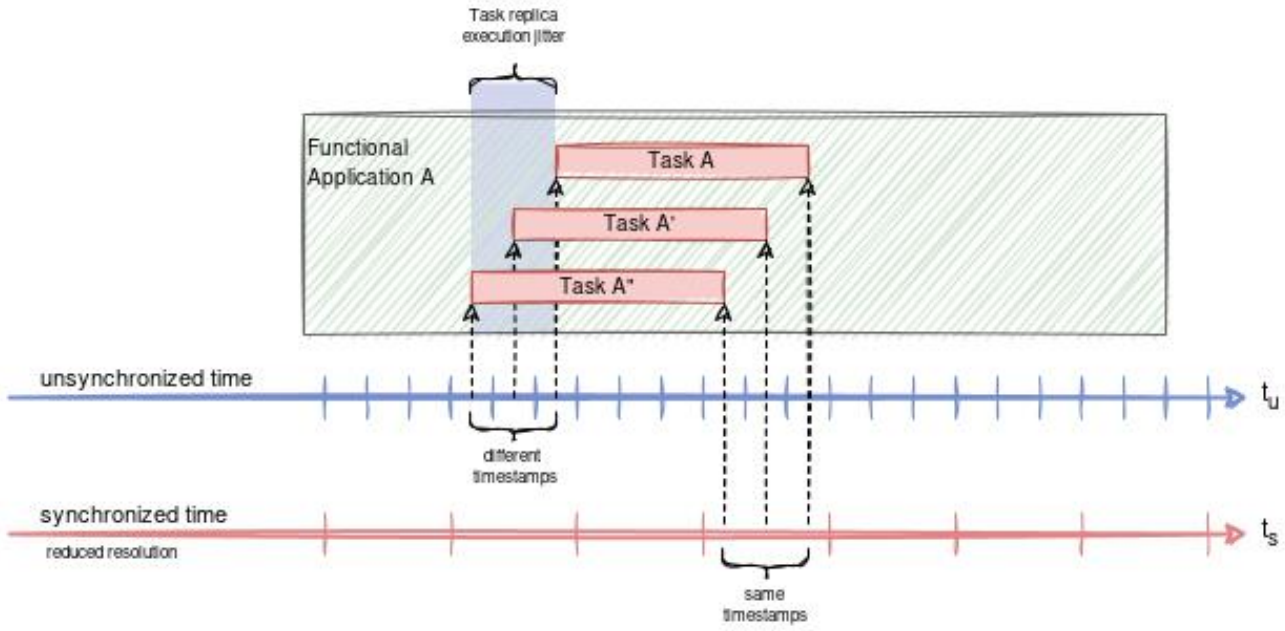


Figure 53: Unsynchronized vs. replica synchronized time

7.4.6.4.2 Timestamps and Messages

For safety as well as for availability reasons, it is essential for Tasks that messages are not delayed beyond a defined maximum message delivery time. The platform shall supervise the message delivery time and inform interested Tasks in case the maximum message delivery time is exceeded.

The platform shall complement messages exchanged via Messaging Relations with timestamps, allowing receiving Tasks to make decisions based on the age of a received message and possibly take appropriate action.

To calculate the age of a message, the notion of synchronized platform clocks (also among distributed platforms) is necessary. Whether messages are complemented with relative or absolute timestamps is for further study.

7.4.6.5 Gateway Concept

To enable Functional Applications to communicate with external entities, a gateway concept is required. platform internal communication, i.e., communication between Tasks running on the same platform, uses Messaging Relations as described in the pervious chapter. In order to exchange information between Tasks running on different platforms, a gateway is necessary. (see REQ-ALPI-032)

Key paradigms regarding external communication are:

- it shall not be visible to a Task of a Functional Application whether it is communicating to another entity on the same platform realization or a remote entity;
- it shall be possible to deploy Tasks of Functional Applications on different platform realizations without having to change the Task implementation;
- required safe or non-safe communication protocols shall be separated from the Functional Application to allow independent evolvement;
- it shall be possible to add new protocols (safe and non-safe) when they become available.

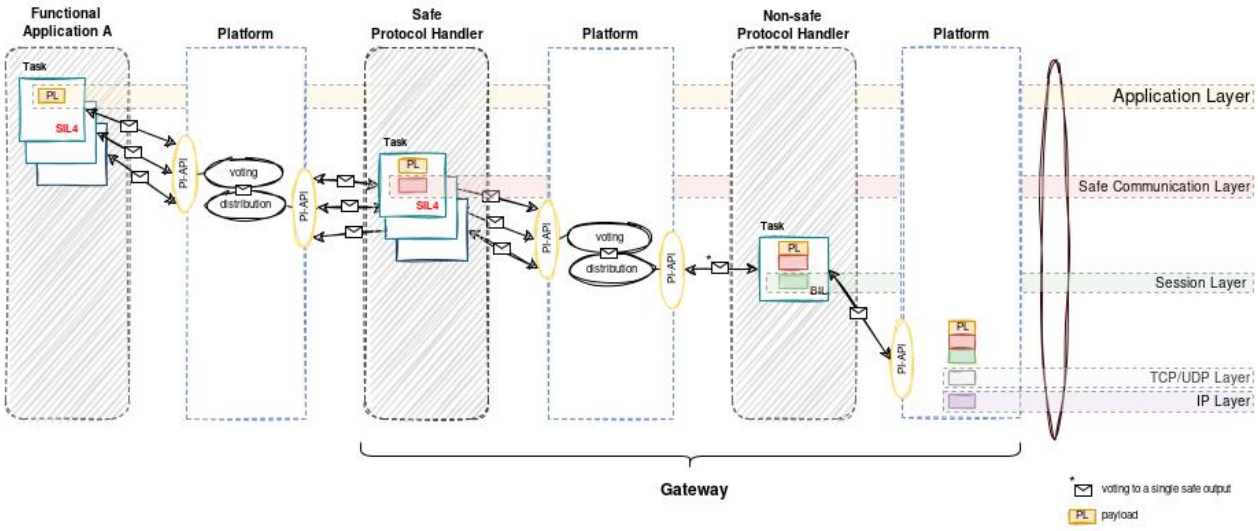


Figure 54: Gateway – contribution to protocol stack

The above scenario depicts a Task of Functional Application A sending a safety critical payload PL to an external system using the gateway concept. The diagram shows how the different entities involved contribute to the overall communications protocol stack toward the external entity.

The Gateway consists of three parts: the safe protocol handler, implementing the safety protocol compliant with the required criticality (e.g., SIL4); the non-safe protocol handler implementing the session layer protocol (e.g., FRMCS); and the platform services implementing voting to a single safe output as well as providing the lower protocol layers e.g., UDP/TCP and IP.

7.4.6.6 Fault, error and failure handling and recovery

The chapter describes how faults, errors and failures shall be handled in context of replicated Tasks and virtual/physical Computing Elements. The subsequent sections follow the terminology used in EN 50129:2018. (see REQ-ALPI-037)

Term	Definition in EN 50129:2018	Meaning in context of replicated Tasks	Expected platform behaviour
Fault	Abnormal condition that could lead to an error in a system	Abnormal condition that could lead to an error in a Task and/or virtual/physical Computing Element.	See section 7.4.6.6.1

Term	Definition in EN 50129:2018	Meaning in context of replicated Tasks	Expected platform behaviour
Error	Discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition	Task replica(s) and/or virtual/physical Computing Element(s) showing a discrepancy between a computed, observed or measured value or condition and the true, specified or theoretically correct value or condition. Example: A Task replica provides different output than its counterpart replicas (or no output at all).	See section 7.4.6.6.2
Failure	Loss of ability to perform as required	Errors of Task replica(s) and/or virtual/physical Computing Element(s) cannot be mitigated by restarting replica(s) or moving them to other Computing Element(s). As a result, Functional Application(s) are impacted in the way that these lose the ability to perform as required.	See section 7.4.6.6.3

Table 5: Fault, Error and Failure in the context of replicated tasks

7.4.6.6.1 Fault Detection and Response

To what extent the platform performs fault detection is platform implementation specific. Nevertheless, Task fault containment must be ensured by sufficient independence between Task replicas (according to EN 50129:2018). It is also left to the discretion of the platform implementation to decide whether a fault (according to EN 20129:2018) has to be flagged as an error.

7.4.6.6.2 Error Detection and Response

Errors shall be detected and handled according to EN 50129:2018. In addition, the platform shall take the following recovery and informational actions:

Affected entity	Actions
Task replica	• Restart the Task replica, recover its state and re-integrate it with its counterpart replicas. • Inform interested Tasks about the affected Task replica failure.
Computing Element	• Restart the virtual/physical Computing Element and recover or restart all affected Runtime Environment instances and Task replicas, recover their state and re-integrate them with their counterpart replicas. • Inform interested Tasks about the affected Task replicas failure.

Table 6: Error detection and response for different entities

In case all recovery actions defined in the above table are unsuccessful (e.g., due to repeated Task replica and/or Computing Element failure, or because more replicas of the same Task are affected

than the redundancy/voting configuration allows for), this results in a failure (according to EN 50129:2018).

7.4.6.6.3 Failure Response

A failure implies that one or multiple Task(s) are no longer able to perform as required. In this case, the platform reaction shall be as follows:

- informs all Tasks that have a Messaging Relation with the affected Task
- informs all Tasks that have registered for diagnostics information about the affected Task

7.4.6.7 Communication Model

The ALPI interface assumes that communications made via the platform, using the I0 interface, are not within the scope of this document.

With reference to Figure 47, the FA performs processing on input data and produces output results that are managed through a communication system. Communication can take place either within the FS or with other external FAs or systems. In order to facilitate and standardize the development of the application, the ALPI interface shall provide a single model of communication services that make it transparent to the FA whether the peer node is internal to the FS or it belongs to another external system. The information that defines the node type, and therefore that allows the RTE layers to implement the communication correctly, is specified in the configuration part of ALPI. The communication services provided by ALPI, after appropriate configuration, allow to implement with a single standard model, every elementary communication between the basic components (FAT) constituting an FA.

ALPI communication interface is based on platform services based on messages as described in chapter 7.4.6.2.

7.4.7 Certification

The discussion around this aspect of application-level platform independence has not started yet. However, there is a dedicated subsequent task in this work package to study certification approaches for modular platforms.

7.4.8 Safety

The MCP allows the management of non-safety related, Basic Integrity Level, and safety-related up to SIL4 Functional Applications. In the case of non-safety related FAs, the FA uses all services available in the RTE without restrictions. In the case of BIL, the FA uses only RTE services compliant with BIL. In the case of SIL4, the Functional Application is implemented through the adoption of mechanisms based on composite safety, which is ensured through redundant architectures compliant with the standard (CENELEC EN50126 / EN50716). The services provided by I5 ALPI effectively enhance transparency by consolidating safety-related data within a secure safety layer safety layer.

The runtime consolidation of the safety-related data is executed through I5 using specific information defined in the configuration part of ALPI.

7.4.9 Security

The ALPI shall provide runtime Security functions in the scope of the Functional Application.

MPC's Security related to external communication is managed end-to-end with TLS at network level. It stops within the RTE, at kernel level, below I4/I5. So, it is not in scope of runtime services of ALPI.

For the security functions in the scope of the Functional Application the ALPI shall provide runtime and configuration services allowing FA to use security services, such as PKI management, authentication management, cryptographic verification/validation inside FATs. (see REQ-ALPI-034)

With reference to the paragraph 6.8, security features should be implemented within each FS. Depending on the solutions and architecture used, the impact on ALPI should consist in the use of a standard interface provided by the IT-Security Layer and included in I4 (chapter 6.8.2), or an integration in I4 of a specific interface towards a compartment dedicated to IT security functionalities. (chapter 6.8.3).

7.4.10 Diagnosis

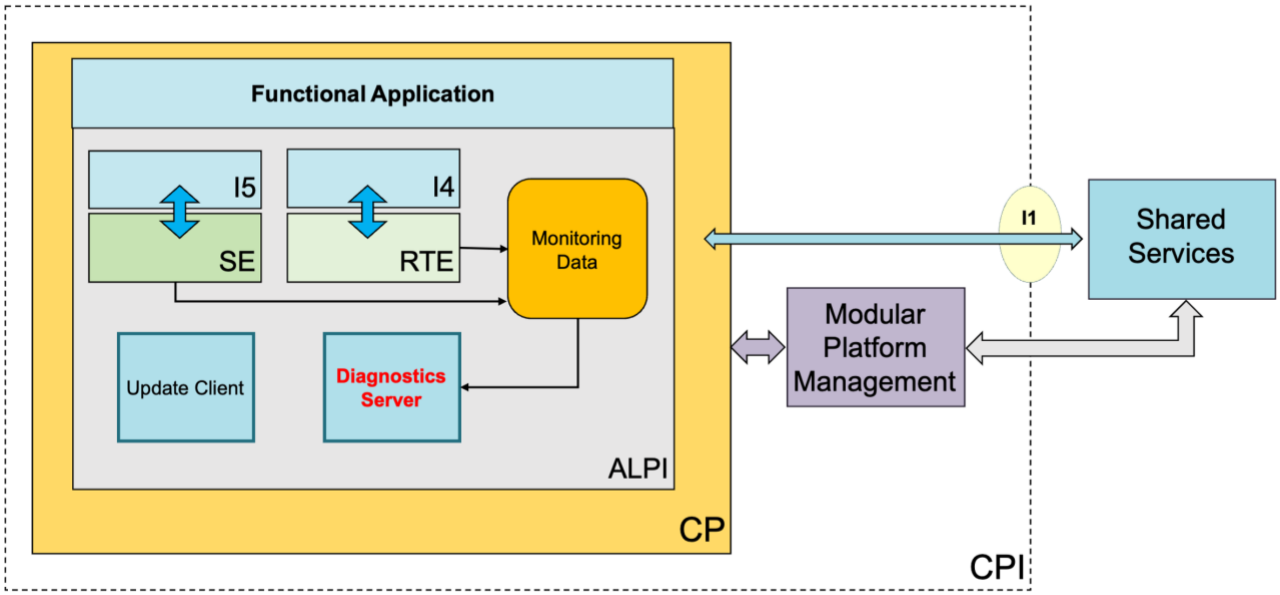


Figure 55: ALPI diagnosis-provided through RTE at CP level

FA's diagnosis data must be provided via the interface I1 (towards the Shared Services Diagnosis) that is considered a central data sink. The ALPI provides runtime services and configuration structures that allow direct and implicit collection of diagnostic data. This data will be transformed within the Platform Management into I1 compatible format.

FA uses runtime ALPI's diagnostic services to provide relevant diagnosis data for root cause analysis and automated initiation of maintenance activities.

7.4.10.1 Diagnosis of the FA

Each FA shall provide diagnosis data about its own health state (e.g. about the health state of running FATs, the outcome of plausibility checks for the Business logic, …).

This FA diagnosis data, provided via ALPI services, shall be available to the FS to be sent to I1 Diagnosis Interface.

FA Diagnosis data is related to faults, errors and failures described in paragraph 7.4.7.6. Some of these are detected at RTE level and aren't in scope of FA. Relevant FA diagnosis data, detectable at RTE level, to be provided towards Platform Management is:

- state of each individual FAT
- state of the communication between FATs
- …

7.4.11 Maintenance

ALPI's I4 and I5 are mainly runtime interfaces, not deploy interfaces. Maintenance and deployment are done at FS level and are not in scope of runtime services of ALPI

The basic deployment of an FS is handled by the RTE (below I4/I5) and does not affect the application.

The maintenance services are done through appropriate external interfaces. External entities manage the ways to build, deploy and maintain a full MPC system and in particular FA & ALPI. (see REQ-ALPI-035)

7.5 Collection of Topics For Future Study

This chapter lists several open topics with regards to Application-level Platform Independence (ALPI) and its central ALPI interface for future study within the work package. The list is not expected to be complete.

- enabling of the development of portable Functional Application, including standardized configuration, update and other artefacts for deployment (see REQ-ALPI-07)
- Interoperability and reusability of applications from different suppliers, enabled by several abstraction mechanisms, e.g. to achieve independence from a specific RTE implementation
- definition of acceptable migration effort from one platform to another (on a scale from binary compatibility meaning zero effort, up to full redevelopment meaning maximum effort) balancing all stakeholder needs, with a strict goal to minimize effort and dependencies where feasible
- possibilities for identification and definition of harmonised SRACs
- integration of diagnostics interfaces for application usage (e.g. operation data coming from the business logic)
- Versioning for all artefacts (also in the context of integration efforts in modular PRAMS). The API shall enable evolvability but at the same time ensures stability and distinctive different life cycles of applications (e.g. deployment).
- syntax and semantics of the ALPI interfaces
  - documentation should contain examples to highlight sematics
  - difference vs "should not use" & "cannot use" w.r.t. SILx

  - handling of different programming languages
- safety and fault tolerance and availability mechanisms shall be provided by the underlying platform, transparently for the applications
- a generic communication model, independent from the actually used transport and from a concrete deployment, shall be used
- be an enabler for safe and secure end-to-end communication, without the need to implement explicit protocols within the application
- be an enabler for modular certification
  - granularity and certification scope need to be developed based on the artefacts defined
  - deployment and update scenarios for artefacts and platform components
  - robust versioning scheme integrated into platform and interfaces
  - forward and backward compatibility needs for interfaces needs to be described
- recording of application and platform events, also usable for juridical recording
- a generic motivation and expectations towards standardizing an ALPI interface
- tools needed over the lifecycle: generic or specific or in-between?
- different targets & different safety levels
  - what can stay the same? what needs to be different? where do we need to innovate? what does the platform need to know/what needs to be configured?
  - example: "vital memory data allocation" e.g. for lockstep systems is a special thing that does not need to happen in other types of systems or with less requirements towards reliability

7.6 Conclusion and Outlook

While previous work and the discussions outlined in this chapter already show on a high level what a future modular platform could look like, there is still a lot of work needed to create a coherent and useful concept for the ALPI – the Application-level Platform Independence.

The goals of independence, standardization and ease of use for the definition of the ALPI interface pose new challenges that imply assumptions for a correct compromise with the complexity of a modular platform. Based on the reported description, it was found that the common definition of the I4 interface is influenced by the following factors:

- Level of independence of ALPI: The common high-level description of ALPI highly depends on the choice of RTE. In case of a single RTE provider, the description of the I4 interface coincides with that of the RTE itself; in case of RTE from different suppliers, a specific adaptation interface for each RTE should be provided in addition.
- Level of transparency for safety: based on what is provided by the Safety Environment, I5 should provide complete transparency about consolidations required for up to SIL4 or in any case a minimum set of runtime services. Complete transparency implies a clear definition of the SIL of input/output data at configuration time.

- Level of transparency for security: security communication is not taken in account by ALPI, being managed end-to-end by the underlying layers. ALPI, during the configuration phase, must allow the selection of appropriate security-oriented protocols to be implemented by lower levels.
- Deployment, maintenance: I4 is primarily a runtime interface, so deployment and maintenance is delegated to specific tools provided by the RTE at FS level. In this case, ALPI requires a configuration data structure for the aggregation of the basic components. The aggregation is managed at FS level.

Possible choices for further simplifications and standardizations: the descriptions of I4 and I5 are based on the safety integrity level of services. It is possible either an incremental approach in which a safety-related FA uses I4 by default and adds I5 for the part involving the SE, or it uses an I5 that integrates a restricted I4, in congruence with the target SIL 1 up to 4.

The definition of the requirements in the appendix was made on the basis of an independent approach, which however also took into account the results of the solutions proposed in chapters 6 and 8.

7.6.1 Open points

7.6.1.1 RTE Single/multi provider

ALPI's scope is to propose a common high level API description. This can be facilitated through the choice of a single RTE provider. In the case of different RTEs, ALPIs will have to include "adapters" and the management of the associated complexities. The choice to impose any constraints on the RTE provider, in order to simplify the implementation of ALPI, should be evaluated in consideration of the trade-off between flexibility in the choice of multi supplier and the related additional complexity associated with the introduction of "adapters" for RTEs harmonization.

7.6.1.2 Consistent safety

With reference to safety management, the modularity and independence of the ALPI SW from the MPC are not sufficient and it is necessary to introduce an overall safety mechanism to ensure that the safe system is consistent. That is, to ensure that a change of version of ALPI, both I4 and I5, is congruent with the SE. This mechanism must be considered in the ALPI configuration data structure.

8 MANAGEMENT, DIAGNOSTICS AND SECURITY RELATED INTERFACES

This chapter deals with platform management, diagnostics and security related interfaces, either internal to the platform, or external to the platform, the latter being referred to as "I1" (see Chapters 3.7.1 and 5). The interfaces are important for integration and interoperability of modular platforms into the railways' computing landscape and operations. The interfaces discussed here are platform-centric and do explicitly not cater to the needs of the functional applications executed on such a modular platform.

8.1 OVERVIEW ON THE INTERFACES

An overview on the stated interfaces is provided in Figure 56 . The high-level components introduced in this context are described in Table 7.

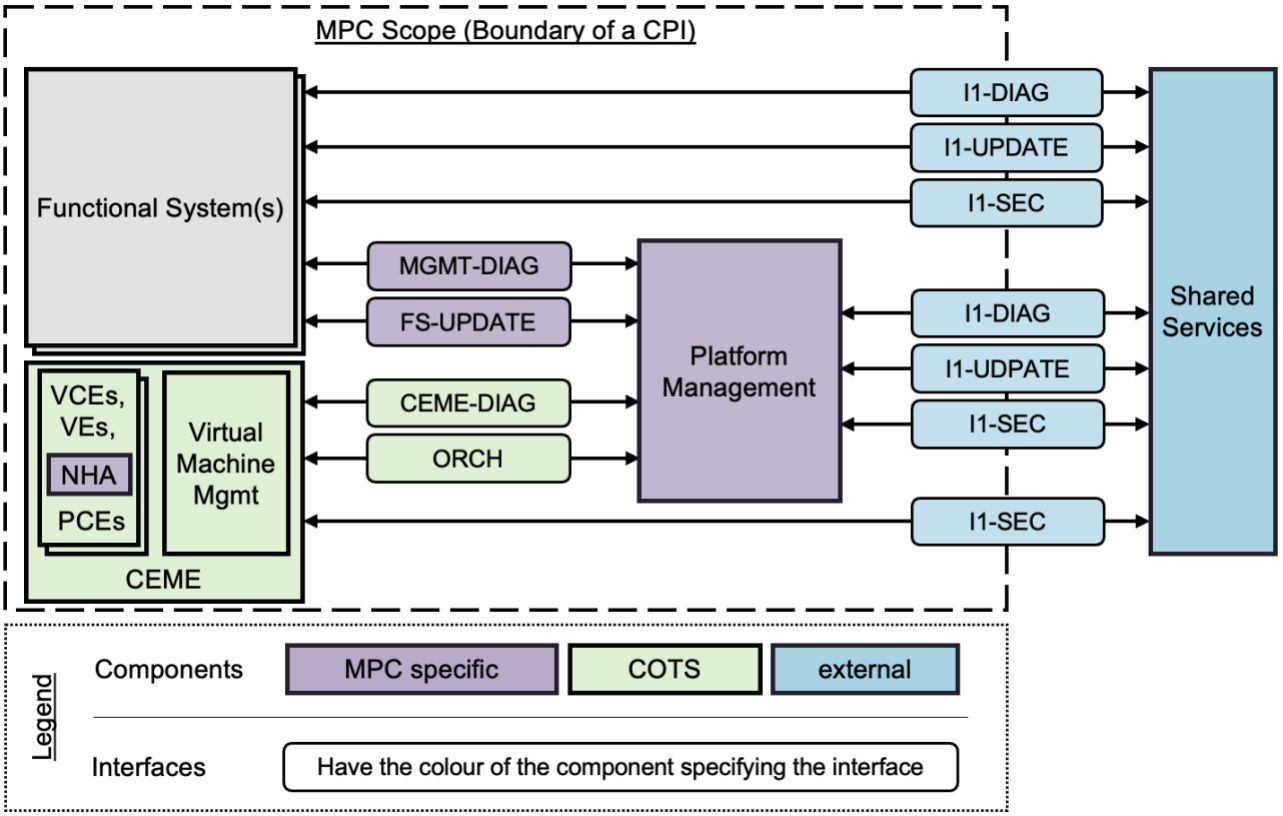


Figure 56: Logical architecture around management, diagnostics and security related interfaces.

Entity	Description
CEME – Compartment Execution and Management Environment	The Compartment Execution and Management Environment comprises the hardware and virtualisation environment along with related (proprietary) • Virtual Machine Management functions, and • Hardware and Virtualisation related Diagnostics Functions.

Entity	Description
	As shown in Figure 49, the CEME is expected to be 100% Commercial-of-the-shelf (COTS), except for the Native Hardware Access (NHA) function that may potentially not be COTS and that is required by the Safety Environments to support Functional Systems with a SIL level.
Functional System(s)	Functional System(s) are defined in chapters 3.7.2 and 5. In the context of the interfaces covered in this chapter, it is important to note that the Functional Systems(s) are expected to contain: • **FS Diagnostics Server**: Entity running in a dedicated Compartment and/or together with a Functional Application in an FS Compartment, which provides diagnostics information to the Platform Management and to Shared Services • **FS Update Client**: Entity running in a dedicated Compartment or together with a Functional Application in an FS Compartment, which responds to and executes update requests from the Platform Management or Shared Services
Shared Services	Shared Services are standardized services related to IT security (e.g., authentication, certificate management), global time provisioning, diagnostics and configuration management / update that are located outside the platform.
Platform Management	This entity supervises the operation of the Functional Systems running on the Virtual Computing Elements in one (or multiple) location(s), e.g. data centre(s). It obtains diagnostics information from the FS Diagnostics Server within the Functional Systems and from the Hardware and Virtualisation related Diagnostics Functions within the CEME. It forwards the information, as appropriate, to the Shared Services via I1-DIAG and reacts by triggering appropriate actions, e.g., the creation of new Virtual Computing Elements inside the Virtualisation Environment or restarting Physical Computing Elements if needed. Note: The Platform Management is not safety-relevant (but ensures the fulfilment of RAM requirements during operation), as the Functional Systems themselves (and the Safety Environments therein) always ensure a safe state and safe output. It is assumed that the Platform Management provides a highly standardized functionality in the way that it reacts to information arriving via CEME-DIAG and/or MGMT-DIAG by triggering actions via ORCH and/or FS-UPDATE in a standardized way, potentially involving FS-specific policies. The Platform Management implements the MPC specific interfaces and abovementioned functionality, potentially utilising as much as feasible COTS solutions, e.g., OpenStack or other solutions.

Table 7: Entities of particular interest in the context of management, diagnostics and security related interfaces.

The interfaces related to management, diagnostics and security are described in Table 8, along with the mapping to the interface names used in D26.1 [18] and in the System Pillar Computing Environments domain [14].

Interface	Description	Mapping to terminology	
		In D 26.1 [18]	In SP CE domain [14]
CEME-DIAG **(Compartment Execution and Management Environment related Diagnostics)**	Via this interface, diagnostics functions within the CEME provide diagnostics information related to the Hardware and Virtualization Environment to the Platform Management (such as the information that there is a HW failure, a VCE failure, etc.). The Platform Management reacts to this e.g., by triggering the (re-)creation of Virtual Computing Elements via ORCH and/or updates of Functional System Compartments via FS-UPDATE. It is assumed that this interface is specified by the chosen COTS CEME including Virtual Machine Management related functionality.	PLAT_HEALTHMGMT	IF-DIAGNOSTICS
ORCH	Via this interface, the Platform Management triggers the Virtual Machine Management within the CEME to setup new Virtual Computing Elements etc. This interface is also used to setup FS Compartments to the extent that they form the endpoint of the FS-UPDATE interface. It is assumed that this interface is specified by the chosen COTS CEME solution, including Virtual Machine Management related functionality.	PLAT_UPDATE	IF-ORCHESTRATION
MGMT-DIAG	Via this interface, the FS Diagnostics Server(s) in the Functional System(s) provide(s) diagnostics information strictly needed for the management of the FS Compartments and V(C)Es (such as information on failures of FS Compartments, etc.) to the Platform Management. The Platform Management reacts to this by triggering the (re-)creation of Virtual Computing Elements via ORCH and/or updates of Functional Systems (Compartments) via FS-UPDATE.	PLAT_HEALTHMGMT	IF-DIAGNOSTICS

Interface	Description	Mapping to terminology	
		In D 26.1 [18]	In SP CE domain [14]
FS-UPDATE	Via this interface, the Platform Management supervises updates of the Functional Systems (Compartments).	PLAT_UPDATE	IF-ORCHESTRATION
I1-DIAG	Via this interface, the FS Diagnostics Server in the Functional System(s) provides additional diagnostics information (beyond that strictly needed for the management of the FS Compartments and V(C)Es) towards Shared Services. Also, the interface can be used by the Platform Management to provide diagnostics information to Shared Services. Note: It is assumed that this interface is specified in the TCCS domain.	PLAT_LOGGING	IF-LOGGING
I1-UPDATE	Via this interface, Shared Services trigger the update of Functional Systems (Compartments) toward the Platform Management and/or directly to the FS Update Client(s) within the Functional Systems (see Section 8.2 for further discussion on this). Note: It is assumed that this interface is specified in the TCCS domain.	PLAT_UPDATE	IF-ORCHESTRATION
I1-SEC	Via this interface, the Virtualization Environment, the Platform Management, and the Functional System(s) access security and synchronization services provided by the Shared Services (see chapter 3.7.4). Note: It is assumed that this interface is specified in the TCCS domain.	PLAT_SECURITY PLAT_SYNC	IF-IT-SEC

Table 8: Interfaces related to management, diagnostics and security internal or external to the platform.

As can be seen from Table 8, some changes in the granularity of the interfaces have been applied compared to the former definitions in D 26.1 and in the System Pillar Computing Environments domain:

- The former “PLAT_UPDATE” / “IF_ORCHESTRATION” interface is now split into two interfaces “ORCH” and “FS-UPDATE”, as they terminate at different points and refer to different layers in the technology stack:
  - The ORCH interface is between the Platform Management and the Virtual Machine Management within the CEME. It is used to manage the setup of Virtualization Environments, Virtual Computing Elements, etc.
  - The FS-UPDATE interface is between the Platform Management and the FS Update Client(s) in the Functional System(s). It is used to supervise Functional System (Compartment) updates in alignment with configurations of the Virtual Computing Elements.
- The former “PLAT_HEALTHMGMT” / “IF_DIAGNOSTICS” interface is similarly split into two interfaces CEME-DIAG and MGMT-DIAG, also because they terminate at different entities and refer to different layers in the technology stack:
  - The CEME-DIAG interface is between the CEME and the Platform Management, conveying diagnostics information related to hardware, the Virtualization Environment, Virtual Computing Elements etc.
  - The MGMT-DIAG interface is between the FS Diagnostics Server and the Platform Management, conveying diagnostics information obtained from the application domain.

8.2 General Assumptions on the Interfaces

It is important to note that the aforementioned interfaces are all assumed to be **not safety relevant**. Their failure may have an impact on the availability of a Modular Computing Platform (e.g., because the Platform Management is not able to correctly react to a hardware or software failure), but mechanisms in the Functional Systems (or in the Safety Layer therein) always ensure that safety is fulfilled.

Regarding **updates of Functional Systems** and the usage of the **I1-UPDATE** and **FS-UPDATE** interfaces it is also important to note that at this point two proposals shall be provided as potential options for further study:

a) **Shared Services could directly interface to the FS Update Clients in the FS Compartments via I1-UPDATE** to trigger updates of Functional Systems (Compartments). This requires that Shared Services are aware that Functional Systems run in the form of redundant replicas in multiple FS Compartments and need potentially awareness of constraints or procedures that have to be considered for the update of these (for instance, FS Compartments may have to be stopped, updated and started in a certain order or with certain timing constraints, etc.). In this case, the Platform Management needs to be informed about any FS Compartment updates triggered by the Shared Services, so that it can supervise needed activities appropriately (and for instance correctly trigger the re-creation of a Virtual Computing Element) when FS Compartment failures are reported through the MGMT-DIAG interface.

b) **Shared Services interface to the Platform Management** and direct any update requests related to Functional Systems to the Platform Management. In this case, the Shared Services need not be aware of the deployment of replicas and FS Compartments, but rather see a Functional System as one atomic entitity that can be updated as a whole based on provided

information. The Platform Management, being aware of the notion of FS Compartments and the related FS Deployment Rules, can then translate update requests from Shared Services into a sequence of requests via FS_UPDATE to the FS Update Clients in the FS Compartments, also taking into account FS Deployment Rule specific timing constraints, etc.

Which of such options is to be used (or whether both would possibly be used concurrently or in combination) needs further investigation. It is assumed that option a) above introduced higher demands on the I1-UPDATE interface specified in the TCCS domain, as the Shared Services need to apply configuration and software updates of Functional Systems, e.g., running in data centres on a different granularity and with different constraints than when performing configuration updates of, e.g., field elements as currently foreseen. Option b) may require less complexity on the I1-UPDATE interface specified in the TCCS domain, as the Platform Management, where the interface would terminate on the platform side, would supervise the implementation, managing their complexity and constraints.

8.3 Requirements on the Interfaces

The requirements for the interfaces mentioned can be found in Appendix D, Management, Diagnostics and Security related Interface Requirements.

The requirements for interfaces being part of I1 (e.g., I1-DIAG, I1-UPDATE, I1-SEC) are not in the scope of work package 26. These interfaces are specified in the System Pillar TCCS domain. It is assumed that the modular platform concept and the entities described in this chapter would reuse the I1-DIAG and I1-SEC as they are considered in the TCCS domain so far, without posing MPC-specific requirements. However, it is still to be checked, ideally in the dialogue between R2DATO and the TCCS domain in the System Pillar whether this is indeed the case.

For the I1-UPDATE, as stated in Section 8.2, further investigations are required w.r.t. how this interface is used in concurrence with FS-UPDATE. From this investigation, further requirements on I1-UPDATE may be derived that would have to be considered in the TCCS domain.

8.4 Conclusions and Next Steps

This chapter has delved into platform management, diagnostics and security related interfaces. A distinction has been drawn between interfaces within the CPI – Compatible Platform Implementation (ORCH, CEME-DIAG, FS-UPDATE, MGMT-DIAG) and those toward Shared Services external to the CPI (I1-DIAG, I1-SEC, I1-UPDATE).

For the interfaces within the CPI, requirements have been derived. The common understanding is that

- The interfaces from the Platform Management to the Functional Systems - **MGMT-DIAG and FS-UPDATE** - are specific to the MPC and can be specified exactly according to the identified MPC needs. Here, the next steps are to further develop the concept and subsequently identify suitable existing protocols for these interfaces and specify the interfaces in detail;
- The interfaces from the Platform Management to the Compartment Execution and Management Environment (CEME) – **ORCH and CEME-DIAG** - are expected to be based on existing interfaces provided by existing COTS implementations of the CEME. Here, there is hence no degree of freedom to specify these interfaces, but it rather has to be checked whether existing COTS solutions fulfill the requirements on these interfaces identified in this

deliverable. At the time of writing of this deliverable, there is confidence that there are multiple COTS solutions on the market that can meet the requirements, but a detailed (gap) analysis has to be performed.

For the interfaces from the CPI to the Shared Services – **I1-UPDATE, I1-DIAG and I1-SEC** – it has to be discussed with the ERJU System Pillar TCCS domain, as noted in Sections 8.2 and 8.3, how these interfaces are exactly to be used in the MPC context, and whether additional requirements may be posed on these from the MPC context beyond the requirements already considered in the TCCS domain.

9 CONCLUSIONS

Modular computing platforms are a proven concept in many areas, and this success already motivated many endeavours to enable their usage in the railway domain [18]. Work package 26, within the environment of ERJU Innovation Pillar Focus Project 2 R2DATO, applied the state-of-the-art to the input and the guidance from the ERJU System Pillar, most importantly from the Computing Environment domain, and elaborated on important, railway specific aspects.

This activity created the “Modular Platform Concept” (MPC), which is presented in detail in this deliverable. The analysis and concept work are expected to further contribute to ERJU SP progress, supporting the domains in defining a suitable environment for modular platforms to be commissioned in the future.

The MPC is presented with a basic analysis (see chapter 3), as high-level requirements (see chapter 4 respectively Appendix A), and the architecture concept (chapter 5). The MPC is built on three internal ideas (HLPI, ALPI, and respective interfaces; see chapters 6, 7, and 8) and provides external interfaces for operational integration as specified by ERJU System Pillar.

The overall outcome has two facets: Modular platforms as the MPC are feasible in the railway context. However, the complexity of a fully defined MPC for safety functions set limits to the depth of specification that can be presented here. As well, it’s an ongoing discussion what actual level of definition respectively harmonization of a platform keeps the right balance between interchange-ability and potential for future innovation and differentiation.

As such, the specification presented is not final and implementable in a way that would fulfil all variants of interchangeability, nor would a final & fixed specification have been advantageous at the time this deliverable was to be finished. This is due to several dependencies and implications, content- and timewise:

- A final specification is expected to describe platform implementations that are certifieable and allow for modularity in a way that enables the stated goals. This will be analysed in work package 26’s next task and its results will further guide the MPC specification.
- The input received from the System Pillar was limited to information received via the FP2 Cluster System engineer respectively via direct contacts in the appropriate ERJU SP domains. This input was not stable and changed regularly. When this deliverable was due, there were no published inputs from the relevant SP domains.
- The input from the SP CE domain as shown in chapter 3.7.1 was in some aspects unexpected and did not align with the goals of all of work package 26’s members. The approach planned in work package 26’s first deliverable [18] was not fully compatible with the input received from the CE domain [14], especially the introduction and prioritization of I2 and I3, as well as the different design of I1 were not expected. On one hand, the increasing scope helped the definition of ALPI and HLPI, as introduced in work package 26’s second deliverable and explained in detail in this current deliverable. On the other hand, the complexity similarly rose. Continous alignment with the SP CE domain helped to steer work package 26 into the right direction.
- The SP CE domain conclusion and final output on the computing environment is likely several years out.

- While it was not expected to be available in the timeframe, it's still relevant to note that there is no input yet, e.g. in the form of requirements, coming from any application in the R2DATO context that would target the MPC.
- In the environment described in the bullet points above, it's not feasible to decide on a clear strategy for standardization approaches in the MPC. The goal of interchangeability can be reached on many different granularities and levels. Not limiting future innovation while creating beneficial harmonizations in the architecture and/or interface realms, however, is a hard requirement for future adoption of the MPC. Existing developments in the sector have to be taken into account, and a good and informed compromise needs to be found. One option is to gain the necessary knowledge and drive the adoption in demonstrator projects, such as the "Onboard Platform Demonstrator" work package 36 in R2DATO, the demonstrator cluster of R2DATO in general, and future phases of R2DATO respectively ERJU. Another option could be to collect insights from incumbent platform vendors and design the MPC interfaces accordingly. This has last option naturally has been applied to this deliverable already, as the authors of the relevant chapters have insights into their companies experiences in this field.

Despite all these factors and caveats, this deliverable shows a concrete and complete picture of the Modular Platform Concept, highlighting many important considerations, requirements, approaches for safety and security, and enables the sourcing of relevant core technologies as commercial-off-the-shelf (COTS) products.

1) Complete picture of the MPC
   Chapter 3 "Modular Platforms Concept (MPC)" introduced and discussed purpose, scope, stakeholders, goals, non-goals, assumptions, known issues and limitation, the alignment with the ERJU System Pillar, PRAMSS approaches, user stories, operational context and scenarios, intended usage scenarios and platform environment examples. In chapter 4 "Modular Platforms Requirements", the methodology and sources for an updated and comprehensive set of high-level modular platform requirements were presented, leading to the actual list of requirements in Appendix A "MPC Requirements". Subsequently, in chapter 5 "Modular Platforms Architecture", an architecture approach was developed, based on ERJU SP input and previous work package 26 work. The modularization architecture – how the modular approach is integrated into the architecture – and the service architecture – how different internal and external interfaces are provided to functions and the outside – are explained and are the basis for the detailed discussions later in the deliverable.

2) Requirements
   For each of the MPC's central ideas, requirements were derived and organized in the appendices: Appendix B "HLPI Requirements" lists the key learnings and rationales from chapter 6, detailing the conditions for the broader virtualisation approach within the MPC. In Appendix C " ALPI Requirements", the application level needs for defining interfaces I4 and I5 are listed, based on the work presented in chapter 7. The remaining interfaces in scope of the MPC are discussed in chapter 8 and the resulting requirements can be found in Appendix D "Management, Diagnostics and Security related Interface Requirements". The requirements collected for three topics build a solid high-level starting point for further consolidation and improvement, especially filling in gaps and checking and augmenting the content by an implementation approach.

3) Approaches for Security
   Security approaches are driven by the ERJU SP work as a framework, and also by concepts

on how to implement appropriate measures in the MPC, e.g. in chapter 6.8. Especially enabling decoupled security driven updates is an inherent strength of the MPC.

4) Approaches for Safety
   For the MPC goal to minimize the areas with functional safety integrity levels above Basic Integrity while still providing an up to SIL4 capable environment two approaches have been compared: HLPI and ALPI.
   The HLPI safety discussion motivates certain measures that have to be implemented to support the required safety capabilities in a virtualized environment, see chapter 6.7. The key challenge is the decoupling or harmonization of the SRACs of the Safety Environment in the AEE from the actual implementation details of the CEE respectively CEME. Proposals for this challenge, like the motivation of the so-called “Native Hardware Access” (NHA), have been made and show potential ways forward.
   For ALPI, the I5 interface presents an opportunity for decoupling Functional Applications with SIL greater 0 from the underlying layers. The discussion can be found in chapter 7.

5) Sourcing of COTS components
   The opportunities to source platform-defining components as COTS without the need to develop new technologies or products is a clear advantage of the MPC. As highlighted in chapters 5.1.1 and 6, COTS hardware for the Physical Computing Environment can be used in the MPC, as well as COTS virtualisation solutions can be used for the CEME. The adaptation of special properties or interfaces of the COTS components is achieved by introducing the Platform Management (PM) component within MPC. The PM enables the adaptation of ERJU-driven external interfaces (I1) and Functional System specific internal interfaces (see chapter 8) to the CEME. The CEME interfaces are depending on the COTS solutions integrated into an actual Compatible Platform Implementation (CPI). With the PM, all variants of CPI behave the same way externally and provide the same interfaces (I1).

Detailed technical analyses and further information including respective conclusions can be found in the appropriate chapters. A comprehensive collection of open points is available in Appendix E.

In summary, this deliverable lays the foundation for the Modular Platform Concept, MPC, prominently enabling capabilities such as decoupled lifecycles of software and hardware (MPC-P01), consolidation of more software on less hardware (MPC-P06), and extensibility (MPC-P04) in a railway safety environment using proven, commercial-off-the-shelf hardware and software technologies. The COTS components are integrated using an individual Platform Management component, leaving room for innovation and differentiation of future Compatible Platform Implementations.

The deliverable acts as an input to current and future ERJU System Pillar activities, and to work package 26 task 3, the study on modular certification. The deliverable concludes work package 26 task 2.

REFERENCES

[1] OCORA website
https://github.com/OCORA-Public/Publications

[2] EULYNX website
https://eulynx.eu/

[3] SIL4@Cloud Report
https://digitale-schiene-deutschland.de/Downloads/Report%20-%20SIL4%20Cloud.pdf

[4] SIL4 Data Center Report
https://digitale-schiene-deutschland.de/Downloads/Research%20Report%20-%20SIL4%20Data%20Center.pdf

[5] Computing Platform – Whitepaper:
https://github.com/OCORA-Public/Publications/blob/master/00_OCORA%20Latest%20Publications/Latest%20Release/OCORA-TWS03-010_Computing-Platform-Whitepaper.pdf

[6] Computing Platform – Requirements:
https://github.com/OCORA-Public/Publications/blob/master/00_OCORA%20Latest%20Publications/Latest%20Release/OCORA-TWS03-020_Computing-Platform-Requirements.pdf

[7] Computing Platform – Specification of the PI API between Application and Platform:
https://github.com/OCORA-Public/Publications/blob/master/00_OCORA%20Latest%20Publications/Latest%20Release/OCORA-TWS03-030_SCP_Specification_of_the_PI_API_between_Application_and_Platform.pdf

[8] OCORA Discussion paper about Configuration and Updates
https://github.com/OCORA-Public/Publications/blob/master/08_OCORA%20Release%20R3/OCORA-TWS07-060_Configuration%20Management-Concept.pdf

[9] OCORA-TWS08-010 MDCM Introduction
https://github.com/OCORA-Public/Publications/blob/master/08_OCORA Release R3/OCORA-TWS08-010_MDCM-Introduction.pdf

[10] OCORA-TWS08-030 MDCM SRS
https://github.com/OCORA-Public/Publications/blob/master/08_OCORA Release R3/OCORA-TWS08-030_MDCM-SRS.pdf

[11] OCORA-TWS01-035 CCS On-Board Architecture
https://github.com/OCORA-Public/Publications/blob/master/00_OCORA%20Latest%20Publications/Latest%20Release/OCORA-TWS01-035_CCS-On-Board-(CCS-OB)-Architecture.pdf

[12] OCORA-BWS02-030 Technical Slide Deck
https://github.com/OCORA-Public/Publications/blob/master/08_OCORA%20Release%20R3/OCORA-BWS02-030_Technical-Slide-Deck.pdf

[13] ERJU System Pillar – Computation Environment Domain
https://rail-research.europa.eu/system_pillar/

[14] ERJU System Pillar, Computing Environment – Deliverable "Recommendation on interfaces to be standardised"
Document list: https://rail-research.europa.eu/system-pillar-key-documents/
Document access:
https://eeigertms.sharepoint.com/:b:/r/sites/SPOpenShare/Gedeelde%20documenten/General/23-09-29%20Steering%20Group%206/SPG-STG-D-SPG-101-01_-_20230920_Task_2_Computing_Environment_-_Interfaces_to_be_standardised.pdf?csf=1&web=1&e=VBeC7n

[15] ERJU System Pillar, Computing Environment – Deliverable "System Concept including Operational Analysis" (the old title was used in this deliverable: "Operational Analysis Specification")
Document list: https://rail-research.europa.eu/system-pillar-key-documents/
Document access: *not yet available*

[16] ERJU System Pillar – Common Business Objectives
https://rail-research.europa.eu/wp-content/uploads/2022/10/SP-Common-Business-Objectives.pdf

[17] ERJU System Pillar, PRAMS – Deliverable "Evolution Management of safety related systems"
Document list: https://rail-research.europa.eu/system-pillar-key-documents/
Document access: *not yet available*

[18] ERJU Innovation Pillar FP2 R2DATO, Work Package 26, Deliverable D26.1 "High level Consolidation"
https://projects.rail-research.europa.eu/eurail-fp2/deliverables/

[19] ERJU Innovation Pillar FP2 R2DATO, Work Package 26, Deliverable D26.2 "Intermediate specification of the Modular Platform"
https://projects.rail-research.europa.eu/eurail-fp2/deliverables/

[20] ERJU Innovation Pillar FP2 R2DATO, Work Package 26, Deliverable D26.4 "Summary of findings and recommendations from study on modular certification and homologation"
https://projects.rail-research.europa.eu/eurail-fp2/deliverables/
Document access: *not yet available*

[21] X2RAIL-3 Deliverable 8.2
https://projects.shift2rail.org/download.aspx?id=0a20cac9-e20f-4cdf-bc63-e0cb28950cfd

[22] EuroSpec European Specifications for railway vehicles
https://eurospec.eu

[23] EuroSpec Software Updates specification V1.0
https://eurospec.eu/download/software-updates-v1-0/

[24] EuroSpec Maintenance Software specification V1.0
https://eurospec.eu/maintenance-software/

Appendix

APPENDIX A MPC REQUIREMENTS

This chapter provides selected and adapted requirements from several sources as listed below, and potentially additional requirements derived here by the work package. The selection process focused on capturing a subset of important characteristics that differentiate the MPC from a traditional railway computing platform. Not all sources previously mentioned (in chapter 4) have had requirements that were in suitable format or specificity to allow their usage in the following list.

Column	Meaning
Source	O: OCORA Modular Platform Requirements [6] S: SP CE Domain OAS [15] (released to Mirror Group on 2024-05-07) +: new #: (heavily) modified or rewritten
Scope	F: full (all target environments) OB: On-board required, trackside optional TS: Trackside required, on-board optional O: optional for all environments X: not in scope for work package 26

Table 9: Sources, Scope and Legend for the requirements table

The following notes capture relevant aspects of the methods applied when selecting and modifying the requirements.

Note 1: Most requirements in the following table have been modified where necessary to reflect the correct usage of the glossary terms from chapter 3.7.2 – this is indicated by the usage of "#" next to the source identifier. Also, the system under consideration was changed to the MPC (Modular Platform Concept) where needed. Some requirements were fully rewritten, as indicated by a "+" in the source field.

Note 2: Some requirements are only relevant when some form of application-level platform independence approach is chosen (on I4 respective I5 level as introduced in chapter 3.7.1). These requirements are prefixed with "Where application-level platform independence is used, …" to indicate the relevant configuration.

Note 3: The “Rationale” explains the reasoning behind the requirement.

Note 4: The “Satisfies” relation tries to connect the requirements to on ore more items of the MPC’s purpose, scope, goals, assumptions, and limitations; at least where feasible and helpful (see chapter 3).

ID	Source	Requirement	Scope
R001	O# MSCP-21	The MPC shall execute Functional Applications with SIL ranging from BI up to SIL4. *Rationale:* *The MPC is a universal platform for all needs. However, actual implementations can limit the supported SIL where necessary, e.g. for trackside BI systems.* *Satisfies:* *MPC-P03*	F
R002	O# MSCP-20	Where multiple Functional Applications are present, the MPC shall execute Functional Applications independent of their individual SIL. *Rationale:* *The SIL can be mixed within Functional Systems and across multiple Functional Systems and its Functional Applications. The MPC implementations have to support arbitrary mixed SIL within their limitations of maximum SIL.*	F
R003	O# MSCP-17	The MPC shall conform to the interface specification of the System Pillar Computing environment domain. *Rationale:* *Interfaces I1 to I5 are specified by the SP CE domain. The OCORA source requirements only referred to I4 and I5.* *Satisfies:* *MPC-P01, MPC-P02, MPC-P03, MPC-P05, MPC-A02*	F
R004	O# MSCP-23	Where application-level platform independence is used, the MPC shall transparently encapsulate the safety and fault tolerance mechanism. *Rationale:* *All safety-related functions not inherent in the application logic shall be implemented as part of the platform. As platform vendors may use their specific approaches to handling safety and fault tolerance, it must be fully encapsulated in the platform. Applications must not include any platform specific code related to safety or fault tolerance.* *Satisfies:* *MPC-P03*	F

ID	Source	Requirement	Scope
R005	O MSCP-18	The MPC shall enforce a clear separation between the platform hardware and the runtime environment. *Rationale:* *A clear separation between hard- and software simplifies platform life-cycle management. Typically, the hardware has a much shorter lifetime than the software running on top of it.* *Satisfies:* *MPC-P01*	F
R006	O MSCP-29	The MPC vendor shall be responsible to ensure (by provision of tooling, documentation, generic product certification, etc.) that a solution using the platform is certifiable according to CENELEC without the explicit involvement of the Computing Platform vendor. *Rationale:* *A full decoupling of the life-cycles of the Computing Platform and the Functional Applications requires that a deployment of Platform and Applications can be homologated without the explicit involvement of the Computing Platform vendor.* *Satisfies:* *MPC-P07, MPC-L02*	F
R007	O# MSCP-89	The Application vendor shall be responsible to ensure, by providing all required artefacts, that a Functional System can be integrated on a MPC and homologated without the explicit involvement of the Application vendor. *Rationale:* *A full decoupling of the life-cycles of the MPC and the Functional Systems requires that a deployment of Platform and Applications resp. Systems can be homologated without the explicit involvement of the application(s) resp. System(s) vendor(s).* *Satisfies:* *MPC-P07, MPC-L02*	F
R008	O# MSCP-28	Where application-level platform independence is used, the MPC shall define a unified set of safety related application conditions (SRACs) (at least to the extent that they relate directly to the Functional System) which all safety critical Functional Systems must comply with in order to be certifiable according to CENELEC safety standards. *Rationale:* *In order to be able to port Functional Systems from one Platform implementation to another, it is key that all Platform implementations delegate the exact same set of safety related application conditions to the Functional Systems. Otherwise, Functional Systems would have to be modified to comply with different conditions on different platform implementations.* *Satisfies:* *MPC-P07, R006, R007*	F

ID	Source	Requirement	Scope
R009	O# MSCP-30	The MPC shall meet the relevant security system requirements as defined by the System Pillar Cyber Security domain. *Rationale: Using a standards-based approach, ensures that adequate controls, processes and procedures are in place to ensure the protection of the platform confidentiality, integrity and availability.* *Remark: The standards situation is being cleared up and the relevant guidelines are expected from the System Pillar. For the HLPI approach, additional investigations can become necessary. IEC 62443:2013 SL3 can be assumed to be the baseline.*	F
R010	O MSCP-109	The MPC shall ensure the authentication and authorisation of Functional Systems. *Rationale: The Platform has to ensure that only authenticated Functional Systems and their components can use the platform. Further Functional Systems and their components need to be able to trust that the entities they are receiving messages from or transmitting messages to are the entities they claim to be.*	F
R011	O MSCP-36	The MPC shall ensure independence (e.g., in CPU and memory usage) between Functional System components to fulfil the CENELEC norm EN 50129:2018 or later. *Rationale: This is required for CENELEC compliance EN 50129:2018 or later.*	F
R012	O# MSCP-38	The MPC shall offer a management interface to set a Functional System to be active or inactive where inactive means that it is not executed and can be moved/replaced/updated. *Rationale: The operations accessible from the outside of a platform are on the Functional System level.*	F
R013	O# MSCP-33	The MPC shall provide the Functional System the ability to report that it needed to deactivate itself. *Rationale: A Functional System might see the need to deactivate itself due to safety goals not being met. The MPC needs to be aware of this change.*	F
R014	O MSCP-91	The MPC shall provide a management interface on which information is provided when a Functional System changes state. *Rationale: If a Functional System deactivates itself this likely requires some external action.*	F

ID	Source	Requirement	Scope
R015	O MSCP-37	The MPC shall be able to dynamically map relevant Functional System components during operation to other Physical Computing Elements in response to hardware failures. *Rationale: This is intended to mitigate hardware failures.*	O
R016	O# MSCP-94 & 92	The MPC shall provide a management interface that allows activation and deactivation of Physical Computing Elements. *Rationale: The operators shall be able to add and remove physical computing element, e.g., hardware.*	TS
R017	O MSCP-93	For a Functional System it shall be transparent on which Physical Computing Elements its components are deployed to. *Rationale: The functionality within the Functional Systems does not need to know where it is deployed. SRACs have to be satisfied, of course.*	F
R018	O MSCP-41	The MCP shall be able to run multiple Functional Systems concurrently (at the same time). *Rationale: Hardware with a multicore processor architecture is commonly available today. It allows running Functional System components side-by-side sharing resources.* *Satisfies: MPC-P06*	F
R019	O# MSCP-39	Where application-level platform independence is used, the MPC shall be able to assign deterministic execution behaviour to Functional System components in regular scheduling intervals, based on configuration. *Rationale: Determinism is paramount in a safety critical environment. Therefore, each relevant Functional System component must have a defined execution period (scheduling: time interval).*	F
R020	O# MSCP-97	Where application-level platform independence is used, the MPC shall be able to assign deterministic execution behaviour to relevant Functional System components triggered by an one-shot-timer event. *Rationale: Besides a regular scheduling interval, additional execution of a Functional System component might be needed, therefore one-shot-timer triggered execution shall be provided by the platform.*	F

ID	Source	Requirement	Scope
R021	O# MSCP-96	Where application-level platform independence is used, the MPC shall be able to schedule Functional System components for execution triggered by an event such as the receiving of a message by a Functional System component. *Rationale: Enable reaction to explicit events prior to the next regular scheduled start.*	F
R022	O# MSCP-40	Where application-level platform independence is used, the MPC shall be able to execute Functional System components for a guaranteed execution budget, which shall be defined in the configuration. *Rationale: Determinism is paramount in a safety critical environment. Therefore, relevant Functional System components must have a guaranteed execution e.g., to be scheduled for configured number of time ticks.* *Remarks: Configuration for all different scheduling paradigms need to be provided, e.g., execution time guarantees might differ between interval and event triggered scheduling and as well between on-board and trackside applications needs.*	F
R023	O MSCP-35	The MPC shall be able to provide strict deadlines and maximum tolerable jitter for deterministic scheduling of Functional System components. *Rationale: Real-time computing is key for designing and/or developing predictable, safe CCS functional applications.*	O
R024	O MSCP-108	Where application-level platform independence is used, the MPC shall detect and handle errors according to EN 50129:2018 (with both the definition of “error” and the handling of these according to EN 50129:2018). *Rationale: This is required to fulfil the norm EN 50129:2018.*	F
R025	O MSCP-118	Where application-level platform independence is used, the MPC shall monitor whether a Functional System component is able to conclude processing within a defined time period. *Rationale: It is important that Functional System components can conclude on processing, e.g., incoming messages within a certain CPU resource. The platform has to monitor this to be able to potentially react.*	F

ID	Source	Requirement	Scope
R026	O MSCP-119	Where application-level platform independence is used, the MPC shall inform the Functional System component if one or multiple of its components have (once or multiple times) not been able to conclude processing in the allocated processing time and take action if configured so. *Rationale: Only the Platform knows about the exceeding and informs the Functional Actor that it might have taken appropriate actions.*	F
R027	O MSCP-106	Where application-level platform independence is used, when there are multiple replicas of the same Functional Actor, the MPC shall ensure that individual replicas process the same messages. *Rationale: This is required, as otherwise it could not be guaranteed that the Replicas yield the exact same output.*	F
R028	O# MSCP-44	Where application-level platform independence is used, MPC shall provide standardised mechanisms for communication between Functional Systems components. *Rationale: Functional Systems and their respective Functional Actors shall be able to exchange data with each other using a standardised message paradigm.*	F
R029	O MSCP-49	~~Where application-level platform independence is used, the MPC shall provide the ability to Functional Actors to access local inputs and outputs.~~ *Rationale: In case there are local I/Os directly connected to the hardware of the MPC, these must be made accessible to the relevant Functional System components.* *Remark: Some on-board Computing Platforms need to supply up to SIL4 outputs (e.g., Emergency Brake) and it is the responsibility of the MPC implementation to realise such functional safe I/Os.*	X
R030	O MSCP-51	The MPC shall provide the ability to Functional Systems to communicate via communication networks. *Rationale: Functional Systems deployed on different Physical Computing Elements need to communicate with each other.* *Remark: It is assumed that the Computing Platform needs to provide network access via commonly used network protocols as e.g., TCP/IP.*	F

ID	Source	Requirement	Scope
R031	O MSCP-55 & 53	The MPC shall allow time synchronisation with an external time server via standard protocols. *Rationale:* *Time synchronisation aims to coordinate otherwise independent clocks. Even when initially set accurately, real clocks will differ after some amount of time due to clock drift, caused by clocks counting time at slightly different rates. The usage of standardised protocols ensure compatibility, interoperability, simplify product development and speed up time-to-market.*	F
R032	O MSCP-60	The MPC shall include a monitoring and diagnostics interface accessible locally and via remote connection. *Rationale:* *In order to analyse the system behaviour and performance during development, test and operation, a diagnostics interface is needed between Computing Platform deployments and the MDCM (Monitoring, Diagnostics, Configuration, and Maintenance).*	F
R033	O# MSCP-59	Where application-level platform independence is used, the MPC shall support monitoring of the execution of Functional System components, for instance by capturing KPIs related memory usage, processor load, etc. *Rationale:* *To support fault analysis as well as to monitor proper operation of deployed system.*	F
R034	O MSCP-84	The MPC shall be able to provide logging and tracing information to an external entity. *Rationale:* *Logging and tracing are critical when analysing system behaviour and faults. Having a unified logging and tracing concept dramatically simplifies the analysis.*	F
R035	O MSCP-67 & 66	The MPC shall provide safe and secure mechanism to update the run-time environment locally and remotely. *Rationale:* *The ability of updating the platform software is essential. To minimize maintenance cost, the normal update deployment mechanism shall be remotely (e.g., over-the-air) with no need for physical presence of any maintenance personnel on site (e.g., on the train). In case remote updates fail for any reason, it must be possible to perform local updates with physical access to the MPC. Updates shall be uploaded via industry standard interfaces.* *Remark:* *Remote updates must not affect the proper operation of MPCs and might need explicit planned scheduling to be applied.*	F

ID	Source	Requirement	Scope
R036	O MSCP-68 & 63	The MCP shall provide safe and secure mechanisms to update the MPC configuration locally and remotely. *Rationale:* *The ability of updating the platform configuration is essential. To minimize maintenance cost, the normal update deployment mechanism shall be remotely (e.g., over-the-air) with no need for physical presence of any maintenance personnel on site (e.g., on the train). In case remote updates fail for any reason, it must be possible to perform local updates with physical access to the MPC. Updates shall be uploaded via industry standard interfaces.* *Remark:* *Remote configuration updates must not affect the proper operation of MPCs and might need explicit planned scheduling to be applied.*	F
R037	O MSCP-69 & 64	The MPC shall provide safe and secure mechanisms to update Functional Systems as a whole or their selected individual components locally and remotely. *Rationale:* *The ability of updating Functional Systems as a whole or some of their components is essential. To minimize maintenance cost, the normal update deployment mechanism shall be remotely (e.g., over-the-air) with no need for physical presence of any maintenance personnel on site (e.g., on the train). In case remote updates fail for any reason, it must be possible to perform local updates with physical access to the MPC. Updates shall be uploaded via industry standard interfaces.* *Remark:* *Software updates of the Functional System are performed by the platform, but software updates of entities controlled by the Functional System are in the responsibility of the Functional System itself (e.g., update of field elements).*	F
R038	O# MSCP-112	The MPC shall leverage existing specifications for interfaces where feasible. *Rationale:* *Proven interface specifications offer the necessary maturity level for the MPC. Also, existing implementations for platform components can potentially be reused.*	F
R039	O# MSCP-113	The MPC shall minimize the number of function calls that are part the interfaces. *Rationale:* *A reduced set of function calls eases certification, acceptance and potentially portability.*	F

ID	Source	Requirement	Scope
R040	O# MSCP-121	The MPC shall define interfaces in a way that is evolvable over time and always enables backward-compatibility. *Rationale: It is expected that the interfaces can evolve over time. Future evolved versions are expected to be decently backward compatible to make sure that existing applications can be integrated in MPCs implementing future versions.*	F
R041	O# MSCP-114	The MPC shall minimize the number of differences in the interface specifications for on-board and trackside environments. *Rationale: A common platform simplifies the specification, and thus portability and reusability of common elements between the different environments.*	F
R042	O# MSCP-73	The MPC shall provide capabilities for Functional Systems to obtain presence and state information of other Functional Systems. *Rationale: If there are dependencies between Functional Systems, they need to know each other's state (e.g., active, inactive, degraded) and react if necessary.*	F
R043	O MSCP-54	Where application-level platform independence is used, the MPC shall provide a mechanism to Functional System components for obtaining the current replica-synchronized time. *Rationale: Functional System components need consistent, replica-synchronised time information which is exactly the same for all replicas and can be used to create output that is voted on.* *Remark: This is important if the time stamp has an impact on any (voted) output of a Functional System component.*	F
R044	O MSCP-98	The MPC shall provide a mechanism to Functional System components for obtaining the current un-synchronised time. *Rationale: Functional System components need un-synchronised time information e.g., replica for specific logging.* *Remark: "Unsynchronised time" corresponds to the time at the point when a Functional System component requests this (and for which different components of the same Functional System may obtain a different result).*	F
R045	O MSCP-102	Where application-level platform independence is used, the MPC shall complement messages with timestamps. *Rationale: Timestamps are important, so that receiving Functional System component can check whether and how strongly received messages are outdated and possibly take appropriate action (i.e., either discard such messages or take other action).*	F

ID	Source	Requirement	Scope
R046	O MSCP-105	Where application-level platform independence is used, the MPC shall inform the appropriate Functional System component if it has not been able to conclude processing within a defined time period (once or multiple times, as configured) and (if configured) shut down the Functional System component. *Rationale: If a Functional System component is not able to conclude processing within a defined time period (once or multiple times), it either has to scale down its load (e.g., by shifting tasks to other Functional Systems, where possible), or the platform has to shut it down.*	F
R047	O# MSCP-76	Where application-level platform independence is used, the MPC shall provide a standardized communication mechanism, to exchange messages between Functional System components. *Rationale: A standardized communication mechanism is needed in order to abstract safe and secure communication and to enable a transparent encapsulated safety and fault tolerance mechanism, realized by the platform.*	F
R048	O# MSCP-116	Where application-level platform independence is used, the MPC shall provide a communications method with location transparency to the Function System components. *Rationale: Addressing and routing should be transparent to the applications, and thus communication messages routed to the appropriate recipients without needing to know their physical execution environment and location.*	F
R049	O# MSCP-115	Where application-level platform independence is used, the MPC shall provide a communications method with replication transparency to the Function System components. *Rationale: The appropriate Functional Systems components have to be agnostic towards the fact that they might be replicated. All complexities with communications coming from replicated execution has to be handled by the platform.*	F
R050	O# MSCP-123	Where application-level platform independence is used, the MPC shall allow communications only by its own communication mechanisms. *Rationale: Side channel communication not using the platform methods has to be avoided. Only this way, standardized and safe communication can be established, and transparent replication implemented.*	F

ID	Source	Requirement	Scope
R051	O# MSCP-101	Where application-level platform independence is used, the MPC shall supervise configured maximum message delivery times. *Rationale:* *Functional System components depend on reliable, reasonably deterministic communication with other Functional System components. It is hence required that the platform supervises defined maximum message deliverable times in, e.g., the scheduling of relevant Functional System components.*	F
R052	O# MSCP-81	Where application-level platform independence is used, the MPC shall ensure the correct ordering of all messages sent and received by Functional System components. *Rationale:* *Functional System components can rely on the correct message distribution order without the need to implement order checking logic.*	F
R053	O MSCP-80	Where application-level platform independence is used, the MPC shall conform to EN 50159. *Rationale:* *In terms of safe communication, EN 50159 provides guidelines which shall be followed to ensure that we have a common base for communication requirements.* *Remark:* *Issues are identified by the platform (e.g., through the usage of message sequence numbers or some other platform-specific mechanism).*	F
R054	O# MSCP-117	Where application-level platform independence is used, the MPC shall supplement messages with their time of creation. *Rationale:* *Time stamping is needed, so that receivers are able to determine how old messages are, and whether they should still be processed or discarded, etc.* *Remark:* *This might require the notion of synchronized platform clocks (also among distributed platforms, see also R043) at least to the extent/granularity (e.g., on the order of tens of ms) that is required to detect outdated messages. When exactly the time stamping happens needs to be discussed.*	F
R055	O MSCP-111	The MPC shall provide a bi-directional interface to exchange diagnostics information with Functional System components. *Rationale:* *Functional System components can use the interface, e.g., to receive diagnostics information or to report diagnostic information to the platform.*	F

ID	Source	Requirement	Scope
		Remark: *Diagnostic information could for instance comprise the health status of the platform or a Functional System component.*	
R056	O MSCP-124	The MPC shall provide an interface towards Functional System components for logging and tracing. *Rationale:* *Applications have the need to provide log and trace data to the platform.*	F
R057	O# MSCP-86 & 87	The MPC shall allow configuration of different logging and tracing levels and categories per platform and on a Function System component level. *Rationale:* *Depending on the required information, it is important to be able to enable logging and tracing only for certain Functional System components and not for the entire system.*	F
R058	S SPT2CE-1520	The Physical Computing Element shall be based on COTS components. *Rationale:* *Leveraging COTS hardware provides several benefits, including cost-effectiveness, readily available components, and ease of integration.*	F
R059	S# SPT2CE-1524	The SRACs of the Functional System shall not limit the use of shared hardware resources. *Rationale:* *To maximize the efficiency and flexibility of hardware usage it is essential to be able to aggregate Functional Systems of various suppliers on the same Physical Computing Element(s).* *Satisfies:* *MPC-P06*	F
R060	S SPT2CE-1533	The safety environment shall identify incorrect deployment of safety critical Functional System compartment. *Rationale:* *It is imperative that safety-critical functions employing composite safety, such as replication and voting, are executed on distinct hardware devices.*	F
R061	S SPT2CE-1529	The safety environment shall not restrict mixed criticality on a physical computing element. *Rationale:* *To enable the wide range of applications with different critical levels to coexists and to attain resource efficiency, flexibility, improved system utilization, optimized performance, and scalability.*	F

ID	Source	Requirement	Scope
R062	S SPT2CE-1528	The safety environment shall not restrict the creation/initialization of new compartments during runtime. *Rationale: To support dynamic configuration the system shall support the aggregation/deployment of Functional Systems during runtime.*	F
R063	S SPT2CE-1534	The safety environment shall support start/stop of Functional System compartments on another physical computing element during runtime. *Rationale: To replace defective HW it is essential that the safety environment shall support Functional System compartment management.*	F
R064	S# SPT2CE-1546	The virtual computing element shall support the remote restart of an individual FS compartment at runtime. *Rationale: This will allow to automate the recovery of FS during runtime.*	F
R065	S# SPT2CE-1549	The safety environment shall support the synchronization of restarted/updated compartments. *Rationale: To synchronize the safe applications replica with other running replicas after update/recovery.*	F
R066	S SPT2CE-1550	Communication between functional systems (I0) shall be realized with 2 redundant channels for availability. *Rationale: In order to support safety-critical communication, fulfilling availability requirements the FS system shall have redundant and independent communication channel.*	F
R067	S SPT2CE-1531	The OI shall provide APIs and tools to orchestrate the FS Compartments. *Rationale: To facilitate the remote deployment of FS it is essential to implement mechanisms that automate, manage, and coordinate Functional Systems in compliance with their certification requirements.*	F
R068	S SPT2CE-1553	The virtualisation environment shall guarantee the assigned CPU resources (cores, memory, memory bandwidth, network bandwidth, network latency) for a FS continuously (7 days /24 hours / 60 minutes / 60 seconds) without any influence or dependency to existence and behaviour of other FS aggregated on same computing element. *Rationale: The VE must provide the committed resources to all FS compartments and ensures no interference between the virtual computing elements on same physical computing element.*	F

ID	Source	Requirement	Scope
R069	S SPT2CE-1530	The configuration of the virtualization environment shall comply with the FS deployment rules. *Rationale: To ensures the function and availability of the intended FS, it is crucial to adhere to its deployment rules.*	F
R070	S SPT2CE-1540	The virtualization environment shall provide standard OI functionalities. *Rationale: Having a common set of OI functionalities for operating all Functional System(s) offers benefits such as resource optimization, scalability, high availability, automation, and cost efficiency.*	F
R071	S SPT2CE-1542	The virtualization environment shall be based on COTS solutions. *Rationale: As there are already a wide range of COTS virtualization solutions available, developing a new one for railways is prohibitive due to complexity and cost.*	F
R072	S SPT2CE-1541	Different virtualization approaches shall be allowed. *Rationale: There are several types of virtualizations approaches available in different domains, each with its own benefit. Therefore, the virtualization solution could allow different implementation approaches as long as a standard Orchestration Interface (OI) is provided.*	F
R073	S# SPT2CE-1537	~~The orchestration interface (OI) implementation shall not bind to a specific approach/programming language.~~ *Remark: In the MPC, this interface is expected to be connected to the Platform Management, thus only available internally. As such, no guidance on approaches and programming languages is necessary. The requirement has been removed.*	X
R074	S SPT2CE-1536	The Virtualization environment shall provide remote orchestration. *Rationale: To enable the efficient management, scalability, cost reduction, it is essential to have a centralised FS management.* *Remark: In the MPC, this interface is expected to be connected to the Platform Management.*	F

ID	Source	Requirement	Scope
R075	S SPT2CE-1545	The Virtual Environment shall allow uninstallation of individual compartment deployed on a virtual computing element without interrupting neighbouring compartments on the same physical hardware. *Rationale: To ensure the safety and availability of other Functional system compartments running on the same physical computing element.*	F
R076	S SPT2CE-1544	The Virtualization Environment shall support remote creation of virtual computing elements without interfering with already running virtual computing elements on the shared physical hardware. *Rationale: To enable remote dynamic configuration of virtual computing elements.*	F
R077	S SPT2CE-1543	The Virtualization Environment shall support remote deletion of virtual computing element without impacting other running virtual computing element on the shared physical hardware. *Rationale: To enable remote dynamic configuration of virtual computing elements.*	F
R078	S SPT2CE-1535	The Virtualization Environment shall provide full hardware abstraction. Changes in the underlying COTS HW may not have any impact to the FS running on the virtualisation environment. *Rationale: Virtualization Environment must be compatible to all the hardware architecture such as x86, ARM, PowerPC, and others to support seamless integration of FS.* *Remark: There is no requirement towards full hardware emulation across different CPU architectures.*	F
R079	S SPT2CE-1551	The Virtualization Environment shall support to do updates of the virtualisation environment computing-element-wise "one after the other" without affecting the virtualisation environment on the other virtual computing elements. *Rationale: This is necessary to update the virtualisation environment (e.g. due to IT-sec patches) during runtime of the FS.*	F
R080	S# SPT2CE-1552	The Virtualization Environment shall provide (backward) compatibility at the configuration interface. *Rationale: A new version of the virtualisation environment shall not have any impact on the FS related configuration data. The configuration of the virtual machine (resources, communication interfaces, etc.) shall not change.* *Remark: In the MPC, the changes in the actual interface would be adapted in the Platform Management.*	F

ID	Source	Requirement	Scope
R081	S SPT2CE-1547	The Virtualization Environment shall provide the diagnostic interface to monitor the virtual computing element. *Rationale:* *The diagnostic will allow to monitor the health of virtual computing elements and may detect SW crashes and enable automatic recovery.* *Remark:* *This is connected to the Platform Management in MPC.*	F
R082	S SPT2CE-1554	The Virtualization Environment shall provide detailed predictive diagnosis about the health state of the physical computing element(s). *Rationale:* *The predictive diagnosis will allow to monitor the health of physical computing element and may detects HW faults earlier.*	F
R083	S SPT2CE-1548	The Virtual Environment shall provide mechanism to ensures the correct deployment of FS compartments. *Rationale:* *To ensure the safety requirements such as to run each replica of FS compartment on distinct physical computing element.*	F
R084	+	When non-safe software parts are changed, the MPC shall ensure that there is no impact on the safe software parts. *Rationale:* *Changing resp. updating non-safe parts (e.g., Basic Integrity applications or parts of the RTE/VE, security updates, etc.) must not create any situation where previously reached conclusions on the freedom of interference towards safe software parts are invalidated. Meaning that changing or updating non-safe parts does not lead to re-certification of safe parts.* *Satisfies:* *MPC-P02*	F

Table 10: Selected Modular Platform Requirements

APPENDIX B HLPI REQUIREMENTS

Column	Meaning
Source	S: SP CE Domain OAS [15] (released to Mirror Group on 2024-05-07) +: new #: (heavily) modified or rewritten
Allocation	FS: Functional Systems VE: Virtualization environment SE: Safety Environment SS Diag: Shared Services for Diagnosis MPM: Modular Platform Management Network Diagnosis
Scope	F: full (all target environments) OB: On-board required, trackside optional TS: Trackside required, on-board optional O: optional for all environments X: not in scope for work package 26

ID	Source	Allocation	Requirement	Scope
REQ-HLPI-1	+ 6.2.4 6.7.1	SE	Each solution of a Safety Environment shall define the own safety concept in a way which allows the usage of a non-safe VE. The SE itself must identify if the safety related parts of the FS are not running in the required time range or performance. Each miss-behaviour of the VE as e.g. wrong scheduling of the individual FS software parts may not have any impact onto the safety of the FS. The SE can't rely on the behaviour of the VE, means the SE must identify each misbehaviour of VE and react safe. Information about misbehaviour of the VE must be provided as diagnosis by FS. *Rationale: For aggregation of several FS compartments on a common non-safe VE its essential that the VE shall not have any dependency to safety.*	F
REQ-HLPI-2	+ 6.3	VE	The VE shall provide the mapping of CPU cores exclusively to VCE. *Rationale: For aggregation of several FS compartments on common VE on the same hardware its essential that the VE provides a stable runtime behaviour of each FS compartment by exclusive core usage.*	F
REQ-HLPI-3	+ 6.3	VE	The CPU performance provided by the mapped CPU resources must be guaranteed for every timepoint during the runtime of an FS Compartment. *Rationale: Variations or instabilities in the provided CPU performance will be identified by the SE and will directly lead to reduced availability as consequence of reactions by the SE. Example: If an individual application replica does not react in the required time then this will be evaluated as a misbehaviour of the application replica and this leads to reduced availability (as e.g. running mode reduced from 2oo3 to 2oo2).*	F

ID	Source	Allocation	Requirement	Scope
REQ-HLPI-4	+ 6.3	VE	The installation of additional FS Compartments (of other FS) in additional VCEs on the same Virtualization Environment Instance must not have any impact on the guaranteed CPU performance (cores) for running FS Compartments. *Rationale: The stability of the CPU resources is essential for independent handling of individual FS running aggregated in parallel on same VE.*	F
REQ-HLPI-5	+ 6.4.1	VE	The individual VCE configurations of FS compartments shall be modular and independent. Each FS Compartment shall have its own configuration for the VCE. Adding or deleting of FS compartments onto the VE instance must not have any impact on the VCE configuration of the other FS compartments. *Rationale: The independency of VCE configuration is essential for independent handling of individual FS running aggregated in parallel on same VE.*	F
REQ-HLPI-6	+ 6.4.2	VE	The virtualization environment shall provide defined and stable user interfaces for the configuration of the usage by FS compartments. A new version of the VE may not have any impact onto the VE Configuration of the FS compartment. Each change in the user interface for the VE configuration shall be compatible in such a way that existing VE configs (of already running system) can be used furthermore. *Rationale: The independency of VCE configuration is essential for independent handling of individual FS running aggregated in parallel on same VE.*	F
REQ-HLPI-7	+ 6.6	SE	Safety concept of each SE solution must be basically independent from the processor instruction set to be able to change the CPU architecture without impact to the safety concept. *Rationale: For future proofness in context of usage of COTS hardware it's essential to be able to change the processor instruction set without impact onto the basic safety concept.*	F

ID	Source	Allocation	Requirement	Scope
REQ-HLPI-8	+ 6.6	VE	The VE shall support “incompatibilities in detail” in context of hardware spare handling. The usage of a hardware spare part may not have any impact onto the FS compartments. *Rationale: Ordering of the same hardware does not guarantee that the exact same hardware is delivered with 100% compatibility to the software.* *HW internal changes of details are possible.*	F
REQ-HLPI-9	+ 6.6	VE	The VE shall support the usage of different variants of hardware – provided by different vendors – at in parallel at the same time. Needed adaptions within the VE for usage of a new hardware variant may not have any impact on the VE instances with already running FS compartments. *Rationale: It’s essential for efficient handling of COTS hardware to avoid the impact on already running FS compartments.*	TS
REQ-HLPI-10	+ 6.7.1.x 6.7.1.1 6.7.1.2 6.7.1.3 6.7.1.4 6.7.1.5	VE	The virtualization environment shall provide a “native running hardware access” (NHA) functionality to provide needed information from the physical hardware in a reliable way to the SE. A first set of identified information is: - Unique identification of the physical hardware device - Core pinning - steady clock input source from the physical hardware - CPU and/or other temperature of the physical hardware - Voltage information *Rationale: The details regarding the needed data depend on the SE solution.* *The “native running mode” of NHA is essential to achieve the needed reliability of the data required by SE solutions. Reliability in such a way that argumentation “data can’t be influenced systematically” can be done for up to SIL4.*	F

ID	Source	Allocation	Requirement	Scope
REQ-HLPI-11	+ 6.7.1.x	SE	The SE itself must ensure that the FS compartments are deployed in correct way running on different physical computing elements. In case of a false deployment (FS compartments running on the same physical computing element) the SE must identify the failure and react in a safe way. *Rationale: Usage of non-safe SW as VE and for orchestration is not reliable. By this the SE must check the correct distribution on different physical computing elements.*	F
REQ-HLPI-12	+ 6.7.2	SE	The SE shall realize safety mechanism to ensure the consistency of the safety related SW parts of an up to SIL4 FS. *Rationale: by usage of non-safe SW for VE, operating system and orchestration software it's not guaranteed that stopping, deleting and starting of safety related software is successful.*	F
REQ-HLPI-13	+ 6.8	VE	3rd party supplier of VE has to consider IEC 62443 to provide certification as needed. *Rationale: Fulfilment IEC 62443.*	F
REQ-HLPI-14	+ 6.9.1	VE	The VE shall guarantee a perfect stable behaviour in context of runtime and reaction time of the SW parts within FS compartments. *Rationale: Variations or instabilities in the provided CPU performance will be identified by the SE and will directly lead to reduced availability as consequence of reactions by the SE. Example: If an individual application replica does not react in the required time then this will be evaluated as a misbehaviour of the application replica and this leads to reduced availability (as e.g. running mode reduced from 2oo3 to 2oo2).*	F
REQ-HLPI-15	+ 6.9.2	SE	The SE shall support to repair a failed FS compartment during the operational phase of the FS, synchronization of the repaired FS compartment with the running FS compartments shall be done automatically by the SE to achieve full redundancy again. *Rationale: Highest FS availability in context of SW maintenance: avoid stopping of the FS due to repair of an individual failure.*	F

ID	Source	Allocation	Requirement	Scope
REQ-HLPI-16	+ 6.9.3	VE	The VE shall support the mapping of VCEs to virtualized ethernet adapters and the alignment of virtualized Ethernet adapters to physical Ethernet cards of the PCE. *Rationale: Flexible usage of Ethernet communication without dependency to FS internal configurations.*	F
REQ-HLPI-17	+ 6.4.2 6.13.1	VE	The VE shall provide for new VE versions backwards compatibility of the VE configuration interface for FS configuration. *Rationale: It must be avoided that a SW update of the VE leads to impact on the VCE Configs of the FS Compartments running above.*	F
REQ-HLPI-18	+ 6.9.4	FS	The FS shall allow to update basic integrity SW parts as e.g. the IT security mechanism individually FS compartment-wise “one after the other” during operational phase of the FS. *Rationale: Highest FS availability in context of SW maintenance: avoid stopping of the FS due to installation of an IT-security patch.*	F
REQ-HLPI-19	+ 6.9.4	VE	The VE shall allow to update the VE software hardware-wise “one after the other” during operational phase of the FS running above. *Rationale: Highest FS availability in context of SW maintenance: avoid stopping of the FS due to installation of an IT-security patch.*	TS
REQ-HLPI-20	+ 6.9.4	PM	The Platform Management must handle the dependency to update “one-after-the-other” during runtime of the FS. *Rationale: Highest FS availability in context of SW maintenance: avoid stopping of the FS due to installation of an IT-security patch.*	TS

ID	Source	Allocation	Requirement	Scope
REQ-HLPI-21	+ 6.11	PM	The Platform Management shall collect the diagnosis data of the VE and 3rd party SW (as e.g. for COTS hardware diagnosis) and provide this data via interface I1 to the Shared Services Diagnosis. *Rationale: Standard solutions for VE / COTS diagnosis will not consider the interface I1 Diagnosis. By this a "protocol-conversion" is necessary to provide the diagnosis data in the required format.*	F
REQ-HLPI-22	+ 6.11	PM	The Platform Management shall process a root cause analysis for the FS state and initiate necessary maintenance activities automatically. *Rationale: It must be avoided that a SW update of the VE leads to impact on the VCE Configs of the FS Compartments running above.*	F
REQ-HLPI-23	+ 6.11.2	FS	The FS shall provide diagnosis data about the own health state via the interface I1 FS Diagnosis to the Shared Services for diagnosis. *Rationale: Shared Services for diagnosis are data sink for all kind of diagnosis data.*	F
REQ-HLPI-24	+ 6.11.2	FS	The FS shall provide diagnosis data about the own health state via the interface I1 FS Diagnosis to the Platform Management. *Rationale: Platform Management is the data sink for diagnosis data which is relevant for the handling of FS compartments running in VCEs on VPEs.*	F
REQ-HLPI-25	+ 6.11.2	PM	The Platform Management must handle the relationship "one FS consists of several individual FS compartments which provide own diagnosis data". Diagnosis data of the individual compartments must be aggregated to an overall state of the FS and this state must be provided via the interface I1 Diagnosis to the Shared Services Diagnosis. *Rationale: Shared Services for diagnosis shall get a defined FS state (independent from details about the solution that the FS is running in several compartments).*	F

ID	Source	Allocation	Requirement	Scope
REQ-HLPI-26	+ 6.11.3	VE	The VE shall provide diagnosis date about the VE itself and about the underlying physical hardware to the PM. The data must be provided by the VE management tool to the Platform Management. *Rationale: Platform Management is data sink for all diagnosis data relevant for the handling of the FS compartments within VCEs running on PCEs.*	F
REQ-HLPI-27	+ 6.11.3	PM	The Platform Management must forward the diagnosis data about the VE and VCEs to the Shared Services for diagnosis. For this the interface I1 Diagnosis must be considered. *Rationale: Shared Services for diagnosis are data sink for all kind of diagnosis data.*	F
REQ-HLPI-28	+ 6.11.4	VE	Information about the health state of the virtual computing elements and physical computing elements shall be provided by the VE or even additional dedicated diagnosis software provided by 3rd party. *Rationale: Platform Management is data sink for all diagnosis data relevant for the handling of the FS compartments within VCEs running on PCEs.*	F
REQ-HLPI-29	+ 6.11.4	VE	A dedicated software for diagnosis of the physical computing elements shall provide diagnosis data to the Platform Management. *Rationale FS running in VCEs is not able to identify details about the detailed states of PCEs, FS only reacts in case of failures within the PCEs. By this a dedicated diagnosis software for the PCEs is necessary.*	F
REQ-HLPI-30	+ 6.11.4	PM	The Platform Management must forward the diagnosis data about the physical computing elements to the Shared Services for diagnosis. For this the interface I1 Diagnosis must be considered. *Rationale: Shared Services for diagnosis are data sink for all kind of diagnosis data.*	F
REQ-HLPI-31	+ 6.11.5	Network Diagnosis	The Network Diagnosis shall provide the diagnosis data about the network to the Platform Management.	F

ID	Source	Allocation	Requirement	Scope
			Rationale: Platform Management is data sink for all diagnosis data relevant for the handling of the FS compartments within VCEs running on PCEs. Network is relevant for the FS internal communication between the FS compartments. By this the network diagnosis is necessary for root cause analysis about FS state.	
REQ-HLPI-32	+ 6.11.5	PM	The Platform Management must handle the relationship between FS compartments and the belonging network communication. Diagnosis data related to the network communication shall be evaluated with be belonging FS compartments in context of root cause analysis. *Rationale: Platform Management is data sink for all diagnosis data relevant for the handling of the FS compartments within VCEs running on PCEs. Network is relevant for the FS internal communication between the FS compartments. By this the network diagnosis is necessary for root cause analysis about FS state.*	F
REQ-HLPI-33	+ 6.12.1	SE	The SE shall support to update the own operating system (with IT-security layer) within the FS compartment compartment-wise “one after the other” to install IT-security patches during runtime of the FS. *Rationale: IT-security patching during operational phase of the FS.*	F
REQ-HLPI-34	+ 6.12.1	VE	The VE shall support to update the VE instances (with IT-security layer) hardware-wise “one after the other” with a new version of the VE instance SW to install IT-security patches during runtime of the FS. After the VE instance update the FS compartments shall be started automatically to achieve full redundancy, e.g., to achieve 2oo3 again. There must not be the dependency to install an update of the VE on all PCEs at same timepoint, because this would lead to stop of all FS compartments. *Rationale: IT-security patching during operational phase of the FS.*	TS
REQ-HLPI-35	+ 6.12.1	VE	The VE shall allow to replace a physical computing element by another physical computing element without impact onto the running VE instances. *Rationale: Replacing individual PCEs during operational phase of the FS.*	TS

APPENDIX C ALPI REQUIREMENTS

Column	Meaning
Source	S: SP CE Domain OAS [15] (released to Mirror Group on 2024-05-07) +: new #: (heavily) modified or rewritten
Allocation	RT: runtime CF: configuration OP: offline process (data preparation, certification)
Scope	F: full (all target environments) OB: On-board required, trackside optional TS: Trackside required, on-board optional O: optional for all environments X: not in scope for work package 26

ID	Source	Allocation	Requirement	Scope
REQ-ALPI-01	7.3.2.1	RT, CF	The ALPI shall provide an **independent** interface towards MPC. *Rationale:* *The development of a Functional Application shall be independent from the MPC platform. ALPI's interface shall be able to hide different MPC implementations based on different HW and SW. Independence shall be based on a common set of ALPI, based on a shared architecture and a common platform behaviour,* *Satisfies:* *MPC-P01, MPC-P02, MPC-A03, R17*	F
REQ-ALPI-02		RT, CF	The ALPI shall provide a **standard** interface. *Rationale:* *The development of a Functional Application shall be based on a standard RTE interface. The standard interface should allow re-use and easy integration of the Functional Application in the case of different RTE suppliers. ALPI shall provide a standardised language to specify the application's deployment-configuration.* *Satisfies:* *MPC-P01, MPC-P05, MPC-P07,*	F
REQ-ALPI-03		RT, CF	The ALPI shall provide a **flexible** interface. *Rationale:* *The ALPI interface shall allow maximum flexibility in the use of all available RTE services and COTS SW, especially in the case of Non-Safety-Related Functional Applications that shall be developed with maximum flexibility to take full advantage of the evolution of ICT and OT technologies. Constraints that limit the use of products with new technologies developed in the COTS environment should be avoided. For Safety-Related Functional Applications ALPI shall provide implicit restrictions, selected via configuration, transparently implemented by runtime services, imposed through adoption of common standard models.* *Satisfies:* *MPC-P01, MPC-P05*	F
REQ-ALPI-04	7.1	RT, CF	The ALPI shall reduce the **complexity** of Functional Application development. *Rationale:* *The ALPI interface shall allow the development of Functional Application in which the complexity of the mechanisms needed to ensure communication, safety and security are not directly managed by Functional Engineer. ALPI shall minimize the number of services*	F

ID	Source	Allocation	Requirement	Scope
			the number of differences in the interface specifications for on-board and trackside environments. *Satisfies:* *MPC-P01,MPC-P03, R028, R039, R040, R041*	
REQ-ALPI-05		RT	The ALPI shall provide **compatibility.** *Rationale:* *The ALPI interface shall be developed to assure backword compatibility during the future evolution of the ALPI interface.* *Satisfies:* *MPC-P04, MPC-P05, R40*	F
REQ-ALPI-06		RT	The ALPI shall provide **transparency.** *Rationale:* *The ALPI interface shall provide services that implement mechanisms/ protocols/ needed to achieve transparency of location, communication, safety, security.* *Satisfies:* *MPC-P01,MPC-P03, R004, R048, R049.*	F
REQ-ALPI-07	7.3.2.1.1 7.3.2.2	RT	The ALPI shall provide a common standard **development** model. *Rationale:* *The development of a Functional Application shall be based on the "Functional Application Task" concept. FAT is the basic component of a Functional Application. ALPI shall provide all services necessary to the creation, configuration, communication, scheduling, aggregation of FAT.* *Satisfies:* *MPC-P01,MPC-P03,*	F
REQ-ALPI-08			(removed)	X
REQ-ALPI-09		RT, CF	The ALPI shall provide a **configurable set** of services to implement **Non-Safe**, **Basic Safety** Integrity and Safety Integrity Level **SIL1-SIL4** Functional Application. ALPI shall allow restriction of the set by means of configuration. *Rationale:* *ALPI shall provide a selected set of services depending on SIL of the Functional Application.*	F

ID	Source	Allocation	Requirement	Scope
			Satisfies: *MPC-P03*	
REQ-ALPI-010			(removed)	X
REQ-ALPI-011		RT, CF	The ALPI shall provide a **restricted set** of services to implement **Basic Safe** Functional Application. *Rationale:* *Basic Integrity Level Functional Application shall be developed using services compliant with Basic Integrity Level requirements as specified in EN 50xxx. ALPI shall allow restriction of the set by means of configuration.* *Satisfies:* *MPC-P03,MPC-G02 , R01*	F
REQ-ALPI-012		RT, CF	The ALPI shall provide a **restricted set** of services to implement **SIL1,.. SIL4** Functional Application. *Rationale:* *SIL1,.. SIL4 Functional Application shall be developed using ALPI services compliant with CENELEC standard as specified in EN 50xxx. In case of SIL4 FA, ALPI shall allow the implicit enabling of the transparent mechanisms that implement composite safety. ALPI shall allow restriction of the set of services by means of configuration.* *Satisfies:* *MPC-P03, R001, R061*	F
REQ-ALPI-013	7.3.2.1		The ALPI shall allow the **aggregation** of Functional Applications with **different SIL** *Rationale:* *aggregation of mixed SIL Functional Application.* *Satisfies:* *MPC-P03, R002, R061*	F
REQ-ALPI-014	7.3.1.3	RT	The ALPI shall provide runtime **Security** functions in the scope of the Functional Application *Rationale:* *ALPI shall provide security services related to PKI management, authentication management, cryptographic verification/validation inside FAT. Security related to external communication is managed end-to-end with TLS at network level and it is not in scope of runtime services of ALPI. ALPI shall provide information of the Security level of network communication via configuration.* *Satisfies:* *R009, R010*	F

ID	Source	Allocation	Requirement	Scope
REQ-ALPI-015			(removed)	F
REQ-ALPI-016	7.1	OP	The ALPI shall provide an offline **Configuration** data structure aimed to characterize the Functional Application *Rationale:* *ALPI shall provide a configuration data structure for the purpose of characterizing ALPI services with respect to functionalities related to communication, safety, security, orchestration, logging of a Functional Application.* *Satisfies:* *R028, R036*	F
REQ-ALPI-017	7.3.2.4	RT, CF, OP	The ALPI shall provide **Models** for Functional Application life-cycle. *Rationale:* *ALPI shall provide models to standardize life-cycle management; list to consider:* • *SW Architecture model: Functional Application is one or more processes* • *Life-Cycle safety assessment model: compliant with CENELEC* • *Process model: Task (UNIX process, ref. glossary)* • *Programming Model: Task, deterministic scheduling, RTC for Safety Related Functional Application* • *Executable generation model: validated compiler, linker, loader* • *Timing Model: timer-clock, execution deadline of FAT* • *Execution model: Start/Init, Operate, Stop/shutdown* • *Communication Model: MOM, standard P2P, publish/subscribe, transparent application of Gateway concept for external communication; use of OPC/UA, SNMP standard protocols* • *Configuration Model: Application Engineering Configuration data; Application-specific RTE data (for FA integration and runtime execution)* • *Security Model: compliant with IEC622443, EN50701, IEC 63452* • *Maintenance Model: provided by RTE; interoperable with external orchestration services IF-ORCH* • *Logging Model: provided by RTE SYSLOG services; interoperable with external services IF-DIAG* • *Error handling: according to CENELEC EN50129:2018* • *Diagnostics model: Collection of Functional Application analytics for KPIs* *Satisfies:* *R009, R010, R011, R019, R020, R021, R022, R023, R024, R028, R033, R047*	F

ID	Source	Allocation	Requirement	Scope
REQ-ALPI-018		CF	The ALPI shall provide an offline **Deployment** data structure aimed to deploy or update Functional Application *Rationale:* *ALPI shall provide a data structure for the purpose of deploying a Functional Application. The data structure is transparently used for **orchestration** purposes. The deployment data structure shall be used to check FA configuration and FA executable integrity.* *Satisfies:* *R035, R036*	F
REQ-ALPI-019	7.3.1.3	RT, CF	The ALPI shall provide resources/mechanism/services for Application Logging, Monitoring, Diagnostics. *Rationale:* *ALPI shall provide a data structure and services for the purpose of exporting Functional Application data to external entities. The data is selected through configuration, and it is transparently used for logging purposes. It shall be also possible direct logging, via SYSLOG (RTE) services* *Satisfies:* *R034, R056, R057*	F
REQ-ALPI-020	7.3.2.1 7.3.2.3	RT, CF	The ALPI shall provide the **SRAC** to be **fulfilled** by a safety related Functional Application. *Rationale:* *ALPI shall provide a clear definition of SRAC to be fulfilled by the Functional Application. These SRAC are imposed by lower layer if necessary.* *Satisfies:* *R008*	F
REQ-ALPI-021	7.3.2.3	RT, CF	*The ALPI shall provide the **SRAC** to be **exported** to installation, maintenance phases.* *Rationale:* *ALPI shall provide a clear definition of SRAC to be exported to installation, maintenance phases.* *Satisfies:* *R008*	F

Detailed requirements for previous general topic

ID	Source	Allocation	Requirement	Scope
REQ-ALPI-022	7.3.1.1	RT, CF	The ALPI shall provide a SW architecture model where a Functional Application SW is implemented through one or more process. A process is referred to as a "UNIX process" as specified in UNIX RTE environment. *Rationale: ALPI shall provide standard sw architecture model for developing Functional Application.*	F
REQ-ALPI-023			*(removed)*	X
REQ-ALPI-024	7.3.1.1	RT, CF	The ALPI shall provide services and functionalities compliant with the CENELEC life cycle. The related documentation shall be usable for modular certification. *Rationale: ALPI shall provide standard life-cycle model for assessing a Functional Application.* *Satisfies: R11*	F
REQ-ALPI-025	7.3.1.1	RT, CF	The ALPI shall provide a standard POSIX interface. This interface is directly mappable on every POSIX compliant RTE. *Rationale: ALPI shall provide a standard interface*	F
REQ-ALPI-026	7.3.1.1	RT, CF	The ALPI shall provide a UNIX process model to develop Functional Application. ALPI interface should be POSIX. *Rationale: ALPI shall provide a process model to develop Functional Application Process*	F
REQ-ALPI-027	7.3.1.1	RT, CF	The ALPI shall provide a Programming Model in which process are realized with task; tasks are executed using deterministic behaviour. For deterministic, safety related task the RTC (Run To Completion) schema should be used. *Rationale: ALPI shall provide a Programming Model to develop deterministic Functional Application Process*	F

ID	Source	Allocation	Requirement	Scope
REQ-ALPI-028	7.4.6.3	RT, CF	The ALPI shall provide an Execution Model of Functional Application Tasks. Task Execution shall be • timer-based, i.e., in configured regular intervals, or in the form of one-shot timers. • event-based, i.e., upon receipt of (certain types of messages). • timer- and event-based, i.e., the Task obtains execution time in regular intervals, or in the form of one-shot timers, only if (certain types of) messages have (or have not) been received. The specific execution modes are defined in the ALPI configuration *Rationale: ALPI shall provide an Execution Model of a Functional Application Process.* *Satisfies:*	F
REQ-ALPI-029	7.3.1.2	RT, CF	The ALPI shall provide qualified tools (compiler, linker, loader, …) for executable generations. *Rationale: ALPI shall provide qualified tools for executable generations* *Satisfies:*	F
REQ-ALPI-030	7.3.1.3	RT, CF	The ALPI shall provide services for timer-clock, for defining and controlling execution deadline of task. *Rationale: ALPI shall provide timing model to be used by Functional Application tasks* *Satisfies: R19, R020, R022, R025, R031*	F
REQ-ALPI-031	7.3.1.4	RT, CF	The ALPI shall provide services to Start/Init, Operate, Stop/shutdown a Functional Application task. *Rationale: ALPI shall provide an Execution model to be used by Functional Application* *Satisfies: R016*	F
REQ-ALPI-032	7.3.1.4	RT, CF	The ALPI shall provide standard communication services. These services shall be based on MOM, P2P and publish/subscribe paradigms. ALPI services shall allow the use of OPC/UA, SNMP or other standard protocols (e.g., as defined in Subset 147). It should be possible to apply transparently the Gateway concept for external communication. *Rationale: ALPI shall provide a Communication Model to be used by Functional Application* *Satisfies: R028, R30*	F

ID	Source	Allocation	Requirement	Scope
REQ-ALPI-033	7.3.2.2	RT, CF	The ALPI shall provide two types of configuration data: Application Engineering Configuration data and Application-Specific RTE data. AEC data for configuring application (i.e IXL DB, data preparation, etc, communications IDs). ASRTE data for Functional Application integration and runtime execution. (i.e. cycle time, communication nodes, SIL of tasks, …). *Rationale:* *ALPI shall provide a Configuration Model to define the behaviour of a Functional Application.* *Satisfies:* *R030*	F
REQ-ALPI-034	7.4.9	RT, CF	The ALPI shall provide security services compliant with IEC62443, EN50701, IEC 63452. Security on communication is transparent to ALPI and it is transparently managed end-to-end by lower layers. Specific security services such as cryptographic algorithm are provided by ALPI run time services. Specific requirement related to the use of PKI (Public key infrastructure) are defined via ALPI configuration and properly implemented by lower layers. *Rationale:* *ALPI shall provide a Security Model to be used by Functional Application* *Satisfies:* *R009*	F
REQ-ALPI-035	7.4.11	RT, CF	The ALPI shall provide maintenance services for Functional Application. These are provided by RTE and shall be interoperable with external orchestration services (IF-ORCH). *Rationale:* *ALPI shall provide a Maintenance Model for a Functional Application* *Satisfies:* *MPC-P05*	F
REQ-ALPI-036	7.3.1.3	RT, CF	The ALPI shall provide Logging services for Functional Application. Run-time Logging will be provided by RTE through Syslog services. Implicit logging of specific application data is achieved through configuration, specifying data and frequency of logging. The logged data shall be interoperable with external services IF-DIAG. *Rationale:* *ALPI shall provide a Logging Model for a Functional Application* *Satisfies:* *R032, R033, R057*	F

ID	Source	Allocation	Requirement	Scope
REQ-ALPI-037	7.4.6.6	RT, CF	The ALPI shall provide error handling services according to CENELEC EN50129:2018. *Rationale: ALPI shall provide a standard Error Model for a Functional Application* *Satisfies: R024*	F
REQ-ALPI-038	7.3.1.3	RT, CF	The ALPI shall provide Diagnostics services. It will be possible the Collection of Functional Application analytics for KPIs. *Rationale: ALPI shall provide a standard Diagnostics Model for a Functional Application* *Satisfies: R033*	F

APPENDIX D MANAGEMENT, DIAGNOSTICS AND SECURITY RELATED INTERFACE REQUIREMENTS

This Appendix lists requirements for the interfaces as discussed in chapter 8, Management, Diagnostics and Security related Interfaces.

Column	Meaning
Source	S: SP CE Domain OAS [15] (released to Mirror Group on 2024-05-07) +: new #: (heavily) modified or rewritten
Allocation	VE: Virtualization environment SS Diag: Shared Services for Diagnosis MPM: Modular Platform Management
Scope	F: full (all target environments) OB: On-board required, trackside optional TS: Trackside required, on-board optional O: optional for all environments X: not in scope for work package 26

D.1 COMMON REQUIREMENTS

ID	Source	Allocation (needed?)	Requirement	Scope
GEN-1	EuroSpec, TCMS_DS	RE	Data transfer shall have no influence on the operation of the overall system. *Rationale: Separation and (de-) prioritization of the data transferred on networks, etc. needs to be guaranteed.*	F
GEN-2	EuroSpec, TCMS_DS	RE	All interfaces shall support means for authentication and encryption.	F
GEN-3	S SPT2CE-1421, Step 4	RE	All interfaces shall provide means for the connected entities to check whether the interface is up and running.	F

EuroSpec [22] provides additional specifications on Software Updates [23] and Maintenance Software [24], which have not yet been considered in this deliverable but might be investigated in further work on the topic.

D.2 REQUIREMENTS ON CEME-DIAG

For the source references, please refer to [15].

ID	Source	Allocation	Requirement	Scope
CEME DIAG-1	+	VE	It shall be possible to ***configure the Virtual Machine Management*** *w.r.t. which diagnostics information types are provided to the Platform Mgmt.*	TS
CEME -DIAG-2	+	VE	Diagnostics information provided by the Virtual Machine Management shall contain ***time stamps.***	TS
CEME -DIAG-3	+	VE	Diagnostics information exchanged shall be based on ***standardized naming convention for entities*** *(CPUs, etc.).*	TS
CEME -DIAG-4	S# SPT2CE-1489 - SW Failure of one complete VE Instance	VE	The Virtual Machine Management shall issue a diagnostics information when a **failure of a complete Virtualization Environment instance has occurred.**	TS
CEME -DIAG-5	S# SPT2CE-1489 - SW Failure of one complete VE Instance	VE	The Virtual Machine Management shall issue a diagnostics information when a **failure of a complete VE instance has been overcome.**	TS
CEME -DIAG-6	S# SPT2CE-1487 - SW Failure of all VE Instances	VE	The Virtual Machine Management shall issue a diagnostics information when a **failure of all VE instances has occurred.**	TS
CEME -DIAG-7	S# SPT2CE-1487 - SW Failure of all VE Instances	VE	The Virtual Machine Management shall issue a diagnostics information when a **failure of all VE instances has been overcome.**	TS

ID	Source	Allocation	Requirement	Scope
CEME -DIAG-8	S# SPT2CE-1496 - Individual HW failure within one physical Computing Element	VE	The Virtual Machine Management shall issue a diagnostics information when an **individual HW failure within one physical Computing Element has occurred.**	TS
CEME -DIAG-9	S# SPT2CE-1496 - Individual HW failure within one physical Computing Element	VE	The Virtual Machine Management shall issue a diagnostics information when an **individual HW failure within one physical Computing Element has been overcome.**	TS
CEME -DIAG-10	S# SPT2CE-1490 - Total HW failure of one complete physical computing element	VE	The Virtual Machine Management shall issue a diagnostics information when a **total HW failure of one complete physical computing element has occurred.**	TS
CEME -DIAG-11	S# SPT2CE-1490 - Total HW failure of one complete physical computing element	VE	The Virtual Machine Management shall issue a diagnostics information when a **total HW failure of one complete physical computing element has been overcome.**	TS
CEME -DIAG-12	S# SPT2CE-1492 - Disaster scenario - failure of all computing elements	VE	The Virtual Machine Management shall issue a diagnostics information when a **failure of all computing elements has occurred.**	TS

ID	Source	Allocation	Requirement	Scope
CEME -DIAG-13	S# SPT2CE-1492 - Disaster scenario - failure of all computing elements	VE	The Virtual Machine Management shall issue a diagnostics information when a **failure of all computing elements has been overcome.**	TS
CEME -DIAG-14	S# SPT2CE-1501 - Failure of one external communication channel regarding I0	VE	The Virtual Machine Management shall issue a diagnostics information when a **failure of one external communication channel regarding I0 has occurred.**	TS
CEME -DIAG-15	S# SPT2CE-1501 - Failure of one external communication channel regarding I0	VE	The Virtual Machine Management shall issue a diagnostics information when a **failure of one external communication channel regarding I0 has been overcome.**	TS
CEME -DIAG-16	S# SPT2CE-1499 - Failure of all external communication channels regarding I0	VE	The Virtual Machine Management shall issue a diagnostics information when a **failure of all external communication channels regarding I0 has occurred.**	TS
CEME -DIAG-17	S# SPT2CE-1499 - Failure of all external communication channels regarding I0	VE	The Virtual Machine Management shall issue a diagnostics information when a **failure of all external communication channels regarding I0 has been overcome.**	TS

D.3 REQUIREMENTS ON ORCH

For the source references, please refer to [15].

ID	Source	Allocation (needed?)	Requirement	Scope
ORCH-1	SPT2CE-1421	VE	ORCH shall offer a function through which the Platform Management can **verify that the deployment of the Virtualization Environment has been performed correctly by the Virtual Machine Management.** *Note: This requirement goes beyond those strictly derived from SPT2CE-1421.*	TS
ORCH-2	SPT2CE-1428	VE	ORCH shall offer a function through which the Platform Management can **verify whether a Virtualisation Environment of the designated Physical Computing Elements complies with the requirements** as per the certified FS Deployment Rules of a Functional System to be deployed.	TS
ORCH-3	SPT2CE-1428	VE	ORCH shall offer a function through which the Platform Management can **confirm whether sufficient resources can be allocated** on the designated Physical Computing Element(s) in accordance with the FS.	TS
ORCH-4	SPT2CE-1428	VE	ORCH shall offer a function through which the Platform Management can **create Virtual Computing Element(s)** according to the FS Deployment Rules of a Functional System to be deployed.	TS
ORCH-5	SPT2CE-1428	VE	ORCH shall offer a function through which the Platform Management can **verify that the correct Virtual Computing Element(s) have been created** and that they are ready for FS Compartment deployment.	TS
ORCH-6	SPT2CE-1431 SPT2CE-1448	VE	ORCH shall offer a function through which the Platform Management can **verify the correct mapping of FS Compartment and Virtual Computing Element.**	TS
ORCH-7	SPT2CE-1439 SPT2CE-1602	VE	ORCH shall offer a function through which the Platform Management can request to **release Virtual Computing Elements.**	TS

ID	Source	Allocation (needed?)	Requirement	Scope
ORCH-8	SPT2CE-1448 SPT2CE-1446 SPT2CE-1602 SPT2CE-1458	VE	ORCH shall offer a function through which the Platform Management can trigger a **backup of the state of a FS compartment.**	TS
ORCH-9	SPT2CE-1456 SPT2CE-1458	VE	ORCH shall offer a function through which the Platform Management can request to **uninstall a Functional System Compartment.**	TS
ORCH-10		VE	ORCH shall offer a function through which the Platform Management can **setup the functions needed within a Virtual Computing Element** for the subsequent usage of the I1-UPDATE interface to manage Functional System installations, updates, etc.	TS

D.4 REQUIREMENTS ON MGMT-DIAG

For the source references, please refer to [15].

ID	Source	Allocation (needed?)	Requirement	Scope
MGMT-DIAG-1		MPM	It shall be possible to **configure the FS Diagnostics Server** w.r.t. which diagnostics information types are provided to the Platform Management.	F
MGMT-DIAG-2		MPM	Diagnostics information provided by the FS Diagnostics Server shall contain **time stamps.**	F
MGMT-DIAG-3		MPM	Diagnostics information exchanged shall be based on **standardized naming convention for entities** (CPUs, etc.).	F
MGMT-DIAG-4	SPT2CE-1483 - Total SW Failure of one FS Compartment	MPM	The FS Diagnostics Server shall issue a diagnostics information when an FS Compartment has failed.	F
MGMT-DIAG-5	SPT2CE-1501 - Failure of one external communication channel regarding I0	MPM	The FS Diagnostics Server shall issue a diagnostics information when a **failure of one external communication channel regarding I0 has occurred.**	F
MGMT-DIAG-6	SPT2CE-1501 - Failure of one external communication channel regarding I0	MPM	The FS Diagnostics Server shall issue a diagnostics information when a **failure of one external communication channel regarding I0 has been overcome.**	F

ID	Source	Allocation (needed?)	Requirement	Scope
MGMT-DIAG-7	SPT2CE-1499 - Failure of all external communication channels regarding I0	MPM	The FS Diagnostics Server shall issue a diagnostics information when a **failure of all external communication channels regarding I0 has occurred.**	F
MGMT-DIAG-8	SPT2CE-1499 - Failure of all external communication channels regarding I0	MPM	The FS Diagnostics Server shall issue a diagnostics information when a **failure of all external communication channels regarding I0 has been overcome.**	F

D.5 REQUIREMENTS ON FS-UPDATE

For the source references, please refer to [15].

ID	Source	Allocation	Requirement	Scope
FS-UPDATE-1	SPT2CE-1431 SPT2CE-1448 SPT2CE-1446 SPT2CE-1602 SPT2CE-1456 SPT2CE-1458	MPM	FS-UPDATE shall offer a function through which the Platform Management can request to **install software / configurations within FS Compartments** onto a corresponding Virtual Computing Element as per the FS Deployment Rules of the Functional System to be deployed.	F
FS-UPDATE-2	SPT2CE-1431 SPT2CE-1456 SPT2CE-1458	MPM	FS-UPDATE shall offer a function through which the Platform Management can request to **start software / configuration within a Functional System Compartment.**	F
FS-UPDATE-3	SPT2CE-1439 SPT2CE-1446 SPT2CE-1456 SPT2CE-1602 SPT2CE-1458	MPM	FS-UPDATE shall offer a function through which the Platform Management or potentially Shared Services can request to **stop software / configuration within a Functional System Compartment.**	F
FS-UPDATE-4	SPT2CE-1448 SPT2CE-1446 SPT2CE-1602 SPT2CE-1458	MPM	FS-UPDATE shall offer a function through which the Platform Management or potentially Shared Services can **test if software / configuration within a Functional System Compartment is up and running.**	F
FS-UPDATE-5	SPT2CE-1456	MPM	FS-UPDATE shall offer a function through which the Platform Management or potentially Shared Services can **check the version of a software / configuration within a Functional System Compartment.**	F

APPENDIX E COLLECTED OPEN POINTS FOR THE MPC

The following lists represents the collected opens for the Modular Platform Concept. They are meant for future work, e.g., in the ERJU SP CE domain, other domain, or future ERJU IP projects.

ID	Source Chapter	Open
Open-001	6.3	What kind of HW architecture aspects will be “bottle necks” in parallel usage by independent FS compartments running aggregated on same physical computing element? • Memory bandwidth? • Network bandwidth? How can this aspects be handled / defined FS compartment wise?
Open-002	6.5	Architecture: how to handle the message-based interface of the NHA (see chapter 6.7) to FS Compartments above – is this interface a part of I3?
Open-003	6.6	The details of the requirements towards the hardware (as e.g. hardware architecture, cores, performance, communication, MTBF, virtualization extension, …) must be defined.
Open-004	6.7.1.1	What is the criteria for unique CPU identification? MAC address? TPM content?
Open-005	6.7.1.2	How to solve the relationship of used CPU cores (used by the FS compartment within the VCE) and the information which shall be provided by the native running software as NHA?
Open-006	6.7.1.4	The details regarding sensor information provided by NHA in context of temperature must be clarified.
Open-007	6.7.1.5	The details regarding sensor information provided by NHA in context of voltage must be clarified.
Open-008	6.7.1.6	It must be clarified, if the required information from the physical hardware can be provided via standardized **interface I2** or if the NHA functionality must be adapted for different HW variants.
Open-009	6.7.1.6	The responsibility and technical handling (installation/update) of such a NHA software must be clarified.

ID	Source Chapter	Open
Open-010	6.7.2	The safe handling of safety critical software in context of a non-safe VE with standard orchestration tools must be clarified, as e.g. to avoid unallowed installation and starting of FS duplicates
Open-011	5.4	For the MPC Architecture, a combined modularization architecture proposal showing how the deeper levels of FS (e.g., compartments, RTE, Functional Applications, etc.) interact with the interfaces introduced in the service architecture, as well as with the Platform Management and/or Shared Services.
Open-012	6.8	Overall certification of the secure device needs to be clarified.
Open-013	6.8	The architecture for access to the TPM of the physical hardware must be clarified in context of - Access by several FS compartments provided by different suppliers - functionality secure boot - certification for IE 62443 SL3. *Rationale: Access to physical hardware is not guaranteed for the IT-security mechanism running within a VCE.*
Open-014	6.9.4	The overall architecture for the update of FS must be clarified. Which dependencies in context of “FS consists of several FS compartments” are handled on side of the Shared Services and on side of the Platform Management?
Open-015	6.9.5	The safety related overall architecture for georedundant FS with safe handling of split-brain problem is not yet defined.
Open-016	6.10	Scalable handling of CPU resources: How to handle the scalable usage of CPU resources (cores, memory, network cards,) for flexible usage of independent FS compartments running on same PCE.
Open-017	6.11	The architectural details regarding “needed diagnosis data to do a root cause analysis and initiate automated repair activities” has to be clarified. Which data is relevant for Platform Management? Is a standardization of this senseful and possible or not?

ID	Source Chapter	Open
Open-018	6.11.5	Architecture for network diagnosis: • Which architecture element is responsible for network diagnosis? • Which architecture element is responsible to identify the root cause in case of network communication failures as e.g. regarding the FS internal communication between FS compartments? • Does this architecture element provide I1 Diagnosis to the Shared Services?
Open-019	6.11.4	Architecture for network diagnosis: which architecture element is responsible to identify the root cause in case of network communication failures as e.g. regarding the FS internal communication between FS compartments? Does this architecture element provide I1 Diagnosis to the Shared Services?
Open-020	6.12.1	Overall architecture in context of installation and update needs to be defined. • How to bring the individual FS Compartments onto the new VE instance on a new HW? • How to update FS Compartment versions? • What are the dependencies between Shared Services for Update and Platform Management? • How to differentiate between update of non-safe parts and safety related parts? • How to handle the NHA software in context of installation and update?
Open-021	6.13	An automated installation of safety critical FS compartments by a basic integrity Platform Management must be evaluated from view of safety. The FS system keeps running as 2oo2 and ensures a safe synchronization of the newly started FS compartment. But duplication of safety related FS compartments has the potential to lead to the split-brain problem in context of a duplication of more than one FS compartment.
Open-022	6.13.1	What exactly is necessary in context hardening of the VE? What kind of VE functionalities must be deactivated or even removed to ensure that the handling of rail systems running on VE is possible in way as needed (efficient handling and available running FS)?

ID	Source Chapter	Open
Open-023	6.13.1	Is a kind of “generic” testing possible for performance and runtime behaviour of a new VE version to avoid the need for integration of each individual FS compartment version with new VE Version?

Table 11: Collected MPC Opens

APPENDIX F MPC GLOSSARY

Term	Abbreviation	Context Chapter	Definition
Application-Level Platform Independence	ALPI	7	Application-Level Platform Independence is achieved through the combination of the Runtime Layer and the Safety Layer providing all necessary safety-related and non-safety-related interfaces and resources for fulfilling an application's functions. This includes diagnosis, logging, and monitoring. In addition, also the SRACs imposed on the application by the underlying platform must be fulfilled, ideally standardized.
Compartment Execution & Management Environment	CEME	5	The CEME is following the definition of CEE (see chapter 3.7.2) and adds the management for PCE, VE and VCE.
Compatible Platform Implementation	CPI	3.11	An implementation of the Modular Platform Concept (MPC) as presented in this deliverable that is able to run Functional Systems.
Hardware-Level Platform Independence	HLPI	6	Hardware-Level Platform Independence is achieved through the combination of the Hardware Layer and the Virtualisation Layer providing all necessary interfaces to aggregate multiple Functional Systems with potentially different safety integrity levels on the same physical hardware.
Modular Platform Concept	MPC	3	A full concept showing how develop, deploy and operate railway applications in a modular way on the trackside, data centres or on-board a train.
Native Hardware Access	NHA	6	The NHA enables access to hardware parameters and data from FS Compartments.
Platform Management	PM	5	Platform Management manages CEME and FS Compartments while providing interfaces to the outside.
Shared Services	n/a	3	The Shared Services represent a collection of overarching services (e.g., update and configuration) defined by the System Pillar TCCS domain.
Virtual Machine Management	VMM	8	Virtual Machine Management refers to the software and processes used to create, monitor, and manage virtual machines.

Table 12: MPC Glossary