Rail to Digital automated up to autonomous train operation

D26.2 – Intermediate Modular Platform requirements, architecture and specification

Due date of deliverable: 31/10/2023

Actual submission date: 03/11/2023

Leader/Responsible of this Deliverable: Maik Fox / DB Netz AG

Reviewed: Y

Document status		
Revision	Date	Description
01	18/09/2023	First issue for internal Review
02	29/09/2023	Second issue for TMT Review
03	24/10/2023	Third issue for Steering Committee Review
04	15/04/2024	Final revision incorporating external feedback

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 Netz AG	Executive Summary, Introduction, Conclusion, Chapters 2/3/4/6, Review
Oliver Mayer-Buschmann	DB Netz AG	Chapter 2/3/4/6
Julian Wissmann	DB Netz AG	Chapter 4, Review
Martin Kochinke	DB Systemtechnik GmbH	Review
Piero Petruccioli	Trenitalia	Chapter 3/6
Thomas Martin	SBB	Chapter 2/3/4/6
Valentin Inocencio Cuesta	SBB	Review
Sonja Steffens	Siemens Mobility	Chapter 5/6, Review
Wolfgang Wernhart	GTS Austria	Chapter 3/4
Stefan Resch	GTS Austria	Chapter 3/4/6
Patrick Rozijn	NS Reizigers	Chapter 3

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 needed 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 these not only to the demonstrator work package 36, building the "On-Board Platform Demonstrator", but also to the ERJU System Pillar and future Innovation Pillar activities. Based on the consolidated learnings of the work package's first task, this second deliverable provides an insight into the intermediate status of the actual specification of a "Modular Platform".

Three different domains were introduced 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 external interfaces – providing interoperability. Each of the domains is discussed after presenting an updated overview over the modular platform concept itself.

The intermediate nature of the work as represented by this deliverable implies that further work is necessary to create a coherent specification suite for modular platforms, as is planned for the subsequent deliverable D26.3. Afterwards, the work package will be studying approaches for certification and acceptance of modular platforms, likely based on the work of the System Pillar. In the end, WP26 plans to provide a holistic view on modular computing platforms, where they are coming from and how they can enable ERJU's vision of the future of the railway system.

ABBREVIATIONS AND ACRONYMS

ADAS	Automatic Driver Assistance System
ALPI	Application-level Platform Independence
ATO	Automatic Train Operation
BMS	Bogie Monitoring System
CONNECTA	CONtributing to Shift2Rail's NExt generation of high Capable and safe TCMS PhAse 3
COTS	Commercial Off The Shelf
CCS	Command, Control and Signalling
ETCS	European Train Control System
FDF	Functional Distribution Framework
FOC	Functional Open Coupling
FVA	Functional Vehicle Adapter
GoA	Grade of Automation
HLPI	Hardware-level Platform Independence
HVAC	Heating, Ventilation and Air-Conditioning
IM	Infrastructure Manager
MDCM-OB	Monitoring Diagnostics Configuration Maintenance On-Board
NG-TCMS	Next Generation TCMS
OCORA	Open CSS On-Board Reference Architecture
PI API	Platform-Independent Application Programming Interface
POSIX	Portable Operating System Interface
R2DATO	Rail to digital automated up to autonomous train operation
RCA	Reference CCS Architecture
RTE	Run Time Environment
RTOS	Real Time Operating System
RU	Railway Undertaking
SCP	Safe Computing Platform
SRACs	Safety Related Application Conditions
SW	Software
TCMS	Train Control Management System
TD	Technical Demonstrator

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

The present document constitutes the Deliverable D26.2 “Intermediate Modular Platform requirements, architecture and specification” in the framework of the WP 26, Task 26.2, of the FP2 R2DATO.

Computing platforms, either on-board a train, stationary along the trackside or in a data centre, are an integral part to the modernization and digitization of railway systems. While these computing platforms come in many variants and are built to vastly differing requirements targeting many applications and use cases, there are several common functionalities, nevertheless. Especially modern and ubiquitous computing platforms implement a wide variety of applications that depend on standardized interfaces, guaranteeing their interchangeability. This interchangeability is also critical to enabling competition between railway undertakings, allowing vendor-neutral access to the network without limiting future technical innovations.

Modular platforms aim to provide the standardized interfaces needed for safety related railway applications, independent on where they are located. Modularization is intended to help in (re-)deploying applications to computing platforms and potentially allow new approaches for (re-)certification, especially when considering software and hardware updates.

Targeting different levels of automated railway systems and grades of automation, new challenges for computing platforms can be found in – but are not limited to – topics such as availability, reliability, and cyber security. Here, the modularization approach intends to balance overall architecture subsystem complexity and development effort by providing a suitable and tailored set of specifications.

An intermediate step towards a full description of such a modular system, the so-called “Modular Platform”, is presented in this document. It is based on the work package’s previous deliverable D26.1 [16], consolidating the state-of-the-art, and structured as follows.

Chapter 2 presents the current state of proposal and alignment with regards to a common glossary.

Chapter 3 gives an high-level overview over the concept of the “Modular Platform” and summarizes the context, introduces architectural concepts, requirements, certification goals and cybersecurity needs. A glossary is also available, trying to align terms over different sources.

Chapter 4 discusses “Modular Platforms” from the perspectives of the application and the runtime environment, focusing on ways to achieve platform independence on the application level. Based on previous work and new discussions, this chapter tries to formulate a first approach for reaching platform independence on the application level. This includes a discussion of the potential structure of runtime environments, interfaces, programming models, and many other artefacts and assumptions needed for a holistic description of the application and its full lifecycle environment. Furthermore, the chapter provides a first, work-in-progress description of the potential interfaces between application and runtime environment, including messaging concepts.

Chapter 5 switches to the perspectives of the hardware and the runtime environment, investigating ways to achieve hardware independence and how to aggregate multiple runtimes. Here, as an intermediate step, requirements are discussed for each of the two, as well as potential solutions and an outlook towards certification options.

Chapter 6 gives a first insight into relevant interfaces external to the “Modular Platform”, needed for interoperability and also having an impact on the definition of the “Modular Platform” itself.

A summary and outlook conclude deliverable D26.2.

As a reminder: This deliverable is an intermediate insight into the work of this work package. As such, it is intentionally not complete and not yet fully coherent. Furthermore, input from multiple sources (e.g., ERJU System Pillar domains) are likely to have an effect on the glossary, interface needs and maybe other parts of this document. This document is not to be treated as a final specification of "Modular Platforms".

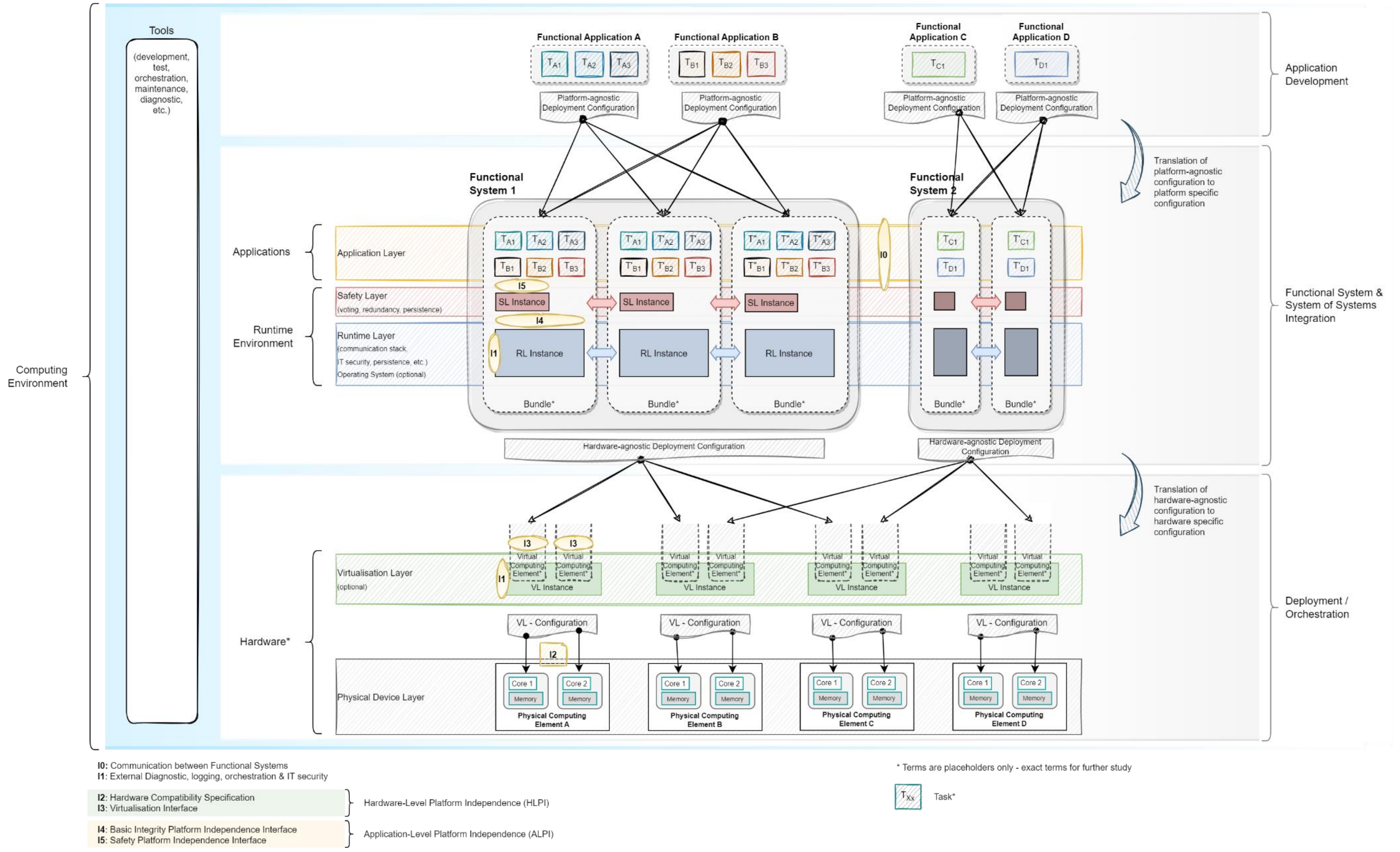
2 MODULAR PLATFORMS GLOSSARY

The Modular Platforms Glossary aims to align and define the term used in this document and also in the context of modular computing platforms within ERJU. However, the alignment and term definition is an ongoing process, not yet finished and currently based on our previous deliverable D26.1 [16] and the SP CE domain input [14]. The glossary needs to be aligned with the ERJU SP CE domain, and if necessary for external interfaces (I1) also with other domains or Innovation Pillar activities. Where we were not yet aware of an existing definition, additional content brought in here can be seen as an input to the SP CE domain and others.

Empty locations in the following tables indicate future study and alignment needs.

Nevertheless, we see the alignment and creation of a coherent glossary as an essential activity for the success of the modular platforms.

Before a traditional glossary in the form a table is given, a work-in-progress overview diagram is shown on the next page. The goal of the diagram is to give an insight into the context and also what needs to be named and aligned. The diagram encompasses more details than are currently discussed in this deliverable and they are, as such, to be seen as areas of future study.



Tools
(development, test, orchestration, maintenance, diagnostic, etc.)
Computing Environment
Functional Application A
Functional Application B
Functional Application C
Functional Application D
Platform-agnostic Deployment Configuration
Application Development
Translation of platform-agnostic configuration to platform specific configuration
Functional System 1
Functional System 2
Applications
Application Layer
Runtime Environment
Safety Layer (voting, redundancy, persistence)
Runtime Layer (communication stack, IT security, persistence, etc.) Operating System (optional)
SL Instance
RL Instance
Bundle*
I0
I1
I4
I5
Functional System & System of Systems Integration
Hardware-agnostic Deployment Configuration
Translation of hardware-agnostic configuration to hardware specific configuration
Virtualisation Layer (optional)
Virtual Computing Element*
VL Instance
I3
VL - Configuration
I2
Hardware*
Physical Device Layer
Core 1
Core 2
Memory
Physical Computing Element A
Physical Computing Element B
Physical Computing Element C
Physical Computing Element D
Deployment / Orchestration
I0: Communication between Functional Systems
I1: External Diagnostic, logging, orchestration & IT security
I2: Hardware Compatibility Specification
I3: Virtualisation Interface
Hardware-Level Platform Independence (HLPI)
I4: Basic Integrity Platform Independence Interface
I5: Safety Platform Independence Interface
Application-Level Platform Independence (ALPI)
* Terms are placeholders only - exact terms for further study
Task*

Figure 1: Intermediate Terminology Landscape (proposal)

Notes for the following table:

- The * in the glossary table indicates not yet aligned terms.
- The # in the glossary table indicates not yet aligned definitions.
- Empty cells are work in progress.

Term	Definition
Functional Application	A comprehensive set of self-contained software functions, assumed to be provided as one product by a single vendor. Functions within one application may have different functional safety requirements.
Computing Platform	Refers to an environment on which functional applications are run, comprised of hardware and software (i.e., the runtime environment).
Runtime Environment	A software that acts as intermediary by providing a generalized abstraction of the underlying hardware and software and enabling communication and data management for distributed applications. Definition derived from RCA/OCORA (see [5]): An (instance of a) runtime environment, which comprises safety services (e.g., integrity checking, fault tolerance, synchronisation and communication services related to safety, hardware and software monitoring as needed in safety context) and system services (e.g., application lifecycle management, platform and software monitoring, tracing and logging, communication services that are not related to safety, security means incl. authentication, encryption, key storage, etc., provisioning and management of persistent storage) and the communication stack for information exchange between Functional Applications running on the same Platform and with external entities.
Hardware	The physical and electronic parts of a computer or other piece of equipment.
Functional System	
Function / Task*	
Tools	# Software provided either generically or by the platform provider for development, test, orchestration, maintenance, diagnostics, etc.
Platform-agnostic deployment configuration*	
Platform-specific deployment configuration*	
System of systems	
Application Layer	
Safety Layer	# Provides voting, redundancy, persistence.
Runtime Layer	# Provides communication stack, IT Security, persistence, etc. Might contain an operating system.
Bundle*	

Term	Definition
Hardware-agnostic deployment configuration*	
Hardware-specific deployment configuration*	
Virtualisation Layer	
VL-Configuration*	
Physical Device Layer*	
Physical Computing Element*	
Hardware*	
Mixed-Criticality Workloads*	
Portability (of a Functional Application)*	
Flexibility (of a Platform)*	
I0: Communication between Functional Systems*	
I1: External Diagnostics, Configuration & Control Interface	
I2: Hardware Abstraction Interface	
I3: Virtualisation Interface	
I4: Basic Integrity Platform Independence Interface	
I5: Safe Platform Independence Interface	
IF-ORCHESTRATION	
IF-DIAGNOSTICS	
IF-LOGGING	
IF-IT-SEC	
Hardware-Level Platform Independence (HLPI)*	

Term	Definition
Application-Level Platform Independence (ALPI)*	
User Documentation*	
Instantiation of the Modular Platform*	# A product or service representing a modular platform as described here. This can be a single piece of onboard hardware with the necessary software, documentation and other artefacts, but can also range to a trackside data centre scale deployment with multiple nodes.

Table 1: Modular Platforms Glossary

3 MODULAR PLATFORMS OVERVIEW

In the context of R2DATO work package 26 (WP26), the “Modular Platform” is a railway-focused computing platform concept for modularized mixed criticality workloads for onboard and trackside environments. Based on this concept, distinct instantiations of the Modular Platform (e.g., COTS server hardware based for trackside use or embedded systems for onboard use) are possible, allowing business logic portability and maintenance across generations and target systems.

Based on input and guidance from various ERJU System Pillar activities, most notable the Computing Environment Domain, and a wealth of previous work in railway computing environment architectures (see WP26’s first deliverable [16]), this work package will attempt to aggregate and extend the state of the art while trying to generalize trackside and onboard needs where feasible.

This intermediate deliverable, as discussed in the introduction above, will give a first insight into the definition of the Modular Platforms and highlight the areas of work and results to be expected for the final deliverable of WP26 Task 2, D26.3. At this time, further input from the relevant System Pillar domains is expected to be available to augment the work and alignment on Modular Platforms.

The current state of the ERJU System Pillar (SP) Computing Environment (CE) Domain inputs are summarized in the following subchapter. A potential high-level architecture for the Modular Platforms, derived from SP CE domain inputs and the previous activities as outlined in [16], is shown subsequently. A chapter about high-level requirements for the Modular Platform discusses available catalogues and the distinction between onboard and trackside systems. Afterwards, the certification goals are explained. A brief outlook on the relevance of cyber security for the Modular Platforms is followed by a glossary.

The following three chapters will discuss in detail the important properties of Modular Platforms, namely application-level platform independence (ALPI) in chapter 4, hardware-level platform independence in chapter 5 and external interfaces to the platform itself in chapter 6.

3.1 ERJU SYSTEM PILLAR COMPUTING ENVIRONMENT DOMAIN INPUT

The ERJU System Pillar (SP) Computing Environment (CE) Domain [13] input document “Recommendation on interfaces to be standardised” [14] discusses relevant user stories for computing environments, based on the so called “Common Business Objectives” provided by the System Pillar [15]. These user stories are asking for advanced functionality not available in today’s railway- or onboard infrastructure, e.g., remote update or hardware replacement with (re-)certification minimal effort. Furthermore, standardisation of a platform would generally lead to a changed competitiveness landscape with the expectations that more potential suppliers can deliver functionality. Also, standardisation potentially has positive effects on education and, therefore, could foster creativity within the whole sector, according to the opinion of the work package members.

In general, the SP CE domain’s approach is to stay as holistic as possible (especially towards trackside and onboard[1]). Where needed, the details will be worked out in different domains (e.g., transversal CCS). As such, especially for interfaces external to the platform (I1), it is not yet sure who will work on the specifications. Potential options are the respective System Pillar domains or in

[1] For example, so far, the SP CE domain did not touch on onboard 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 onboard systems in the future and likely to be supported by the same specialized COTS hardware.

Innovation Pillar work packages. In general, this work package as well favours to avoid differentiation where none is strictly necessary.

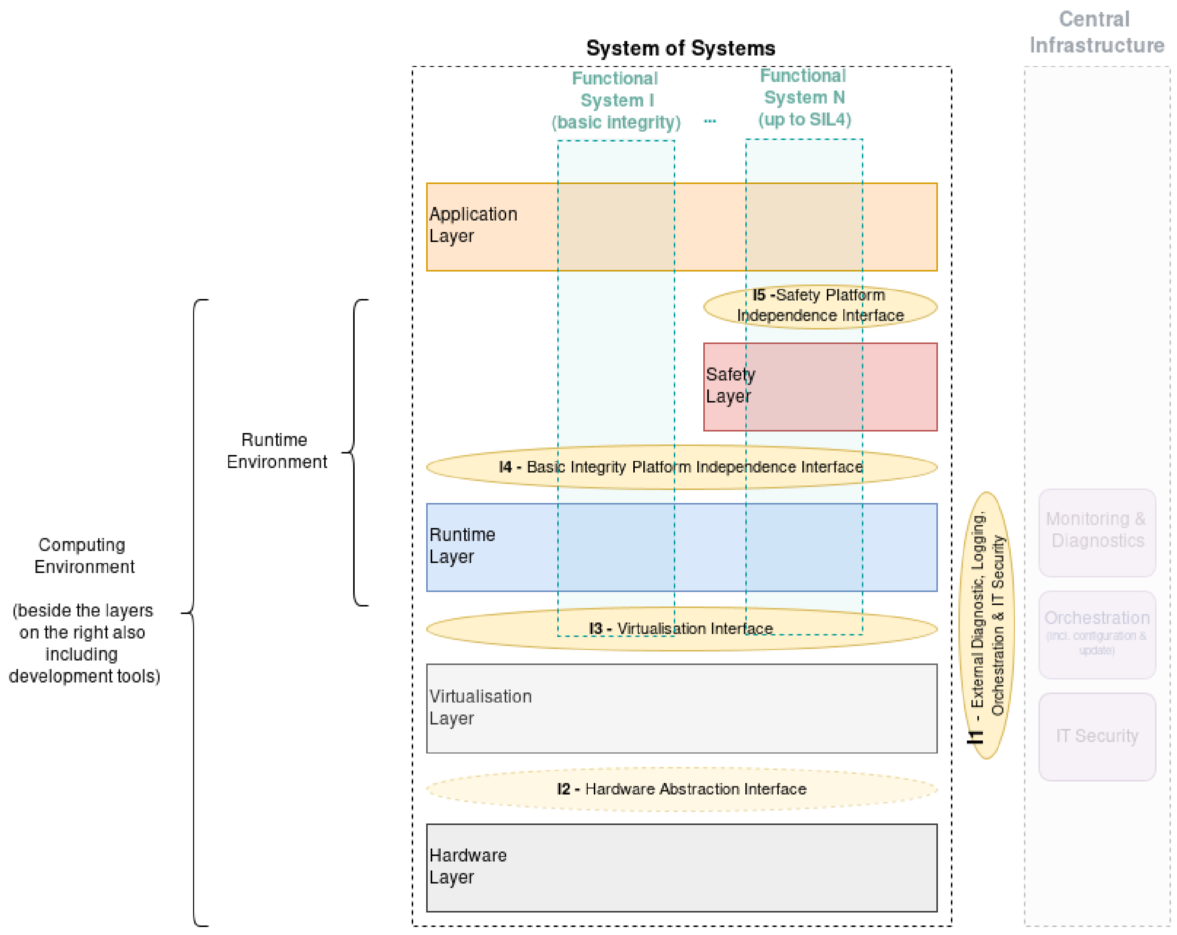

System of Systems
Central Infrastructure
Functional System I (basic integrity)
...
Functional System N (up to SIL4)
Application Layer
I5 -Safety Platform Independence Interface
Safety Layer
Runtime Environment
I4 - Basic Integrity Platform Independence Interface
Runtime Layer
Computing Environment
(beside the layers on the right also including development tools)
I3 - Virtualisation Interface
Virtualisation Layer
I2 - Hardware Abstraction Interface
Hardware Layer
I1 - External Diagnostic, Logging, Orchestration & IT Security
Monitoring & Diagnostics
Orchestration (incl. configuration & update)
IT Security

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

Based on SP CE domain layer model we see the following interfaces (see Figure 2 above):

- 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

For the time being, WP26 did not address HW standardisation (I2 and I3). For a solid definition of the base, where upper interfaces/layers could run on, it will be necessary to include those, at least by defining assumptions and requirements (see chapter 5 for a first discussion of this new topic).

Even if there are recommendations given in [14], WP26 will treat all interfaces of equal importance for the purpose of this intermediate release of our work.

For further details on the SP CE domain's work on the interfaces, please refer to [14].

3.2 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 2). To enable these benefits, following the previous work and SP CE domain inputs, three distinct domains for the Modular Platform 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. The relation between the domains is shown in the following diagram.

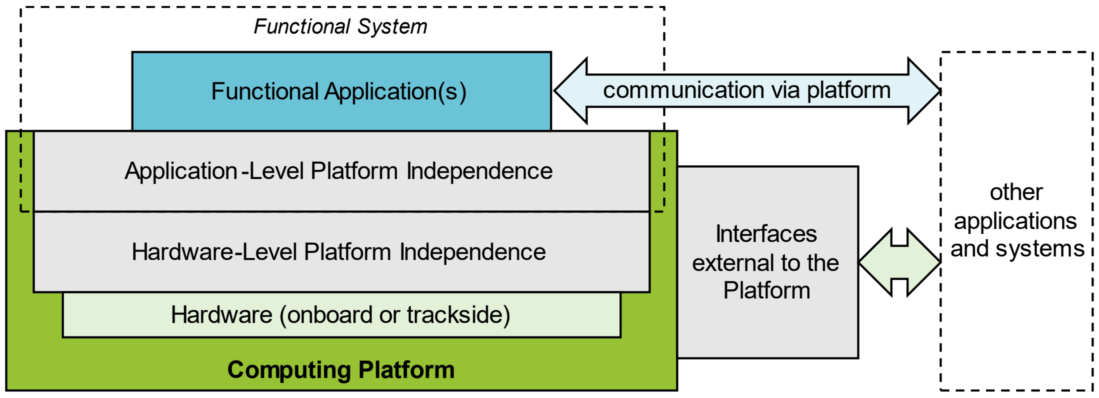

Functional System
Functional Application(s)
communication via platform
Application-Level Platform Independence
Hardware-Level Platform Independence
Hardware (onboard or trackside)
Computing Platform
Interfaces external to the Platform
other applications and systems

Figure 3: 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 dedicated subsequent chapters. For the purpose of this overview chapter, the next figure will show how the Modular Platform domains implement the SP CE domain interface recommendations, 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:

- external interfaces → I1: External Diagnostics, Configuration & Control Interface
- Hardware-Level Application Independence (HLPI)
  - I2: Hardware Abstraction Interface
  - I3: Virtualisation Interface
- Application-Level Platform Independence (ALPI)
  - I4: Basic Integrity Platform Independence Interface
  - I5: Safe Platform Independence Interface

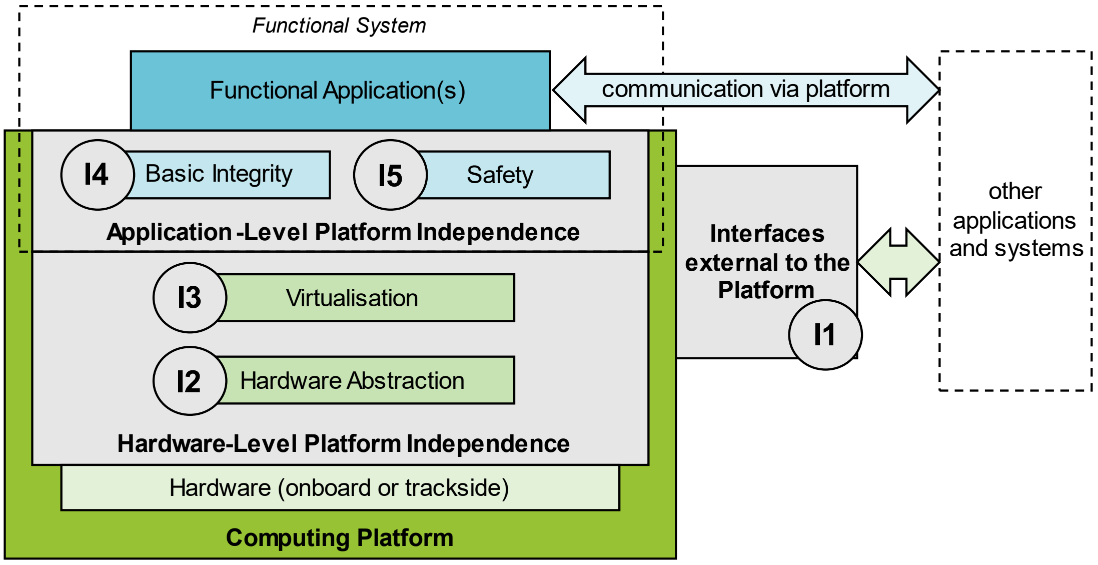

Functional System
Functional Application(s)
communication via platform
I4
Basic Integrity
I5
Safety
Application -Level Platform Independence
I3
Virtualisation
I2
Hardware Abstraction
Hardware-Level Platform Independence
Hardware (onboard or trackside)
Computing Platform
Interfaces external to the Platform
I1
other applications and systems

Figure 4: 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. Interfaces I2 and I3 are providing means for hardware abstraction and, if needed, aggregation respectively integration of multiple runtime environments running on the same hardware, e.g., by means of containerization, virtualization or by using a hypervisor or other approaches. 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.

Between the SP CE domain interfaces, common parts are needed for the implementation of an actual computing platform, as shown in the next figure.

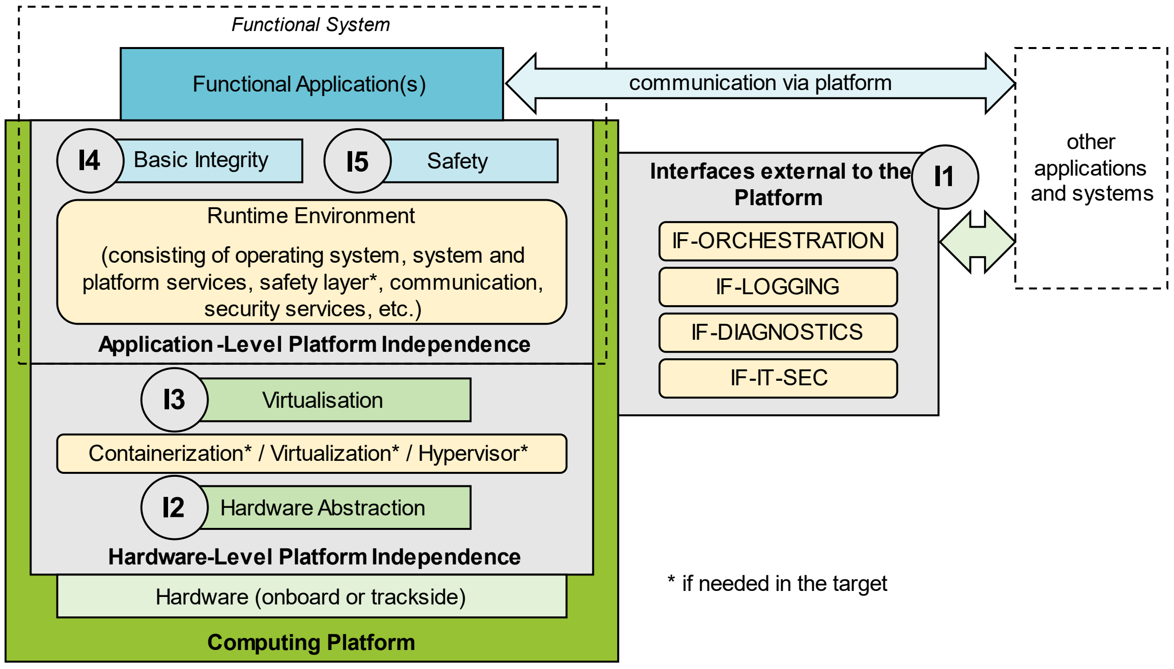

Functional System
Functional Application(s)
communication via platform
I4
Basic Integrity
I5
Safety
Runtime Environment
(consisting of operating system, system and platform services, safety layer*, communication, security services, etc.)
Application -Level Platform Independence
I3
Virtualisation
Containerization* / Virtualization* / Hypervisor*
I2
Hardware Abstraction
Hardware-Level Platform Independence
Hardware (onboard or trackside)
Computing Platform
Interfaces external to the Platform
I1
IF-ORCHESTRATION
IF-LOGGING
IF-DIAGNOSTICS
IF-IT-SEC
other applications and systems
* if needed in the target

Figure 5: Modular Platform architecture showing key ingredients and interfaces

These common parts, also called “layers” by the SP CE domain, shown in the figure 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.

More insights into the modular platform architecture concepts can be found in chapters 4.3 and 5.

3.3 Modular Platforms Requirements

Requirements towards (modular) computing platforms have been defined in the past. Sometimes, there is distinction made between sets of requirements for trackside and onboard systems. This work package will use available sources in future work and, where possible, provide a generic set of requirements, potentially containing some optional requirements for onboard or trackside. Additional sources (e.g., SP) of requirements might become available to support this activity.

For the purpose of this intermediate deliverable, a brief discussion is given in the two subchapters below.

3.3.1 Onboard Requirements Sources

With regards to the current and well-established RCA/OCORA initiative, their valuable targets, their strict interrelation and the results already achieved, it's recommended to consider the OCORA initiative as main source for the On-board Computing Platform requirements and in particular the document OCORA TWS03-020-”Computing Platform Requirements” v. 4.1 [6], part of the OCORA Release 4, notably all the “approved” requirements MSC-XX, with XX from 01 to 127 (including the optional ones).

3.3.2 Trackside Requirements Sources

Specifications relevant for current trackside solutions do not include modular platform requirements as such. However, it is expected that some EULYNX requirements indeed can be fulfilled or at least supported on the level of the modular platform.

Additionally, the “SIL4 Data Center” report [4] lists relevant requirements.

3.4 Certification Goals

The Modular Platforms are intended to support all safety needs, ranging from basic integrity up to SIL4. As such, ways towards a modularized certification for Modular Platform instantiations and their Functional Applications are going to be analysed in detail in Task 3 of this work package, with the results going to be available in deliverable D26.4. Currently, the expectation is to align this future work with the ERJU System Pillar Modular PRAMS activities.

Additional insights into how to approach modular safety for state-of-the-art systems can, for example, be found in the “SIL4 Cloud” research report [3].

3.5 Cyber Security for Modular Platforms

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.

These requirements can very likely be based on previous work, such as X2Rail-3 [17], X2Rail-5 [19], RCA/OCORA [1], EULYNX [2] as well as the appropriate standards for onboard and trackside systems (for example IEC 50701, IEC 62443, …). Additional input sources in the future can potentially be the System Pillar cybersecurity efforts and R2DATO work package 3 (task 3), but this depends on actual timelines and alignment. So far, WP26 participates in the SP Shared Services mirror group for updates related to security.

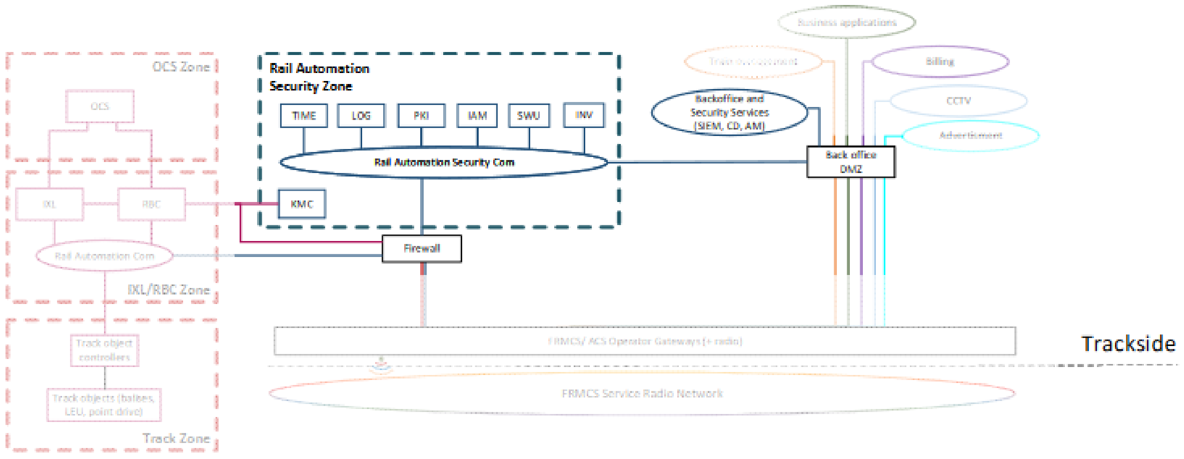

OCS Zone
OCS
IXL
RBC
Rail Automation Com
IXL/RBC Zone
Track object controllers
Track objects (balises, LEU, point drive)
Track Zone
Rail Automation Security Zone
TIME
LOG
PKI
IAM
SWU
INV
Rail Automation Security Com
KMC
Firewall
Business applications
Train ops assessment
Billing
Backoffice and Security Services (SIEM, CD, AM)
CCTV
Advertisment
Back office DMZ
FRMCS/ ACS Operator Gateways (+ radio)
Trackside
FRMCS Service Radio Network

Figure 6: X2Rail-3 Security Zone and Shared Security Services Overview, from [18]

The example figure above shows the shared security services from X2Rail-3:

- Time service (TIME)
- Central Logging (LOG)
- Intrusion Detection / Continuous Monitoring (IDS)
- Security Incident and Event management (SIEM)
- Identity and Access management (IAM)
- Backup (BKP)
- Asset Inventory (INV)
- Public Key management (PKI)
- Central Software Update (SWU)

These shared services could potentially match or be very similar to the service that are planned to be offered by the SP shared security services concepts. As such, they might find use for the modular platforms.

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, onboard and trackside, deployment options. This work is ongoing.

4 APPLICATION-LEVEL PLATFORM INDEPENDENCE (ALPI)

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.

The discussions presented in this chapter are based on the inputs from the SP CE domain (chapter 3.1, deliverable [14]), the architecture proposal in WP26's first deliverable [16] and the ongoing work in WP26 itself.

4.1 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 a monitoring. In addition, also the SRACs imposed on the application by the underlying platform have to be fulfilled, ideally standardized. A general goal should be to define as many common SRACs as possible, but ending up with exactly the same SRAC list shared between different platforms is most likely unachievable.

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. 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.

4.1.1 Previous Work as discussed in D26.1

Several pre-existing documents expressing previous thoughts on the topic influenced this document, especially this paragraph:

- 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.

4.1.2 Structure Overview

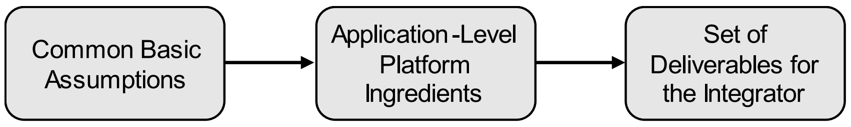

Common Basic Assumptions
Application-Level Platform Ingredients
Set of Deliverables for the Integrator

Figure 7: High Level Process of Application Development

The document tackles the endeavour in 3 different stages:

- "Common Basic Assumption" (see chapter 4.1.2.1)
- "Application-Level Platform Ingredients" (see chapter 4.1.2.2)
- "Set of Deliverables for the Integrator" (see chapter 4.1.2.3)

All of them are described in detail in the following chapters.

4.1.2.1 Common Basic Assumptions

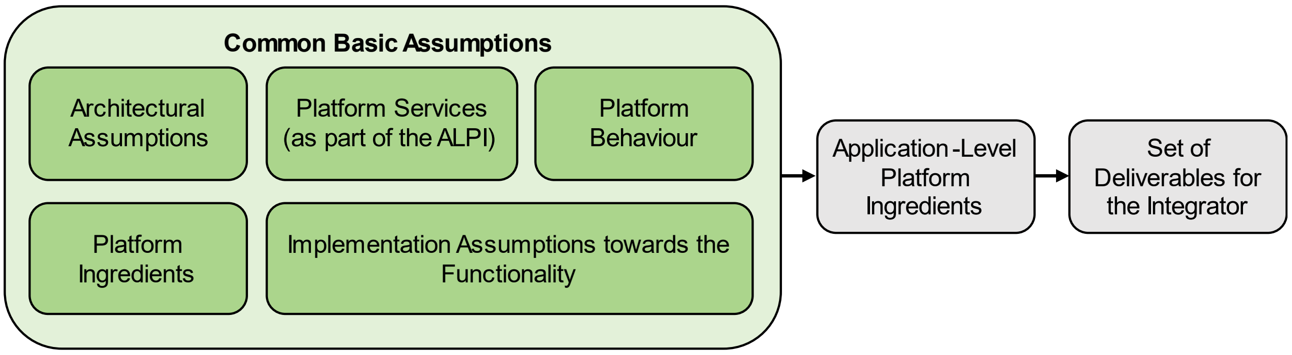

Common Basic Assumptions
Architectural Assumptions
Platform Services (as part of the ALPI)
Platform Behaviour
Platform Ingredients
Implementation Assumptions towards the Functionality
Application-Level Platform Ingredients
Set of Deliverables for the Integrator

Figure 8: Common Basic Assumptions Overview

This chapter summarizes basic and agreed assumptions. As a goal, a general direction of the result should be imaginable after reading this chapter.

4.1.2.1.1 Architectural Assumptions

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

1. Each application consists of one or more processes.
2. The platform can be assessed against relevant standards (at least CENELEC 5012x series). These standards influence architectural decisions for the modular platform concept.
3. Redundancy is supported, depending on and controlled via configuration.
4. Communication is message-based.
5. Run-to-completion scheme within a (sub)process.
6. Replica deterministic behaviour.
7. Clear application lifecycle, minimum: Start/Init → Operate → Stop/Shutdown

4.1.2.1.2 Platform Ingredients

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

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

4.1.2.1.3 *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)
- Process control (e.g. create, clone, delete)
- Memory management (e.g. malloc, free)
- Timing (e.g. time of day, sleep, different clocks)
- Communication (e.g. sockets, message queues)
- IO control (e.g. ioctl)
- Maintenance-related diagnostics and errorcodes
- Security functions
- Persitency functions

Note that parts of these services can/have to be safety relevant.

4.1.2.1.4 *Functionality Implementation Assumptions*

The design and implementation of an application that is targeted for the modular platform must fulfil certain assumptions concerning architecture, lifecycle, communication and diagnostics. 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 kind of assumptions and rules need to be well defined within the user documentation.

Topics are, non-exhaustive:

- OPC/UA or SNMP communication
- Safety communication (with different protocol implementations)
- Logging scheme
- Lifecycle: Start/Operate/Stop phases
- Redundancy and Safety configuration

4.1.2.1.5 *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.

- Replica management
- Communication synchronization
- Timing/synchronisation rounds
- Platform health indication and management data available
- Logging/diagnosis environment & behaviour
- External interfaces (see also chapter 6)

4.1.2.2 Application-Level Platform Ingredients

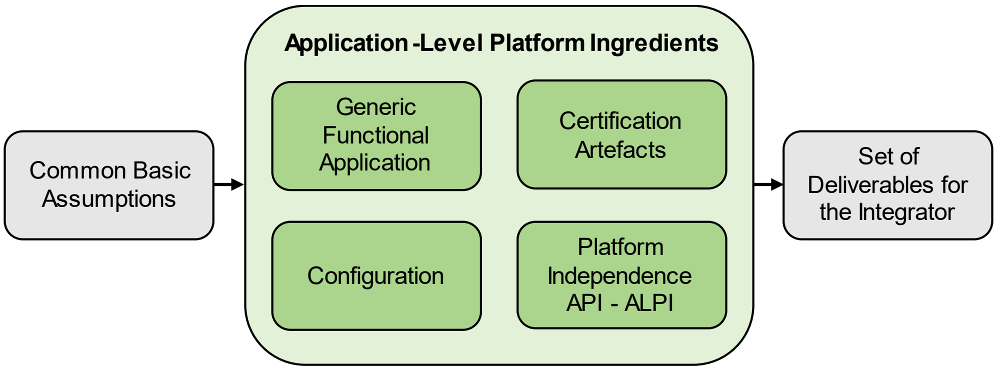

Application-Level Platform Ingredients
Generic Functional Application
Certification Artefacts
Configuration
Platform Independence API - ALPI
Common Basic Assumptions
Set of Deliverables for the Integrator

Figure 9: Application-Level Platform Ingredients Overview

An application consists of different parts, which are drafted in the figure above. Note that this description shall be explanatory only, a real realisation of such a platform can also add other parts, if needed.

4.1.2.2.1 ALPI Interface (PI API)

The ALPI interface (Application-level Platform Independence, previously: PI API – Platform Independent Application Programming Interface) is the concrete implementation how the 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.

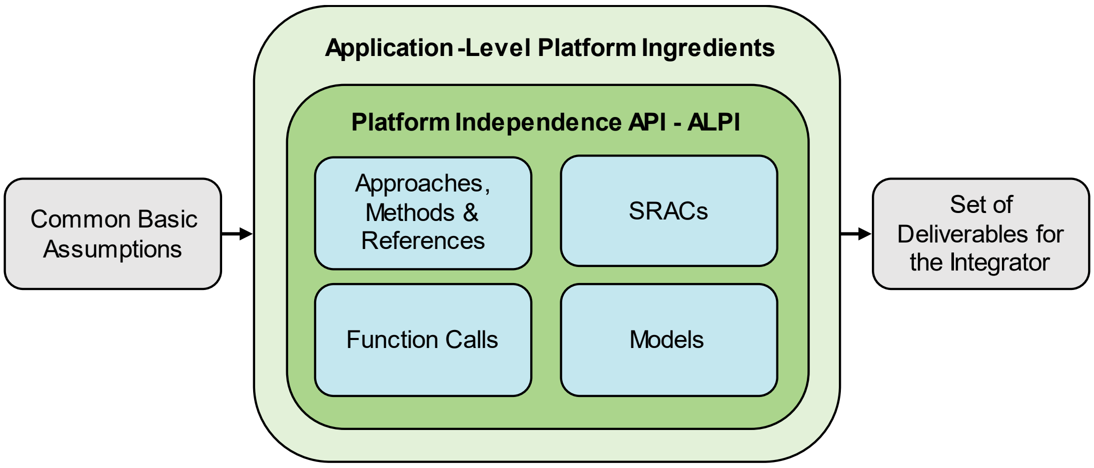

Application-Level Platform Ingredients
Platform Independence API - ALPI
Approaches, Methods & References
SRACs
Function Calls
Models
Common Basic Assumptions
Set of Deliverables for the Integrator

Figure 10: ALPI (PI API) Overview

- 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
  - 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. onboard)
  - Persitence and security functions
  - ...

4.1.2.2.2 Generic Functional Application

The Generic Functional Application implements a certain business logic into a piece of software, that might be accompanied by SRACs (from RTE), if necessary. “Generic” refers to the fact, that the goal is to enable development of different applications independent of a concrete execution platform. Additionally, the same application (at least source code) could also be used on platforms of different vendors if this is beneficiary and necessary.

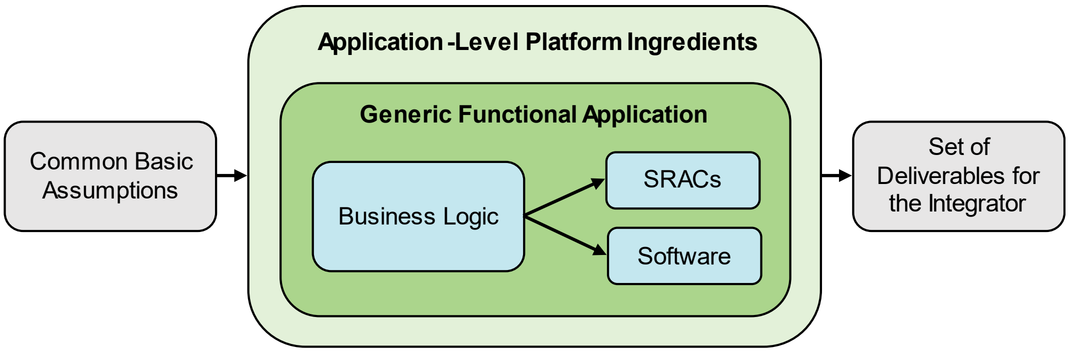

Common Basic Assumptions
Application -Level Platform Ingredients
Generic Functional Application
Business Logic
SRACs
Software
Set of Deliverables for the Integrator

Figure 11: Generic Functional Application Overview

As a general concept, the platform shall allow to run applications with different levels of criticality in parallel on the same platform. 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. Of course, this goal can only be reached if the safety solution supports it. And, changing of functionality and using same functions on a different RTE without doing anything regarding safety assessment will probably never be possible.

4.1.2.2.3 Configuration

Configuration data is consisting of at least two fundamental parts: The engineering data needed for the business logic to work, and the RTE configuration data for software execution. Often the engineering data itself is again partitioned into market/customer specific data and product generic data.

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 RTE specific configuration are:

- Communication endpoint configuration
- Communication protocol details
- Redundancy configuration
- Voting algorithms
- Security algorithms
- ...

4.1.2.2.4 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
  - RTE Security Conditions
    - Have to be defined and implemented from the RTE

      - But need to be fulfilled from application (according 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

4.1.2.3 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 environment
    - Depending on the concrete case: RTE, OS, HW, Virtualization, etc.
    - Test specs/cases to perform integration wherever possible

4.1.3 Modular Integration and Certification based on SP PRAMS goals

The discussion around this aspect of application-level platform independence has not started yet.

4.2 Assumptions

The discussion around this aspect of application-level platform independence has not started yet. While WP26 prepares a thorough list of assumptions for the next deliverable, the results presented elsewhere in the deliverable are based on the assumptions sourced as discussed in the intro to chapter 4.

4.3 Architecture and Layers

This chapter briefly continues the discussion from chapter 3.2 to introduce some additional aspects that need attention in our future work in modular platforms.

To account for mixed criticality including replication and voting, multiplicities have to be introduced to the diagrams shown in chapter 3.2, as well as a more detailed representation of what a Functional Application is made of (Functional Actors and their Replicas). However, to highlight the key message of the diagram, some details are dropped compared to the previous illustration in chapter 3.2. Also, details about how replication and message passing/synchronization are not shown. Furthermore, there is no implication towards the Computing Platforms shown being identical or different in any way, as this aspect of heterogeneity is for future study.

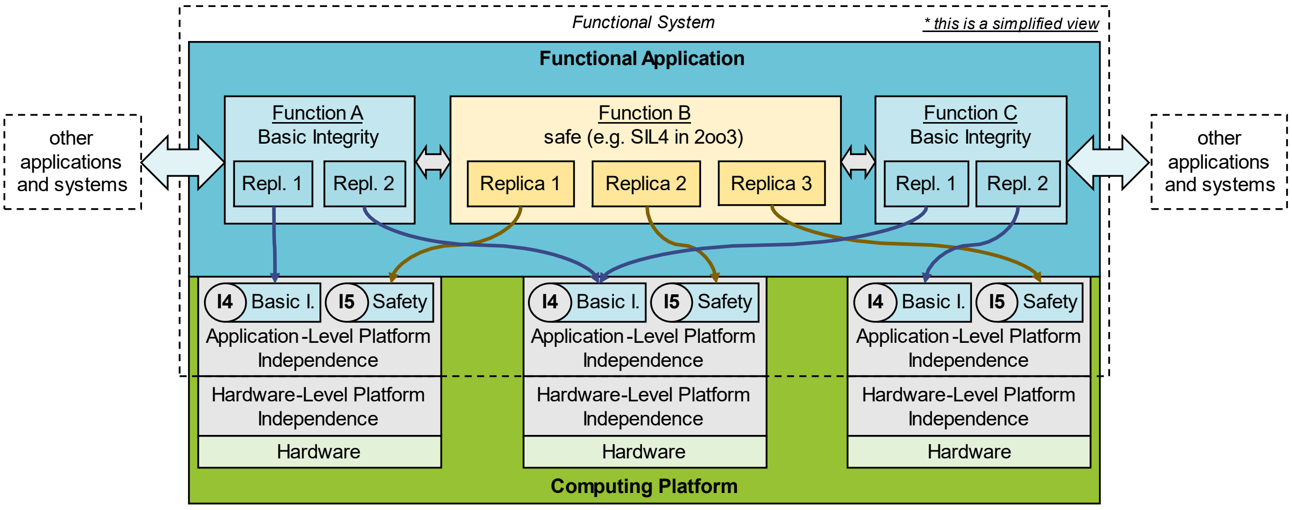


Figure 12: Modular Platforms instantiations executing a single mixed criticality Functional Application containing three actors of different safety needs (simplified view)

The view shown in the previous figure is very simplified and does not represent a real system configuration. Nevertheless, it was drawn to illustrate several aspects of the modular platform and also be a reference for the usage of the terms as defined in the glossary (see chapter 2).

The figure shows the following:

- a computing platform hosting a Functional Application
- a Functional Application consisting of three Functions (resp. Tasks)
  - Function A, Basic Integrity, with two replicas for redundancy; using I4
  - Function B, safe, with 3 replicas in a 2oo3 configuration for safety, using I5
  - Function C, Basic Integrity, with two replicas for redundancy, using I4

- communication flows from the outside and internal to the Functional Application
  - Function A and C each have a communcation partner outside of the Functional Application and outside to the Computing Platform
  - Function A and B as well as B and C communicate with each other inside of the Functional Application's scope

The figure omits at least the following details and aspects of a potential real-world deployment:

- system is not symmetric, as Functions A and C are not fully replicated
- there is no explanation how the communication flow to the external entities works and what protocols are used
- safety and redudancy measures are mixed
- systems of systems aspect not fully represented

A complete architectural overview will be presented in the subsequent deliverable, D26.3.

From a Modular Platforms architecture perspective, the potential deployment options proposed by the SP CE domain for Functional Systems are for future study in the context of this work package.

In the meantime, more details can be learned from previous work such as [6].

4.4 APPLICATION-LEVEL PLATFORM INDEPENDENCE APPROACH

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 specific hardware and software – allowing for portable applications.

4.4.1 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 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.

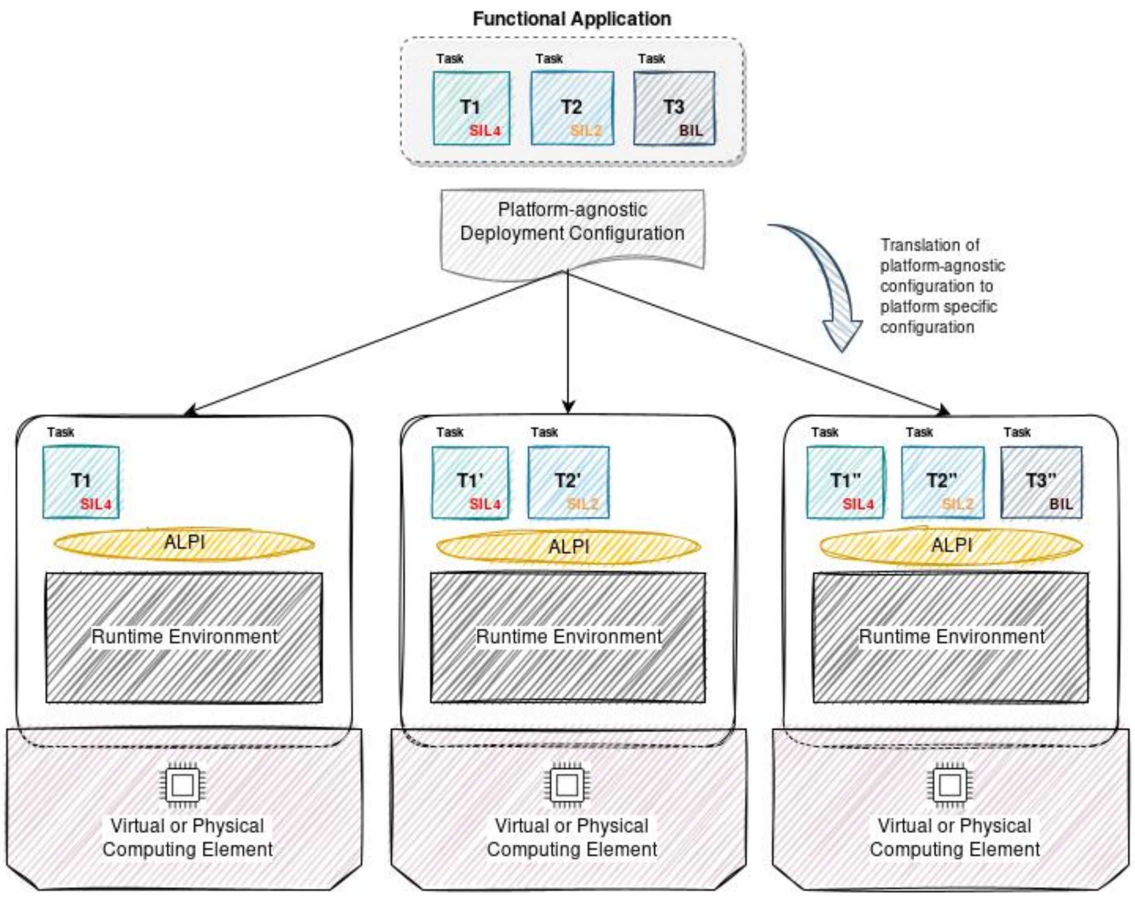

Functional Application
Task
T1
SIL4
Task
T2
SIL2
Task
T3
BIL
Platform-agnostic Deployment Configuration
Translation of platform-agnostic configuration to platform specific configuration
T1'
T2'
T1"
T2"
T3"
ALPI
Runtime Environment
Virtual or Physical Computing Element

Figure 13: Functional Applications, Tasks and Deployment Configuration

4.4.2 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.

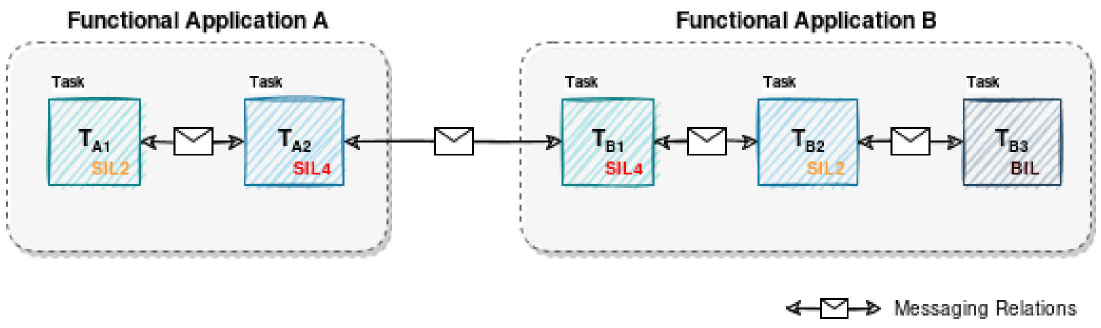


Figure 14: Messaging Relations between Tasks

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.

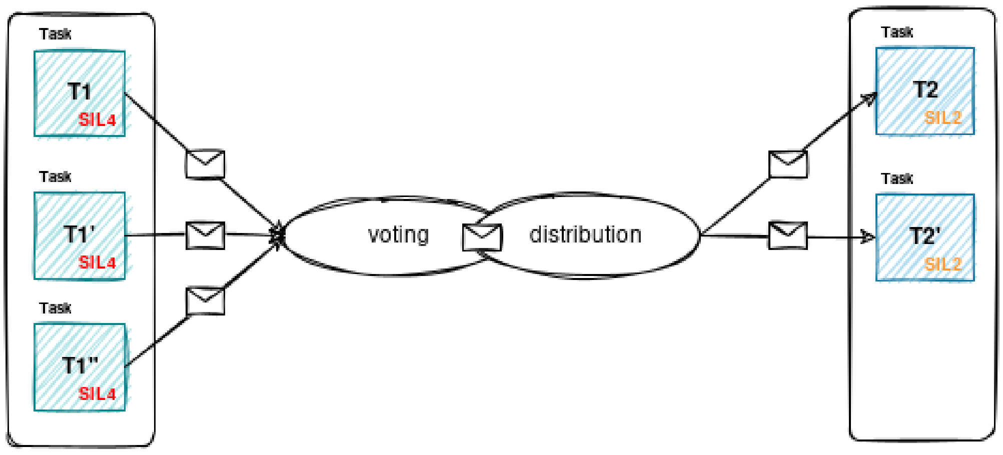


Figure 15: Message voting and distribution

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;
- Messages shall be time-stamped by the platform, 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)

4.4.3 Scheduling and timing

4.4.3.1 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.

4.4.3.2 Time

4.4.3.2.1 *Timestamps and Task replication*

The platform shall be able to provide timestamps with the following two 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 is requested). This is especially important if the timestamp is used in any voted output message.

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.

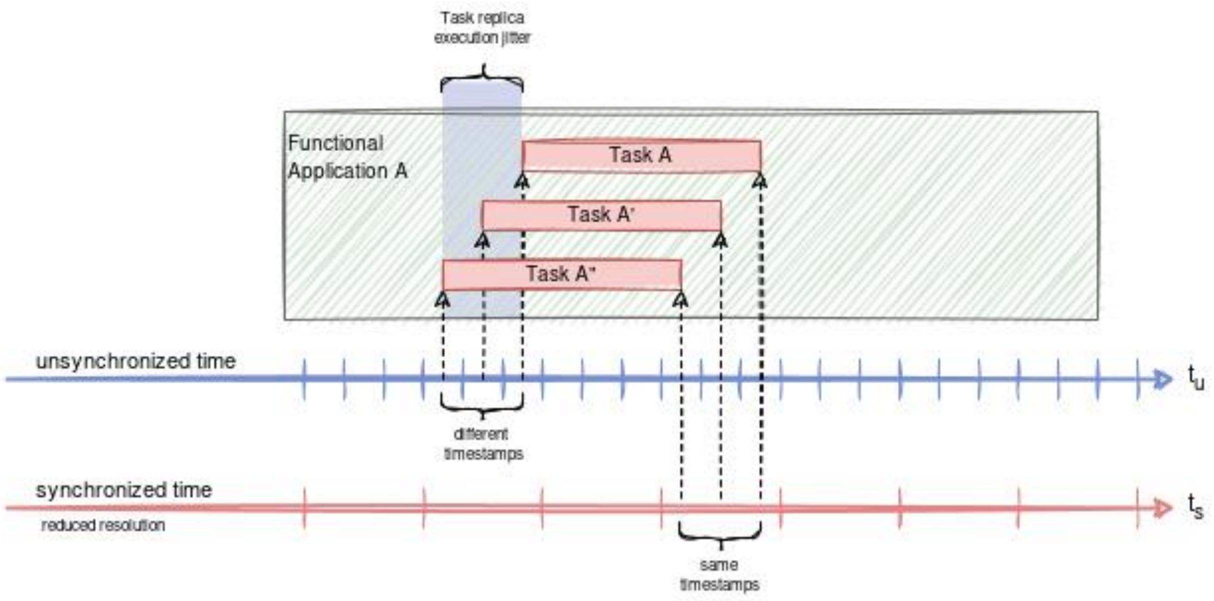


Figure 16: Unsynchronized vs. replica synchronized time

4.4.3.2.2 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.

4.4.4 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.

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.

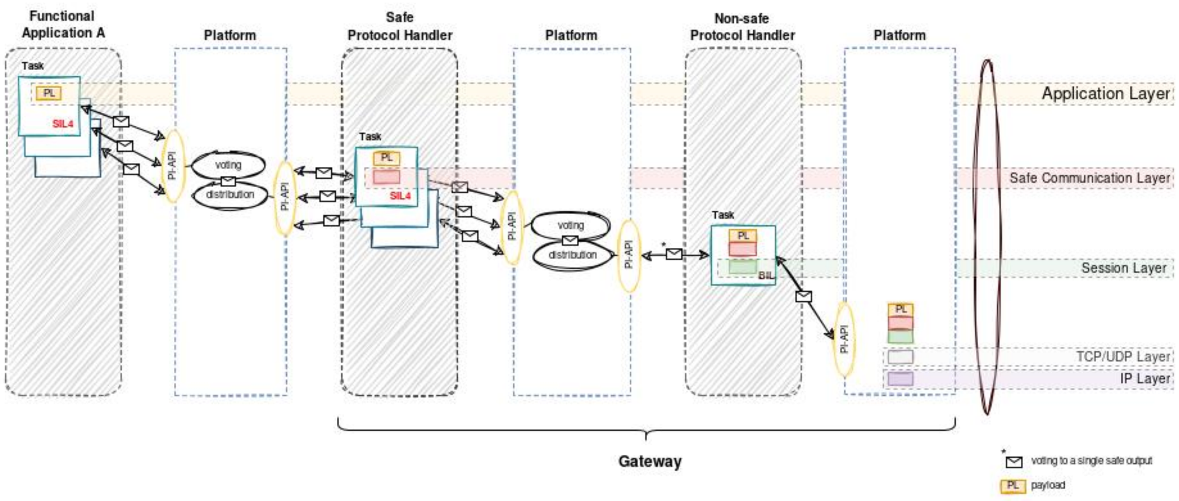


Figure 17: Gateway – contribution to protocol stack

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.

4.4.5 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:

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 4.4.5.1
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 4.4.5.2

Term	Definition in EN 50129:2018	Meaning in context of replicated Tasks	Expected platform behaviour
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 4.4.5.3

Table 2: Fault, Error and Failure in the context of replicated tasks

4.4.5.1 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.

4.4.5.2 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 3: 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).

4.4.5.3 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

4.4.6 List of Artefacts

The discussion around this aspect of the approach to application-level platform independence has not started yet. For now, please refer to the discussion in chapter 4.1.

4.4.7 API Components/Blocks

The discussion around this aspect of the approach to application-level platform independence has not started yet. For now, please refer to the discussion in chapter 4.1.

4.4.8 Programming Model

The discussion around this aspect of the approach to application-level platform independence has not started yet. For now, please refer to the discussion in chapter 4.1.

4.4.9 Communication Model

The discussion around this aspect of the approach to application-level platform independence has not started yet. For now, please refer to the discussion in chapter 4.1.

4.4.10 Diagnostics

With respect to diagnostics services and interfaces provided to the functional applications, there is so far no update compared to D26.1 [16], where the interface "APPL_DIAGNOSTICS" was mentioned. Integration of these services and interfaces is for future study, especially with respect to treating onboard and trackside environments.

4.5 Generic Functional Application

The discussion around this aspect of application-level platform independence has not started yet. For now, please refer to the discussion in chapter 4.1.

4.6 Configuration

The discussion around this aspect of application-level platform independence has not started yet. For now, please refer to the discussion in chapter 4.1.

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.

4.8 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
- 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

4.9 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.

5 HARDWARE-LEVEL PLATFORM INDEPENDENCE (HLPI)

A main goal of modular platforms HLPI in a new architecture is the aggregation of different Functional Systems with different safety integrity levels on same virtualization layer on same computing element (meaning same hardware) with best possible hardware independency.

For this it's necessary to define the details of the individual interfaces between hardware, virtualisation layer and Functional System software close to the safety architecture and security architecture regarding the handling of virtualization software and hardware as basic integrity parts without safety relevance and fulfilling security relevant requirements.

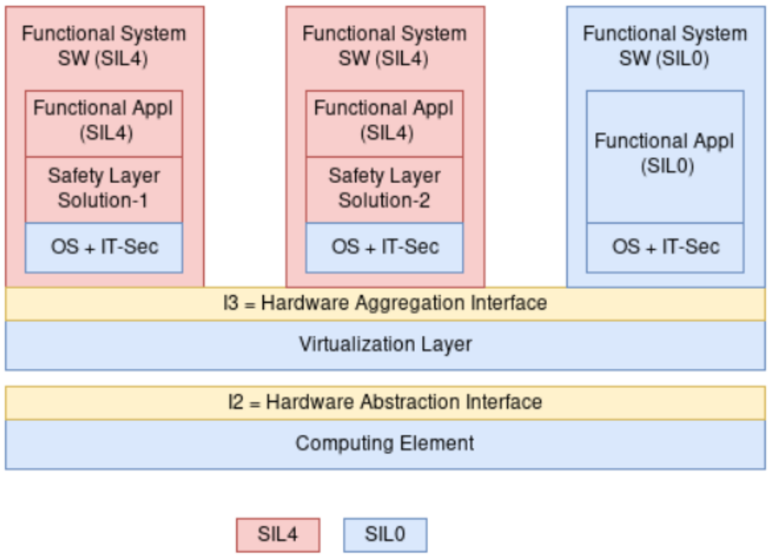


Figure 18: Aggregation of Functional Systems

This aspect as part of the modular platforms work is new compared to the previous deliverable and was introduced by the SP CE domain.

Figure 18 shows the proposed safety-architecture for flexible and efficient handling of aggregated systems (on the same computing element) provided by different suppliers. For this a "basic integrity software basis" (possibly provided by a third party) below the safety critical software parts is essential to achieving the complete decoupling of the aggregated systems.

Open issues:

- safety-concept of a SIL4 hypervisor running on COTS hardware
- requirements from view of safety-layers regarding the usage of hypervisor to find out if a common approach for different safety-layers is possible or not.

5.1 VIRTUALISATION INTERFACE

The interface I3 of the virtualization layer describes the basic aspects regarding the virtualization layer in context of aggregation of several systems with possibly different safety criticality levels and possibly provided by several vendors running together on the same hardware.

This interface 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.

In this especially the architecture "different solutions of safety concepts within SIL4 Functional Systems are running aggregated on same virtualization" has to be analysed to define the basic requirements for the virtualization layer from view of different safety layers running above.

It depends on the solution specific the 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 basic integrity virtualization layer.

Such aspects as Hypervisor Type-1/2, Container Solution (without Hypervisor) need to be discussed and analysed.

Furthermore, the requirements to the safety layer from the view of "aggregated running in basic integrity virtualization" need to be discussed and defined to be sure that the different possible safety-layers will allow to be run aggregated with any other software on same virtualization layer. Dependencies between the virtualization layer and the different variants of software running above shall be identified, especially from view of safety and security.

A list of potential requirements needed to define this interface is following (subject to further study).

- Requirements from Functional Systems running above regarding the virtualization layer:
  - flexible usage of guest OS on virtualization layer
  - partitioning of CPU resources (e.g., core mapping, ETH communication)
  - time related aspects (performance, reaction time of systems)
  - decoupling of systems (flexible handling of the individual Functional Systems)
  - safety related aspects required by safety layers (with different safety concepts)
  - security related aspects required by security layers (with different security concepts)
- Requirements from the virtualization layer towards the software running above:
  - Requirements to the safety layers to achieve “aggregation of different software on basic integrity virtualization”
  - Requirements to the operating systems above regarding IT-security

5.2 Hardware Abstraction Interface

The interface I2 of the hardware is not an interface in sense of "programming interface" but it is the definition of required CPU characteristics as processor instruction set, needed CPU features as performance, etc. from view of the Functional System software running aggregated on the virtualization layer above.

In this context the topic of "flexible and HW-independent usage of HW" has to 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 touching the software of the Functional Systems running above.

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:
    - Flexible support of different variants of hardware
- Requirements from the virtualization layer 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)

5.3 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 basic integrity parts as the COTS based hardware or in common SW layers as the virtualization can be handled without impact onto safety- and security-certification of the Functional Systems running above.

5.4 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.

6 INTERFACES EXTERNAL TO THE PLATFORM

Interfaces external to the platform are important for integration and interoperability of modular platforms into the railways computing landscape. The external interfaces discussed here are platform-centric and do explicitly not cater to the needs of the functional applications executed on such a modular platform.

With regards to alignment of the interfaces, there are several SP domains (CE, TCCS) working on processes and requirements that need to be fulfilled by upcoming specifications for these interfaces. This alignment process in ongoing and not yet finished. Especially trackside and onboard differences on these interfaces need a thorough discussion in the future.

Based on the input from the SP CE domain [14], the following tables shows how the new names for the external interfaces subsumed under the identifier “I1” relate to what this work package discussed in its previous deliverable D26.1 [16].

SP CE domain: I1 interface	deprecated WP26 D26.1 naming [16]	Comment
IF-ORCHESTRATION	PLAT_UPDATE	Update, configuration and maintenance of the computing platform and its functional applications.
IF-DIAGNOSTICS	PLAT_HEALTHMGMT	The SP CE domain definition of diagnostics focuses on providing the health status of the computing system and its components (e.g., functional application status, runtime status, hardware status, etc.) to a central diagnostics system.
IF-LOGGING	PLAT_LOGGING	Provide logging information towards a central logging service.
IF-IT-SEC	PLAT_SYNC PLAT_SECURITY	Time synchronization and shared security services (e.g., Identity and Access Management – IAM).

Table 4: Overview and mapping of I1 interfaces

The individual interfaces are discussed in the following subchapters.

6.1 IF-ORCHESTRATION

The I1-Interface IF-ORCHESTRATION contains many aspects of operation and maintenance of a computing platform. They are discussed in the following subchapters.

6.1.1 Update and Configuration

Update and configuration, potentially of both, platform and applications, is another challenging topic in the context of functional safe systems, usually coupled with high availability expectations. For now, this deliverable shows a first discussion around aspects and challenges of such a common interface and its implications on the modular platform itself.

6.1.1.1 Open Topics and Challenges

The following bullet points are only showing a snapshot. The list is not exhaustive.

- Granularity needs to be defined. What can be updated alone, what needs to be updated together, so that the system is always in a consistent state after an update? This is relevant for both, platform components (such as the RTE) and the functional applications being executing. In this context of granularity, how is configuration data handled? What changes need to happen at the same time, what can be decoupled?
- Trusted supply chain for update data needs to be established and not be limited to a single trusted source, as different functional applications might need updated executables and configuration data from individual and/or redundant sources. Additionally, push and pull approaches and preloading of data (especially for onboard systems) need to be discussed.
- Impact analysis of potentially shared files (e.g. configuration data, engineering data, map data) between functional applications and the consistency of the application's operation and guarantees a "certified system" after update steps. How is this checked? What kind of rollback is needed in the case of consistency failures? From a modular platforms perspective, how can this be achieved without being tailored but generalized instead?
- The modular platforms can only define one side of the interface, but an infrastructure is needed as well. Alignment ongoing.
- EULYNX definitions especially in basic context of maintenance SMI (remote update) should be taken into account in context of interface I1. In this also such experiences as "incompatible changes between EULYNX Baselines (e.g., differences in defined update processes between EULYNX Baseline 3R5 and 4R2)" should not be ignored. As an additional challenge, right now the SMI specification of EULYNX is not detailed enough for an implementation.

6.1.1.2 Existing Input Sources for Update and Configuration

There is previous work for update and configuration management interfaces, as shown below. The analysis and alignment ongoing, however.

- EULYNX Standardised Maintenance Interface (SMI)
  - Preloading and activation of certified software and configuration data.
  - Not yet ready for an interoperable implementation for modular platforms.
- OCORA concept (TWS07-06) Configuration management [8]
  - Provides a concept for configuration management based on "building blocks".
  - "Manifest Files" are part of the building block descriptions, see chapter 5.2.1f in the document.
  - Further investigation necessary.
- "the update framework" [20]
  - Provides generic specification and example implementations for a trusted supply chain for data files. It provides no direction on what to do with the data files once they are downloaded.
  - Can be investigated as a distribution mechanism for data files.

6.1.2 Orchestration of Applications on Hardware

Orchestration and hardware management, e.g. for failover scenarios on spare hardware, need a detailed discussion, both from the external interface side (I1) as well as from the perspective of the modular platform itself. First thoughts on these aspects and their challenges are given in this chapter, all subject to further study and alignment.

In the topic "orchestration of SW running on computing elements" the main goal is to achieve a common approach for all kind of software, means for all kind of Functional Systems running on the computing elements.

In the scenario "aggregation of several different Functional Systems on the same computing element" new challenges in context of "remote orchestration" arise:

- The orchestration of individual Functional Systems shall be completely decoupled, means there shall be no inter-dependencies between the Functional Systems. E.g., a SW-update for an individual Functional System-1 shall not touch another Functional Systems-2/3/4, ... which are running on the same computing element.
- The process for remote update of safety-relevant Functional Systems shall fulfil the safety-relevant requirements based on a safety-case for remote update.
- Safety-case for remote update is necessary (considering the independency of Functional Systems).

In the trackside scenario "centralized data centres" additional challenges arise:

- For efficient handling of spare parts in the centralized data centre (to replace failed computing elements) the remote installation of individual SW instances onto another computing element – possibly located at another data centre – shall be possible.
- This flexible spare-handling shall be supported by the orchestration processes and shall be automized by the orchestration-tool (behind I1) in best possible way to avoid manual maintenance activities as good as possible.
- The mechanism for automatization shall consider the different installation scenarios for the different possible variants of system SW configurations (2oo3, 2x2oo2, ...).
- For each **automized orchestration mechanism** for safety-critical Functional Systems realized within the orchestration-tool behind I1 **a safety-case is necessary.**
- An overall-management of the provided and used CPU resources of the computing elements is necessary behind I1.

6.1.3 Policy Updates

The discussion around this aspect of IF-ORCHESTRATION has not started yet.

6.2 IF-DIAGNOSTICS

The discussion around this aspect of I1 has not started yet.

6.3 IF-LOGGING

The discussion around this aspect of I1 has not started yet.

6.4 IF-IT-SEC

The discussion around this aspect of I1 has not started yet.

6.5 CERTIFICATION

The discussion around this aspect of interfaces external to the platform has not started yet.

6.6 CONCLUSION AND OUTLOOK

The interfaces external to the modular platform have many dependencies to the outside infrastructure and also to the inner workings of the modular platform itself. As such, future work and alignment are needed, especially to clarify this work packages focus on relevant aspects of these interfaces.

7 CONCLUSIONS

A coherent and useful description of the universal modular platforms concept is a complex task. This intermediate deliverable of the work package's second task shows the current state of the work. It is based on its previous deliverable D26.1, discussing the state-of-the-art and numerous alignment meetings within the work package, R2DATO and the appropriate System Pillar domains.

One important step forward with respect to managing the complexity of modular platforms was introduced in this deliverable: The separation into three distinct domains – application-level platform independence (ALPI), hardware-level platform independence (HLPI) and external interfaces – helps with architecture, specification, implementation, and deployment.

For each of these domains, the current and intermediate state of the work was presented, additional to an introductory overview over the modular platforms concept.

The next deliverable of this work package, D26.3, will present the final results of the modular platforms specification task and will show the continuation of the work presented here. Based on this deliverable, the work package's final task, Task 26.3, will then focus on a study on modular certification and acceptance approaches for modular platforms.

Work package 26 will continue to closely align with ERJU's System Pillar Computing Environment Domain and the Transversal CSS Components Domain to deliver modular platform specifications that are well integrated into the overall future railway system.

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 – Common Business Objectives
https://rail-research.europa.eu/wp-content/uploads/2022/10/SP-Common-Business-Objectives.pdf

[16] ERJU Innovation Pillar FP2 R2DATO, Work Package 26, Deliverable D26.1
https://projects.rail-research.europa.eu/eurail-fp2/deliverables/
Note: Document was not yet released at the time of writing.

[17] X2RAIL-3
https://projects.shift2rail.org/s2r_ip2_n.aspx?p=X2RAIL-3

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

[19] X2RAIL-5
https://projects.shift2rail.org/s2r_ip2_n.aspx?p=X2RAIL-5

[20] The Update Framework
https://theupdateframework.io/