RCA
Reference CCS Architecture

OCORA

An Approach for a Generic Safe Computing Platform for Railway Applications

A joint White Paper from the Initiatives RCA and OCORA

Version 1.0, July 2020

Editor: Patrick Marsch (DB)
Contributors (in alphabetical order): Albert Ledermann (SBB), Christian Daniel (SNCF), Conny Woelk (SBB), Detlef Brückner (DB), Dirk Spenneberg (DB), Frank Eschmann (DB), Frank Skowron (DB), Hélène Arfaoui Kaynak (SNCF), Jan Welvaarts (NS), Jean-Baptiste Simonnet (SNCF), Markus Burri (SBB), Markus Kuhn (SBB), Max Schubert (DB), Oliver Mayer-Buschmann (DB), Patrick Marsch (DB), Prashant Pathak (DB), Thomas Martin (SBB)

1. Introduction

The railway sector is currently undergoing the largest technology leap in its history, with many railways in Europe and across the globe aiming to introduce large degrees of automation in rail operation. Beyond the rollout of the European Train Control System (ETCS), most railways are for instance aiming at introducing Automated Train Operation (ATO), in some cases up to fully driverless train operation (grade of automation 4, GoA4), and an automated dispatching of rail operation, typically referred to as a Traffic Management System (TMS).

In this context, the railway initiatives Reference Control Command and Signalling Architecture (RCA) [1] and Open Control Command and Signalling Onboard Reference Architecture (OCORA) [2] are driving a functional architecture for the trackside and onboard functions for future rail operation. More precisely,

- **RCA** is an initiative by the members of EUG [3] and EULYNX [4] to define a harmonised architecture for the future railway Control Command and Signalling (CCS), including a definition of components and interfaces among these, with the main goal to substantially increase the ratio between performance and total cost of ownership (TCO) compared to today's implementations.
- **OCORA** is first and foremost a platform for cooperation to the benefit of the European railway sector. Recognising that a coherent, modular, upgradeable, interchangeable, reliable and secure onboard architecture is paramount to overcome the challenges for the overall CCS system, the intent is to establish the OCORA onboard architecture in coherence with and complementarily to the trackside control command and signalling.

It is obvious that a future-proof, modular functional architecture is only one essential step toward the digitalisation of rail operations. Beyond this, it is also vital that the rail sector maximally leverages latest advances in the IT sector, for instance related to Cloud technology and high-performance computing. In this context, RCA and OCORA are jointly working toward a **generic safe computing platform approach** for onboard and trackside CCS applications (and possibly other railway applications), in particular aiming to decouple applications from the underlying computing platform, considering their very distinct life cycles, and to achieve platform independence.

This White Paper is jointly issued by the stated initiatives and provides

- the high-level objectives behind the design of a generic safe computing platform for railways;
- key design requirements and design paradigms which have been agreed upon;
- a description and comparison of two possible platform approaches;
- a discussion on how a safe computing platform for railways could leverage concepts and experience from the automotive and aviation sectors; and
- the envisioned next steps towards the specification of the API between application and platform.

The purpose of this White Paper is to create awareness within and beyond the railway sector on the ambition and initial plans of leading railways to introduce a generic safe computing platform for railway applications, to stimulate broad discussion on this, and to obtain early feedback from related application and platform vendors on key design principles and possible platform realisations.

The considerations on onboard and trackside computing platforms presented in this paper are driven mainly by the needs of safety-relevant CCS applications like ETCS and its further evolution. However, they shall not be limited to such applications, but potentially also be used for other, possibly non-safety-relevant applications (wherever a reuse of the same platform design or even same physical platform appears beneficial).

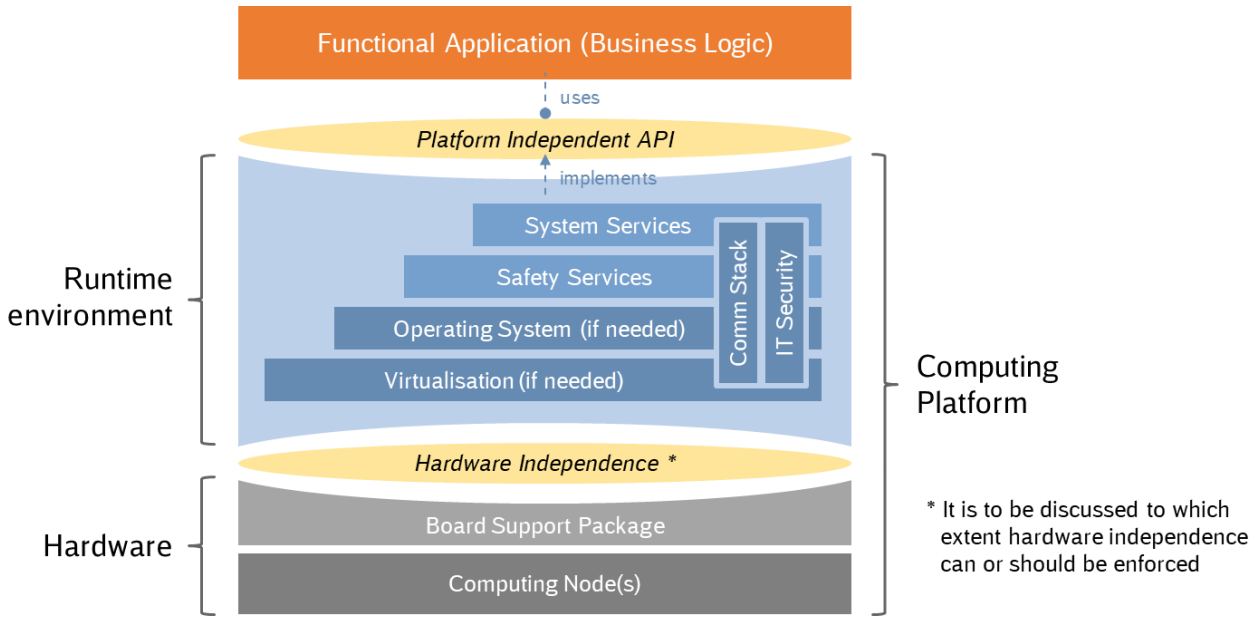

Figure 1. General computing platform principle and terminology.

2. Notion of Platform Independence and overall Platform Terminology

A key term used throughout this document is the notion of **Platform Independence**. Platform independence is achieved when an application, based on a **generalised abstraction** between the application logic and system interfaces, runs unchanged on different platform implementations. This is seen as a key prerequisite for the **portability** of applications among different platform environments, which in turn is a key enabler for the reduction of development costs.

Note: This White Paper considers platform independence from a computing platform perspective only. It does neither address vehicle independence nor bearer independence, which are both important aspects of modularity, but which are dealt with in other workstreams of RCA and OCORA.

Beyond the notion of platform independence, the following terminology is applied throughout this paper, as also shown in Figure 1:

- **Functional Application** refers to a software (SW) implementing the actual business logic of a railway function (e.g., that of a so-called Vehicle Locator or Vehicle Supervisor, as examples of CCS functions according to the RCA architecture);
- **Computing Platform** refers to the environment on which applications are run, comprised of
  - **Hardware** (i.e., compute nodes, memory, etc.) and
  - **Runtime Environment**[1], itself comprised of Safety Services, System Services, the communication stack for information exchange among applications on the same platform and with external entities[2] and possibly (depending on the actual platform implementation) an operating system and virtualisation.
- **Platform Independent (PI) API** refers to the aforementioned general abstraction that allows applications to run unchanged on different computing platform implementations.

[1] Note that the term "runtime environment" here covers a broader scope than in AUTOSAR, e.g., also encompassing "basic software" and "platform foundation" acc. to AUTOSAR terminology.
[2] It is an open design question if the communication stack should be logically separated from the rest of the runtime environment or not.

Further, the term **Tools** refers to all necessary tools to develop and test a product based on the platform (e.g., to test and configure the platform and develop, compile, integrate and test applications). To foster the development of easily exchangeable and competitive tools, and avoid single source, a description of a **standard development process** (i.e., defining the artefacts needed for each step and the transition between them) may be released with the platform. This may for instance take orientation in the AUTomotive Open System Architecture (AUTOSAR) [5] methodology, see also Section 6.

As indicated in Figure 1, it is also envisioned to have some extent of **independence between runtime environment and hardware**, again considering their different market ecosystem and life cycles. More precisely, any runtime environment implementation should be able to run on a decently wide range of commercial-off-the-shelf (COTS) hardware, so that hardware can be added or replaced as technology advances. It is, however, understood that it may be difficult to enforce this in the short and medium term for onboard systems, as here often embedded solutions are used with a strong cross-optimization between runtime environment and hardware.

In more detail, **Safety Services** are expected to comprise functions that ensure that the computing platform satisfies the railway norms EN 50126, EN 50128 and EN 50129 by covering:

- Integrity checking;
- Fault tolerance (e.g., achieved through redundancy and voting, or through diagnostic functions);
- Synchronisation and communication services related to safety (e.g., needed for fault tolerance);
- Hardware and software monitoring as needed in safety context.

System Services are expected to comprise functions that cover at least:

- Application lifecycle management (incl. start-up, supervision, restart, stop, update, recovery);
- Platform and software monitoring (process, software stack and interface modi);
- Tracing and logging;
- Communication services (not related to safety) incl. network management and compression;
- Access to external entities, including sensors and actuators;
- Security, including means for authentication, (HW accelerated) encryption, key storage, etc. (possibly provided in a centralised way by some platform implementations);
- Persistence services, e.g., provision and management of persistent storage.

3. High-Level Objectives

The high-level objectives for a generic safe computing platform are listed in Table 1.

Table 1. High-level objectives.

No.	Objective	Applicability to	
		Onboard	Trackside
1	**Meet safety and real-time requirements of CCS (and similar) railway applications**. The platform shall meet safety requirements of applications up to safety and integrity level (SIL) 4, e.g., acc. to EN 50126, EN 50128 and EN 50129, and support applications with real-time characteristics (e.g., overall processing cycles in the order of 10-100ms).	✓	✓
2	**Respect diverse lifecycles of business logic, runtime environment and hardware**. The platform shall be partitioned with respect to the different lifecycles of business logic, runtime environment and hardware. The platform shall support fully independent life-cycle handling, i.e., with minimal dependencies.	✓	✓

3	**Open market to new players**. The platform shall open the market to new, non-rail-oriented software and tooling companies. They shall be able to become involved in functional application development without providing their own platform safety mechanisms (e.g., related to safe communication, fault tolerance implementation, etc.).	✓	✓
4	**Minimise total cost of ownership**. The platform shall minimise the total cost of ownership, i.e., the overall life-cycle cost.	✓	✓
5	**Vendor independence**. Different vendors shall be able to provide functional applications, computing platforms and development tools, respectively, without a vendor lock-in. It shall be possible to purchase hardware directly from different vendors throughout the lifespan of the software. The platform shall build on existing HW/SW solutions, stimulating competition among vendors and allowing them to shine with their specific expertise and distinctive solution features.	(✓) * with possible limitations in HW choice, at least short-term, due to highly integrated solutions	✓
6	**Industrial readiness**. It shall be possible to procure a platform as off-the-shelf solution supported by an open and dynamic market. The solutions shall be mature (e.g., reliability proven in field) and backed by effective acceptance and integrated logistical support (e.g., maintenance service, tooling, availability of spare parts).	✓	✓
7	**Migratable and portable business logic**. The business logic is considered a significant system asset, being the component with the longest lifetime. It must hence be portable to different computing platform evolutions. We here further differentiate: • **Migrateability for legacy applications**: It should be decently easy to migrate legacy applications to the new platform; • **Portability for new applications**: Applications running on the platform should be portable to any other vendor's or evolved version of the platform.	✓	✓
8	**System evolvability**. The platform shall be open to extensions (in the sense of additional system services that are added over time, e.g., related to FRMCS). Adding new functionalities shall be possible with minimal to no changes to existing applications (though these may naturally not be able to leverage the new functionalities).	✓	✓
9	**Facilitation of application development**. The platform shall use an open, well-documented application model and programming interface, facilitating that third parties develop applications.	✓	✓
10	**Modularity**. The platform shall allow for a modular safety certification process, using pre-certified components leading to a dramatically simplified and shortened full system certification process. An evolution or update of the platform shall not require a new E2E homologation of application and platform, as detailed in Section 7.	✓	✓
11	**Encapsulated, transparent fault tolerance mechanism**. The platform shall transparently encapsulate the safety and fault tolerance mechanisms. Vendors may offer different (new) approaches to safety and fault tolerance as they become available on the market - solution agnostic and future-proof.	✓	✓
12	**Scalability**. The platform shall be highly scalable, i.e., it should by design be able to support an arbitrary number of applications and arbitrary number of compute nodes.	(✓)	✓

13	**Flexible usage of compute resources.** The platform shall enable a flexible mapping of business logic to compute resources (e.g., such that the platform can be expanded while applications are running, and that business logic can be re-mapped when compute nodes fail). It shall be able to leverage advances in computing technology (i.e., when better compute nodes are available, it shall be possible to assign more instances of business logic to the compute nodes).	✓	✓
14	**Centralisation.** The platform shall allow to centralise applications physically in a safe data centre to simplify life-cycle management, reduce TCO by means of simplified, optimised operations, and benefit from increased availability and optimised resource usage.		✓
15	**Support for running multiple applications (also with different SIL levels) on one physical platform.** It shall be possible to run multiple applications, possibly with different SIL levels, on a single physical platform to reduce cost, space, power dissipation, etc., and simplify certification, maintenance, system evolution, etc.	✓	✓ (though need less pronounced as for onboard)
16	**Life-cycle management capabilities.** The platform shall provide automated mechanisms related to software lifecycle and configuration management, diagnostics, etc. This may also be expanded to software development automation, e.g., taking orientation in SPEM.	✓	✓

From Table 1, it is already visible that there is a high level of commonality among the objectives for onboard and trackside computing platforms, suggesting a common platform approach for both sides. It should be noted that it may obviously not be possible that a particular computing platform design fulfils all requirements, and that compromises may hence have to be accepted.

4. General Computing Platform and PI API Design Considerations

In order to achieve the aforementioned high-level objectives, it is expected that a safe computing platform for digital rail shall follow at least these main design paradigms:

- **Satisfy railway norms such as EN 50126, EN 50128, EN 50657 and EN 50129 in their application up to SIL4:** provide functional applications with safe, reliable, deterministic and real-time performance as well as the level of availability required;
- **Clear separation of concern between the functional application and the platform:** Notably, functional applications shall handle only business logic, while all other required functions to maintain the application's life-cycle (incl. mechanisms for safety, fault tolerance, persistence, communication and application management) shall be handled by the computing platform in a way that is transparent to the business logic;
- **Provide a mechanism for safe and secure communication with external entities:** (e.g., with on-board peripheral devices and external systems via the OCORA Gateway [6]);
- **Implement a harmonised PI API:** possibly common for onboard and trackside;
- **Follow a modular safety concept:** to minimise homologation efforts;
- **Maximise usage of COTS components (CPU, I/O, SW, etc.), tools and Open Source software (where applicable):** to minimise vendor lock-in and leverage advances in other sectors;
- **Provide mechanisms to sufficiently isolate applications**, in particular when involving different SIL levels, which are running on the same physical platform.

While this is not seen as mandatory, it is expected to be beneficial if computing platforms also

- **Consider utilising virtualisation techniques or similar means of abstraction of computing resources:** for better evolvability, scalability, the support of mixed SIL constellations and a more flexible mapping of applications to compute resources.

It should be noted that beyond the agreed basic design paradigms above, the railways are currently consolidating a comprehensive list of requirements for the generic safe computing platform.

Application and Communication Model

We will now venture into details on the possible design of a generic safe computing platform for railway applications, in particular regarding how functional applications interact with the platform and with external entities. It should be noted that all subsequently described aspects are still to be seen as proposals and subject to further study. Furthermore, the focus is mainly on safety-relevant applications; for non-safety-related applications, obviously leaner mechanisms could be envisioned.

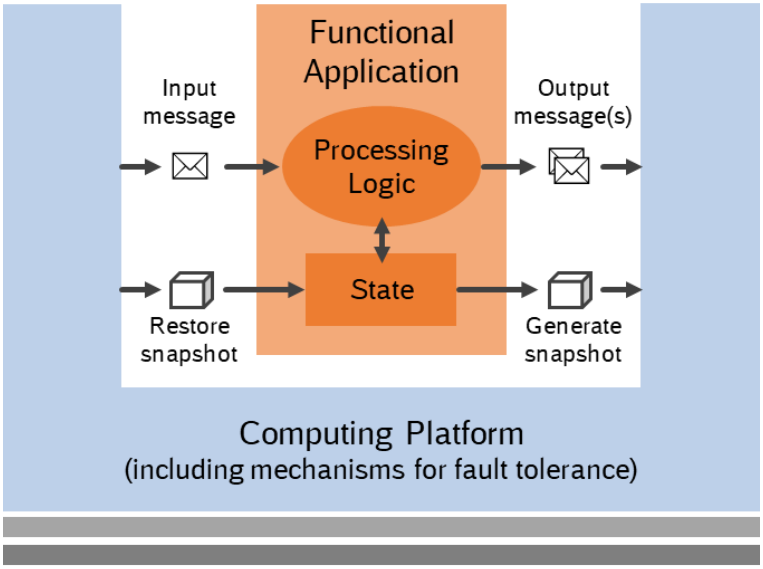

Figure 2. Illustration of application model.

While this may not be the only approach to meet SIL requirements, it is in the following assumed that "composite fail-safety" according to EN 50129 is used for achieving fault tolerance. This means that a functional application is run multiple times (for M-out-of-N redundancy), and the results of the replicated instances are compared to provide one overall safe output. It is here assumed that the functional application consists of a processing logic and a state, and that some form of State Machine Replication (SMR) is used. This allows to support a big range of approaches for achieving fault tolerance, which, as stated before, is considered to be handled by the platform and transparent to the functional application. The resulting application model is illustrated in Figure 2.

The SMR requires that the functional application has a defined initial state and is deterministic in the sense that with a given state and given message received, it always results in exactly the same change of the state and the same sequence of sent messages. It is in particular not allowed that there is randomness in the execution or the sent messages, dependency on local or overall timing, or dependency on the local node state. One example of how a computing platform could run a functional application for achieving sufficient fault tolerance is shown in Figure 3, where the functional application is run in three replicas.

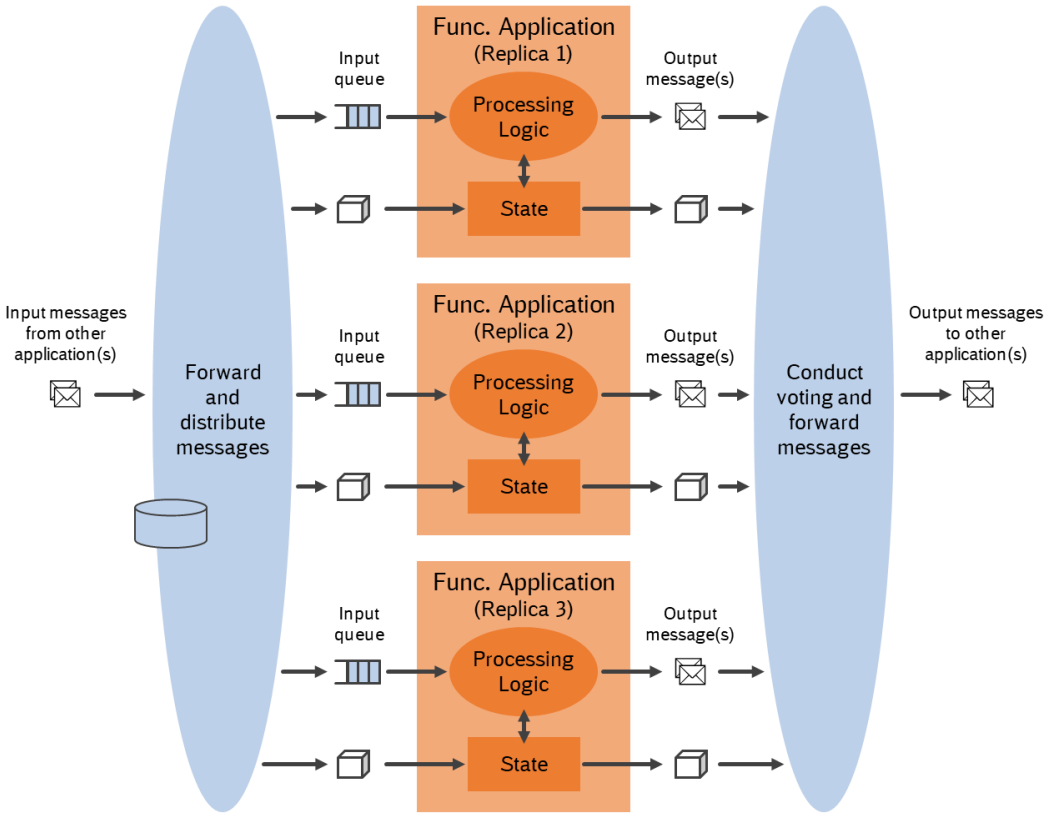

Figure 3. Illustration of possible usage of functional application replicas for fault tolerance (assuming a composite fail-safety approach and highly simplified for brevity).

Here, the message distribution logic ensures an identical sequence of messages delivered to the functional application replicas by using a so-called atomic broadcast. As the application replicas all have the same initial state and are deterministic, they will produce an identical sequence of sent messages and will go through the same sequence of states. The voting logic then compares the sent messages and detects inconsistencies. In case of such, it provides an error indication which is processed further by additional monitoring mechanisms to ensure a safe reaction of the overall system.

It is expected that the computing platform provides state management and handles all procedures needed to (re-)start functional application replicas after a crash, fault detection, or maintenance work. It may do so by requesting a snapshot of the state from still running replicas and then providing the snapshot to the started replica, before starting to provide the input messages.

The communication between different functional applications running on the same computing platform could be based on asynchronous messages, based on which also synchronous communication could be realised. In this respect, the computing platform should ideally support point-to-point, point-to-multipoint and publish-subscribe approaches, as shown in Figure 4.

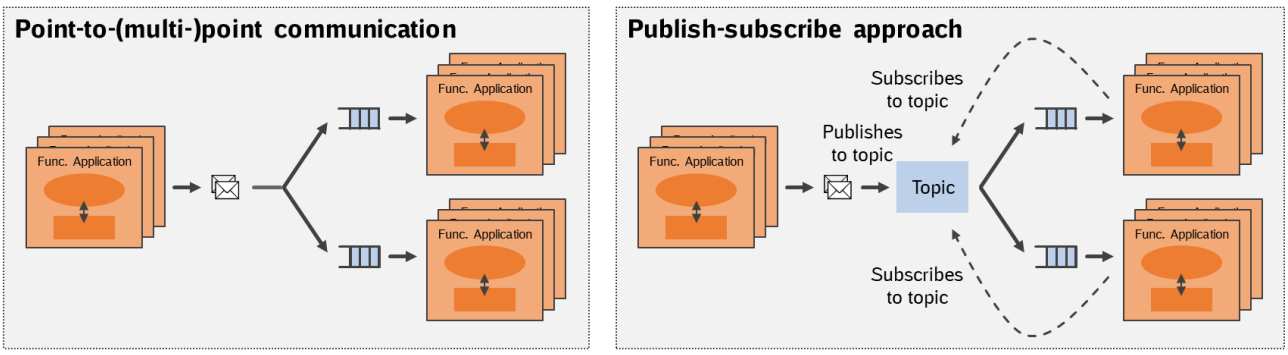

Figure 4. Forms of communication among application instances that the platform should support.

The receiver or topic would likely be specified by the sender with an address following a standardised addressing scheme. A target functional application of messages should be informed with presence information when a critical source functional application is either no longer running or the communication from the source is interrupted, such that the target application can react to this, if required.

Interfacing to External Systems

It is expected that the computing platform also provides mechanisms for the interfacing to external systems, for instance through the notion of a "gateway" that implements the communication stack that is used to the external system, as depicted in Figure 5.

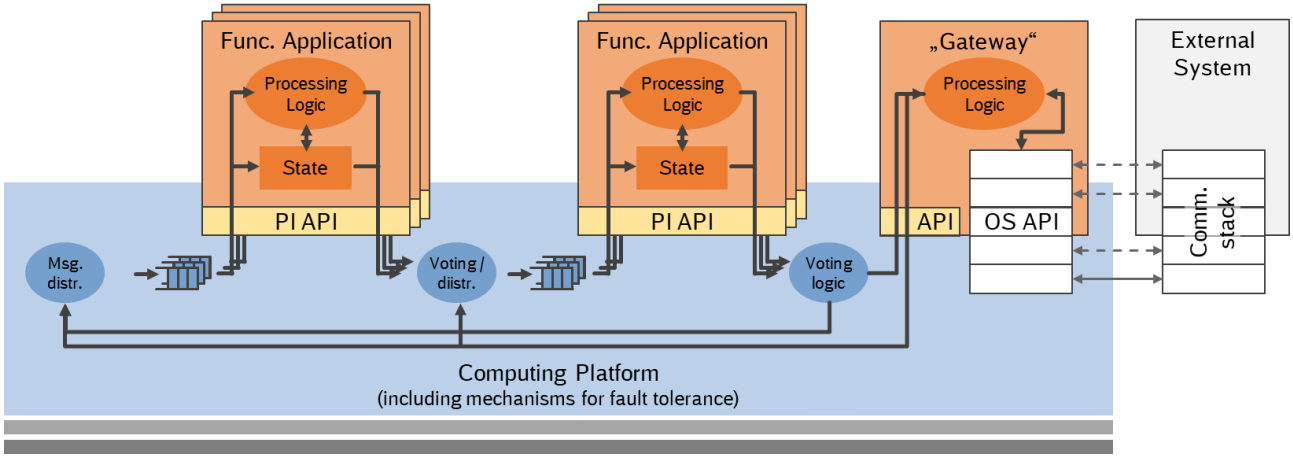

Figure 5. Possible "gateway" for communicating with external system.

The communication to one external system can be based on a "single active communication link" (e.g., single TCP connection), in which case the exchanged data must be protected by information redundancy such as parity bits, a Cyclic Redundancy Check (CRC) code or Message Authentication Code (MAC) that is generated by the application (before the voting).

A form of gateway could also support the communication to many external systems (e.g., trains), in which case the message between the application and the gateway may contain the identification of the external system to which the message must be forwarded, respective from which the message was received. Further, the gateway and the application exchange information about the establishment and termination of "communication links". It should be noted that it is to be clarified up to which level in the Open Systems Interconnect (OSI) protocol stack is covered by the gateway.

Expected Capabilities of the Platform Independent API

In consequence of the previous sections, it is expected that the API between applications and platform covers at least the following functions:

- Functions for applications to communicate with other application instances or with external systems (such as sensors and actuators), either based on a point-to-(multi)point communication or a publish-subscribe logic;
- Functions via which applications can determine or be informed about the presence of other (platform-internal or external) application instances;
- Functions related to tracing and logging, security and access to persistent storage.

IT Security Considerations

IT security supports digitalisation and cannot be captured as a single requirement. Instead, IT security comes with a plethora of requirements related to law, norms, safety and life cycle.

Law and norms (incl. regulatorily mandated standards) foresee a dynamic life for security components to follow constantly changing threats and vulnerabilities. Safety and life cycle, on the other hand, focus on much longer release cycles from economic but mainly operational point of view. That is why the basic requirement and design principle should be "security as a shell". In particular, the security concept should only demand a minimum possible support from the safety system to fulfil the requirements, and the security shell should allow for maximum flexibility.

This principle applies to interfaces as well as to systems and sub-systems themselves and must be applied continuously throughout patch and release management. Especially for releasing security updates it is necessary to implement processes that allow to quickly respond to vulnerabilities and conduct testing, without requiring certification by federal accredited organisations when following a risk-based analysis. In addition, an outstanding capability is 'detection'. Detecting security incidents is quite common for trackside infrastructure but totally new on vehicles. In the context of the introduction of a generic safe computing platform, detection should be natively enabled on both sides while protecting and respecting the distinct security zones on both sides.

5. Possible Realisations of the Computing Platform and PI API

While there is a broad common understanding on the key design principles of a generic safe computing platform, the likely relation and information exchange between applications and platforms, and the main functions to be supported by the PI API, there is no conclusion yet on the exact realisation of the PI API. In the following, two possible platform and PI API approaches are shortly introduced and assessed.

Approach where applications are programmed against PI API

In this approach, applications are programmed against the PI API, which includes

- a set of hardened libraries implementing a well-defined, standardised set of system functions covering the required API capabilities as listed in Section 4, and
- a well-defined, standardised set of safety related application conditions (SRACs) that every application must comply with in order to get safety certified.

To run an application on a specific platform, the application must be compiled for a target system (OS and CPU architecture) that is supported by that specific platform (noting that a platform may support one or several target systems), for which there are in principle two options:

- The platform supplier provides or determines all tools to be used by the application provider to develop, integrate and test the application for that specific platform (possibly imposing further SRACs on the application), or
- the target system (OS and CPU architecture) is specified and the application provider chooses the tools that support this target system.

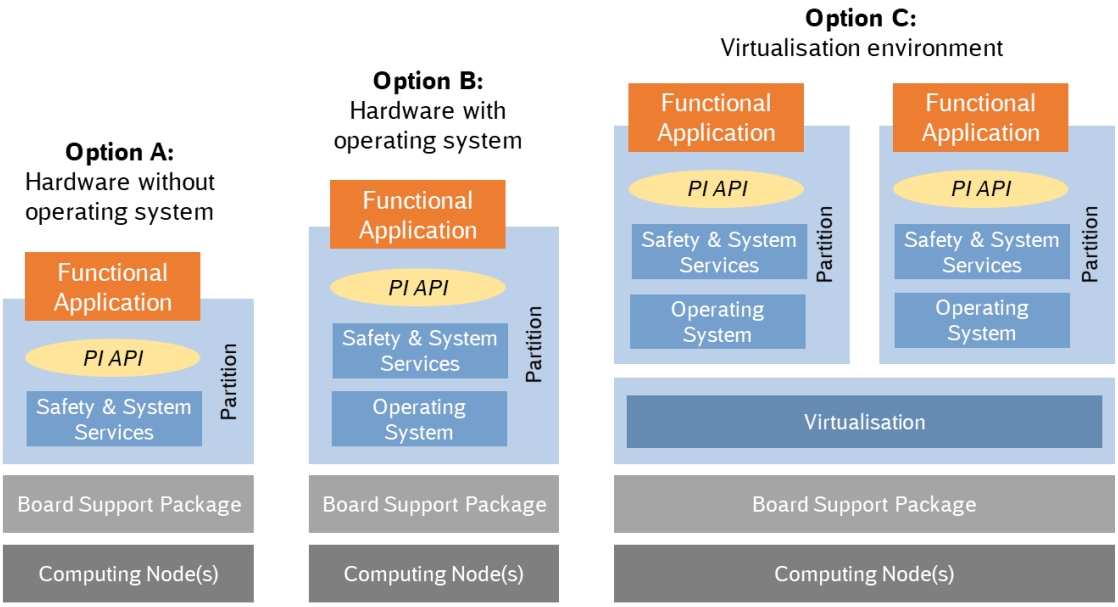

Figure 6. Possible platform options where applications are programmed against PI API.

Applications programmed against the PI API are at minimum source code portable, or possibly even binary code portable, between different platform implementations. All safety-related functions not inherent in the application logic are implemented as part of the platform. Key properties are:

- The application does not depend on how the underlying system is built and which mechanism is used to achieve the required safety, availability and performance. For instance, a supplier could realise the platform based on virtual machines on COTS servers, allowing that applications run in different virtual environments with regards to supported OS and version, etc. Alternatively, however, a supplier could provide a platform based on embedded HW, possibly without virtualisation or even without an OS, i.e., based on “bare metal”. Exemplary options out of a plethora of possibilities are illustrated in Figure 6. Note that the term “partition” here refers to isolated execution environments with guaranteed processing and memory resource;
- The platform supplier must provide an overall concept for the mechanism that is used to achieve safety and availability. It also has to provide the safety case showing that the platform executes a correct software with the required failure rate. This includes a standardised set of SRACs that an application provider needs to comply with.

Approach where applications run with a guest OS in virtual machines or partitions

In a possible alternative approach, applications run (likely together with a guest OS) in virtual machines (VMs) or partitions on the platform. The PI API, again fulfilling all the functions listed in Section 4, could here for instance be based on socket communication. Key properties of this approach, as shown in Figure 7, are:

- Each application runs in its own virtual machine or partition;
- The application provider provides the content of the virtual machine or partition of its application, which includes a "safe" guest operating system and mechanism for safety (like health monitoring) that must run in the virtual machine or partition. The content of the virtual machine must be compatible to the used virtualisation technology defined by the platform provider. The application provider must provide a safety case showing that the used "safe" operating system and mechanism for safety running in the virtual machine provide together with the platform the required failure rate;

- The platform supplier provides the virtualisation platform which ensures that applications run in sufficiently safe environments (before voting) and provides additional mechanisms for safety that run outside of the applications' virtual machines, like voting.

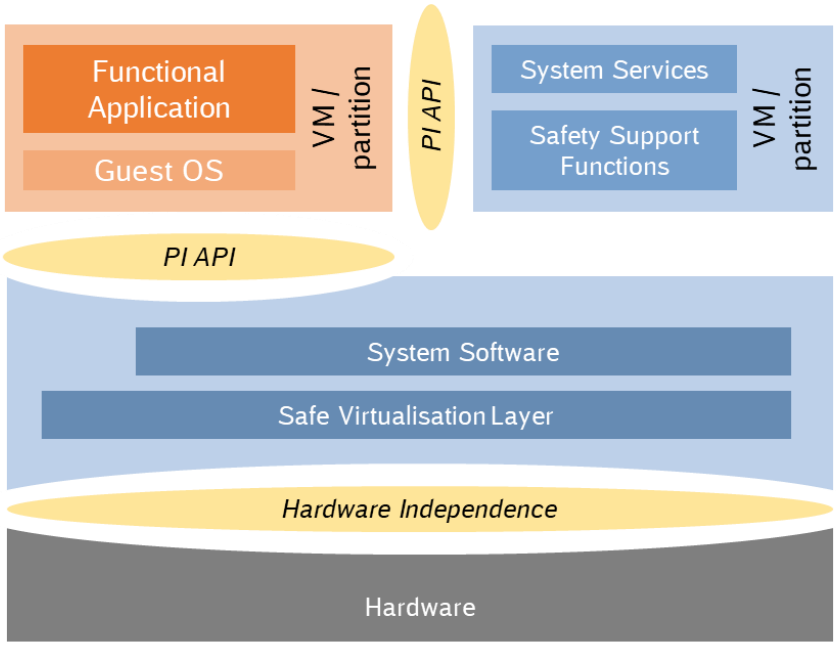

Figure 7. Possible approach where applications run with a guest OS in virtual machines or partitions.

Preliminary comparison of the two approaches

It should be noted that both described platform and API approaches in principle fulfil all objectives stated in Section 3, but there are some subtle differences regarding how well some of the objectives are met, as elaborated in Table 2.

In general, one should stress that the two approaches may also be combined, i.e., on the same physical platform different API levels could be foreseen that correspond to the two compared approaches.

Table 2. Extent to which described platform and API approaches fulfil selected high-level objectives.

Objective	Approach where application is programmed against API	Approach where applications run with guest OS in VM or partition
2 Respect diverse lifecycles (of application, platform, etc.)	The platform can be exchanged (possibly requiring re-compilation of the application) with different technology or different safety concept as long as the PI API remains the same.	The same principle applies, but the application VM and the application itself may have to be modified if the virtualisation approach changes.
3 Open market to new players (in particular new application providers)	As safety and fault tolerance are handled entirely in the platform, application programmers can focus on business logic only while adhering to the SRACs. Therefore, a larger supplier market is expected to develop.	Application providers typically also have to provide a "safe" operating system with health monitoring capabilities. However, this may also be chosen and provided by the platform vendor, so that the application vendor need not bother about this.
5 Vendor independence		

Objective	Approach where application is programmed against API	Approach where applications run with guest OS in VM or partition
(regarding range of possible OSs and CPU architectures)	The platform may implement a range of OSs and CPU architectures or an approach without OS (e.g., allowing to scale down to small system).	Depending on the hypervisor product, the virtual machine can host a broad variety of guest operating systems.
(regarding freedom for application provider to choose OS and CPU arch.)	OS and CPU architecture are defined by the platform (which may support multiple architectures, for instance through virtualisation)	Application provider may choose all OSs and CPU architectures supported by the chosen virtualisation technology.
(regarding freedom for application provider to choose tools)	If tools are chosen by the application provider, these are limited to those for the required OS and CPU architecture. If tools are provided by the platform vendor, only supported programming languages can be used.	Because the operating system is determined by the application provider, this has full flexibility regarding the tool chain to be used.
6 Industrial readiness (regarding reuse of existing solutions, etc.)	Platform suppliers may offer their existing platform extended with the Platform Independent API. In general, the approach provides openness for platform vendor differentiation and further platform evolution.	Platform suppliers must have a virtualisation platform including safety and fault tolerance mechanisms supporting a wide range of guest OS and CPU architectures to support solutions of existing application providers.
10 Modular safety certification process	A modular safety assessment and authorisation is likely facilitated, as detailed in Section 7.	A modular safety assessment and authorisation is likely a bit more difficult, as detailed in Section 7.

6. Comparable Approaches from other Industries

The platform approaches elaborated in this document can be compared to approaches used in the automotive sector (AUTOSAR [5]) and in avionics (IMA [7], typically using the real-time operating system or RTOS interface definition ARINC 653 [8]), which have a long development history and proven record of implementation in their industries. Despite some differences across the sectors regarding application lifecycles, market size, equipment cost, etc., AUTOSAR and IMA / ARINC 653 share quite some commonality with the computing platform approach envisioned for the railway sector:

- A standardised layered architecture (of 3 or more layers) is used to decouple hardware and upper software layers;
- Based on a standardised runtime environment, the application developer focuses on the business logic only, minimising the efforts to reduce the risk of faults in the lower software layers.

AUTOSAR and IMA / ARINC 653 are designed for onboard applications with stringent requirements on bus communication, networking and environmental factors, which also apply to railway onboard systems (and partially to trackside systems), hence both architectures may provide some inspiration for the design of railway computation platforms.

A technical analysis of **ARINC 653** shows that it would likely fulfil objective 1 from Section 3 in the way that it provides mechanisms for process management, time services, fault isolation and health monitoring that are suitable to meet safety requirements for a single application. However, it may not meet the real-time requirements of the railways and likely does not support mixed-SIL setups according to objective 15 due to limitations in partition and memory management. Further, a flexible usage of resources as per objective 13 may be limited as ARINC 653 does not allow runtime (re-)configuration of the system.

AUTOSAR would likely meet objectives 1 regarding both safety and real-time support as it provides various means related to partition, process, time, memory and communication management, fault isolation and health monitoring. However, it likely fails to support objective regarding mixed SIL support, and the static configuration scheme of AUTOSAR likely also makes it difficult to meet objective 13 related to a flexible resource usage.

To better support computing-intensive tasks and more flexible architectures in the vehicle needed for todays and future use cases like automated driving, multimedia applications and over-the-air software updates, a new **AUTOSAR Adaptive** platform [9] has been introduced, which for instance breaks with the static configuration paradigm. AUTOSAR Adaptive would likely be able to better fulfil the railway requirements, as it also foresees:

- A service-oriented API to access the hardware and network resources, thus simplifying the hardware and software integration;
- Ability to develop Electronic Control Unit (ECU) applications independently of one another in distributed work groups;
- Multiple applications with different SIL level can be run on the same platform, in order to improve performance and reduce certification cost;
- Dynamical linking of services and clients during runtime;
- Applications can be reconfigured on spare hardware modules if the primary module is detected faulty during operations, increasing the overall availability of the applications.

On a different level, the IEEE **Time-Sensitive Networking** (TSN) suite of standards is in development and could gain the capabilities relevant for critical applications in automotive, industrial and IoT applications. Other technologies such as Software-Defined Networking (SDN), DetNet or Wavelength Division Multiplexing (WDM) can expand the range of system integration options in critical integrated systems over the longer term (10-15 and more years).

The robotic sector is also currently developing a middleware supporting robotics, known as **Robotics Operating System** (ROS). In the long term, middleware approaches could be capitalised for standardised railway middleware solutions supporting good performing computing platforms.

In conclusion, none of the most popular computing platform approaches in the automotive and aviation sectors is perfectly applicable to the railway sector, but the approaches from other sectors clearly provide a source of inspiration and orientation. It should be stressed that different from the other sectors, the railway sector is characterised by an internationally rather harmonised set of railway applications and a rather limited vendor ecosystem. This may in fact make it possible to strive for a more decisive and bold computation platform and API design than a design that has to accommodate the individual needs of a large number of market players, as in the case of AUTOSAR.

7. Considerations on Certification and Authorisation

Certification and authorisation are important aspects to be anticipated in the computing platform design as they are today a key cost driver and often delay the start of operation. The challenge is particularly pronounced when using digitalised and virtualised functions which are not state-of-the-art solutions in the railway sector.

The starting point for the European railway sector for certification and authorisation is the mandatory application of standards EN 50126 - EN 50129 in the CCS regulations (often referred to as the EN CCS regulations). Classically, when a railway application and the underlying computing platform are provided monolithically by the same vendor, the vendor provides the end-to-end safety assessment to obtain the authorization for a specific implementation of both application and platform.

When application and platform are separated and potentially provided by different vendors, it becomes essential that the safety assessment can be performed on a modular basis. For instance, a platform vendor could certify that a specific platform implementation fulfils the stated norms for any generic application, as long as the application fulfils a set of safety-related application conditions (SRACs) provided with the platform. In return, an application vendor would assess that a specific application implementation meets the relevant norms and satisfies the SRACs of the platform. Based on this, a subsystem integrator could then obtain the overall authorisation for both application and platform, as shown in Figure 8. It is expected that this form of modular safety assessment may slightly differ for the two platform and API approaches compared in Section 5, namely:

- In the approach where applications are programmed against the PI API, the overall subsystem approval would likely be rather simple and could be conducted by any entity (i.e., the platform vendor, application vendor or a third party), as the safety mechanisms reside solely in the platform;
- In the approach where applications reside in virtual machines or partitions, it may be required that the application vendor also provides the overall subsystem integration, as safety mechanisms in application and platform have to jointly contribute to the safety case.

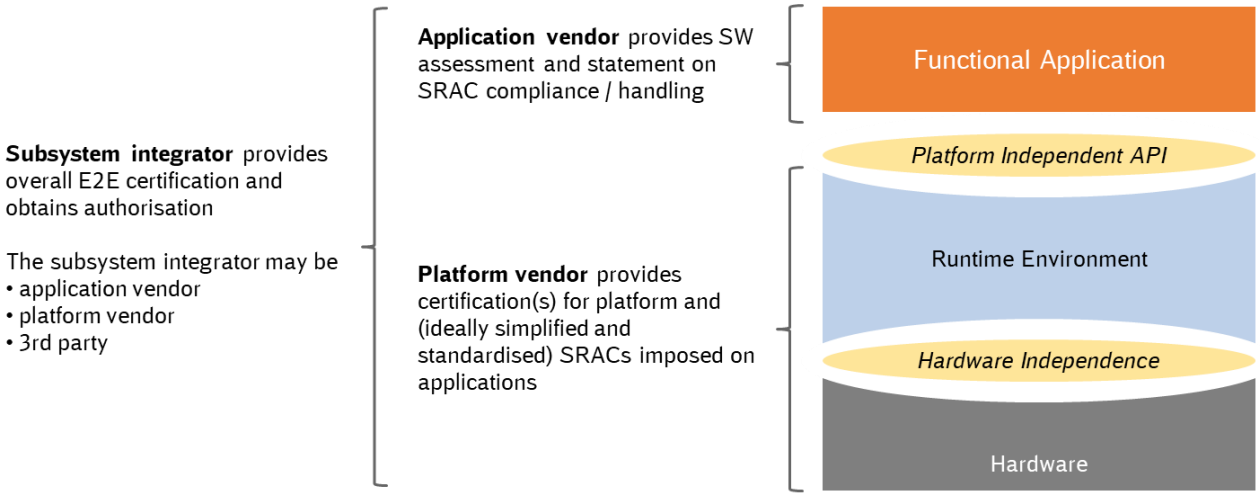

Figure 8. Expected responsibility split in the context of a modular safety assessment.

An important aspect to consider is also how (re-)authorisation is handled when for a given application and platform setup the platform is modified or replaced. A first and somewhat intuitive step would be to standardise the SRACs imposed by the platforms, as then the effort for re-authorisation of the application could likely be minimised. In the longer term, it would be desirable to aim for an authorisation approach that allows replacing the platform without any re-authorisation of the application.

Clearly, a modular safety assessment is only possible if there is a clear split of safety-related mechanisms in application and platform, and a clear separation of hardware resources for safety-critical and non-safety-critical applications, as discussed earlier in this paper. In general, it is in this respect of course desirable that (standardised) SRACs are maximally simplified.

Required evolution of the certification framework

Enhancements of certification (process and regulatory framework) will serve the computing platform design when they contribute to:

- **Improved verification and validation (V&V) at subsystem and system level** (e.g., allowing that a functionality can be changed or added in a CCS onboard without the need to perform testing for each train type and class): Means and methodologies for checking non-regression (from specification to design and V&V) should be developed and standardised, and a virtual environment (lab) shall be an acceptable solution for validation and certification;
- **Authorisation for generic configuration**: Investigation is needed on the possibility to certify as interoperability constituents either the computing platform, the software application hosted by it, the peripheral connected to it, or each of these components. The certification process of interoperability constituent(s), as it is set in the interoperability directive (EU) 2016/797, could be a practical solution allowing to capitalise on a single certification for the component valid for a wide range of use.

In other industries, there are several alternatives (e.g., IEC 61508) for the EN CCS regulations that provide a similar level of quality and safety, are globally accepted, are regularly applied in safety-critical industry branches including the transportation industry and are more suitable regarding the application of state-of-the-art technologies. In principle, also the EN CCS regulations allow for a use of similar standards, if equivalence can be demonstrated. The core activities of equivalence demonstration, if anticipated and agreed at sectoral level, can reduce the time-to-certification and consequently the time-to-market and delays in the authorisation process.

8. Summary and Next Steps

In the context of the ongoing digitalisation of rail operation, leading railways envision to introduce a generic safe computing platform, with the particular aim to separate railway applications from computing platform technology and hence reflect the different life cycles of the domains, and to leverage latest advances in the IT sector, while still meeting the stringent homologation requirements for safety-critical railway applications.

As detailed in this paper, there is already a common understanding among key railway players on the high-level objectives for a generic safe computing platform, which indicates a large extent of synergy among trackside and onboard platform requirements, and there is an agreement on key design principles. Further, detailed concepts are available regarding the likely API between applications and computing platform, and for possible platform realisation approaches, though these are to be further assessed in discussion with potential supplies and ultimately to be assessed through prototyping. Also, many questions are still open regarding how the railway sector could possibly leverage solutions or at least the experience from the automation and aviation sectors, or on how authorisation and certification could be handled more efficiently when applied to a new safe computing platform.

The envisioned next steps are the following:

- In the months following the publication of this paper, detailed interaction among railways, railway application vendors, platform vendors and system integrators is envisioned (e.g., via

technical workshops). The objective here shall be to refine applicable performance parameters and design criteria in order to develop a comprehensive list of requirements;

- Based on obtained feedback and the comprehensive list of requirements, it is envisioned that the PI API specification is drafted by end of 2020;
- From 2021 on, different platform approaches adhering to the specified PI API shall be trialed.

The initiatives RCA and OCORA encourage the dialogue with further railways and vendors, explicitly also from different sectors, to jointly perform the big technology leap the railways are aiming at.

Abbreviations

Abbreviation	Explanation
API	Application Programming Interface
ATO	Automatic Train Operation
AUTOSAR	AUTomotive Open System Architecture
CCS	Control Command and Signalling
COTS	Commercial-off-the-shelf
CRC	Cyclic Redundancy Check
E2E	End-to-End
ECU	Electronic Control Unit
ETCS	European Train Control System
FRMCS	Future Railway Mobile Communication System
HW	Hardware
OCORA	Open Control Command and Signalling Reference Architecture
OS	Operating System
OSI	Open Systems Interconnect
PI	Platform Independent
RCA	Reference Control Command and Signalling Architecture
ROS	Robotics Operating System
RTOS	Real-Time Operating System
SDN	Software Defined Networking
SIL	Safety and Integrity Level
SMR	State Machine Replication
SPEM	Software Process Engineering Metamodel
SRAC	Safety Related Application Condition
SW	Software
TCO	total cost of ownership
TCP	Transmission Control Protocol
TMS	Traffic Management System
TSN	Time-Sensitive Networking
V&V	Verification & Validation
WDM	Wavelength Division Multiplexing

References

[1] RCA initiative, see https://www.eulynx.eu/index.php/news
[2] OCORA, see https://github.com/OCORA-Public/Publication
[3] EUG (ERTMS Users Group), see https://ertms.be/
[4] EULYNX, see https://www.eulynx.eu
[5] AUTOSAR, see https://www.autosar.org
[6] OCORA-40-001-Beta, see https://github.com/OCORA-Public/Publication
[7] IMA, see https://www.easa.europa.eu/document-library/rulemaking-subjects/integrated-modular-avionics-ima
[8] ARINC 653, see https://www.arinc.com
[9] AUTOSAR Adaptive, see https://assets.vector.com/cms/content/know-how/_technical-articles/AUTOSAR/AUTOSAR_Adaptive_ElektronikAutomotive_201809_PressArticle_EN.pdf