RCA Beta – Chapter on Platform Independence

Table of contents

Beta.1	26.8.2019	B. Rytz	Ready for publication after review core group

1. Introduction

1.1. Purpose of the document

The publication of RCA Alpha has generated interest among railways and suppliers. Direct feedback and feedback from several workshops have provided insight, where clarifications or additional content would be helpful. In the document RCA Architecture Overview, the chapter 6.2. “RCA technical architecture” gives an a very brief overview over the technical architecture of RCA, including a short sketch of platform independence.
This document provides additional information on the topic of **platform independence**, a topic well-known in other industries, but so far rarely applied in the railway domain[1].

This is the very first publication on this topic from RCA and its main purpose is to establish the topic and get discussions with stakeholders in the sector going.

From the RCA FAQ: What is the role of platform independence in the RCA?
In addition to the “logical” decomposition of functions into components, RCA plans to specify a platform (runtime environment and communication stack) on which the logical components can be executed. The logical components can become pure software which will increase the options for “mix-and-match” of components, for virtualization, co-location, even cloud-based models. This is the standard model in IT and, increasingly, automotive. Of course, this also requires a mature concept for “modular safety”.

1.2. Definition of Platform Independence (PI)

Portability in high-level computer programming is the usage of the same software in different environments. The prerequisite for portability is the **generalized abstraction** between the application logic and system interfaces. When software with the same functionality is produced for several computing platforms, portability is the key issue for development cost reduction.[2]

Platform independence is then the fact of using a “generalized abstraction” between application logic and system interfaces.

In other words, we want to provide a **standardized runtime environment**[3] (hardware and system software) allowing to implement CCS functions as software-applications.

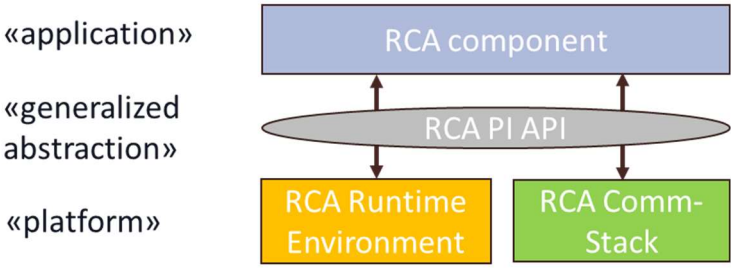
To concretely achieve platform independence, we need the following elements:

- A definition of a “generalized abstraction” (in the form of a specification).
- At least one provider implementation of the “generalized abstraction”, we call this a **platform**.
- One or more applications applying the specification and being able to run on the platform.

Platform independence is achieved when an application, based on the generalized abstraction, runs unchanged on different platform implementations[4]. For this, the application shall only use external functions through a defined API[5] (see RCA PI API below).

[1] Some suppliers of signalling equipment have the concept of COTS/platform independence - but the lack of a mature modular safety concept and framework has not allowed to achieve cross-vendor independence.
[2] Software portability, see https://en.wikipedia.org/wiki/Software_portability
[3] Runtime environment, see https://en.wikipedia.org/wiki/Runtime_system
[4] How stringent this requirement needs to be (i.e. binary portability vs source code portability) is open.
[5] API = Application Programming Interface

The following diagram shows the concepts applied to RCA (using the terms of the “Architecture overview”):



In this diagram:

- RCA components are (software-) **applications** running on the platform.
- The RCA PI API is the **generalized abstraction** defined for running RCA components.
- The RCA Runtime environment (RTE) and the RCA Comm-Stack make up the **platform**[6].

1.3. Purpose of Platform Independence (PI) in the context of RCA

Platform independence is an application of the basic RCA principle of modularity (→ exchangeability).
While the RCA interface architecture focuses on (logical) components and their interfaces, platform independence takes an orthogonal perspective[7] and makes a proposal for the internal structure of each (logical) component.
Since PI is orthogonal to the logical components, it is possible to implement an RCA-based system without applying PI. This is important since PI is new for the railway sector and it is yet unclear, when solutions for PI will be available on the market and it is expected that the inclusion of PI will encourage new vendors to build the solution for railway from other industries and also existing vendors to bring better solutions based on the PI paradigm.

So, the added benefit of including PI in RCA is: PI is a proven concept that provides various technical benefits and having it covered under RCA shall add strategic, technical and economic benefits described in chapter 2 “Goals of Platform Independence (PI)”.

1.4. Important characteristics of Platform Independence

Typical features of the generalized abstraction include:

- Executing code (computing resources)
- Accessing memory (volatile) and storage (persistent)
- Access parallelization and synchronization mechanisms
- Instance lifecycle: startup, supervise, restart, stop, SW-update, recovery
- Communication with other systems / devices
- Security mechanisms
- Safety mechanisms (see ch. 4.4 for an example)
- Operations support functions such as logging, monitoring etc.

This means the platform includes features typical for Operating System (OS) and features found in “middleware”. Additionally, features known from data-centre / cloud operations will be increasingly important:

- Performance: Compute load balancing, Platform Scalability (if more compute power (processors, RAM etc) is required).
- Dynamic shifting/transfer of compute load (e.g. when there is a breakdown).

---

[6] It is an open design question if RTE and the Comm-Stack should be logically separated or not.
[7] Orthogonal in the sense of independent i.e. platform independence contributes to modularity (separation between functional and technical aspects) and the RCA interface architecture contributes to modularity (separation between functions).

- Hosting of applications from different vendors on the PI platform.
- Synergy among different PI platforms (abstraction layer): two PI platforms from two vendors can share information for orchestration purpose.

Two important aspects of platform independence in RCA are the requirements:

- that there must not be a commercial lock-in (e.g. a unique commercial supplier) for the platform i.e. the platform would need to be “open” or many suppliers provide a platform on the market.
- that it must be possible for independent suppliers to efficiently develop applications based on the generalized abstraction. This includes topics such as tool-chain, licensing etc.

1.5. Examples in other domains

In IT varying degrees of platform independence have existed for many years:

- Source code portability of the C language including associated libraries (standardized by ANSI 1989).
- Standardized core of the Unix OS family (standardized in IEEE POSIX 2001, applied by commercial and open source OS).
- The GNU/Linux family of OS in a de facto standard (applying an open source model).
- Various popular hypervisor softwares such as VmWare, Hyper-V are in use for many years in the IT industry. Initial search shows that there are safety certifiable PI platforms (OS), which are also available commercially-off-the shelf.
- Container based virtualization uses operating-system-level virtualization to create PI.
- Java also provides platform independence by executing the same bytecode on any platform using JVM (java virtual machine).
- JEE provides a large platform “ecosystem” with many middleware-type services (security, communication, persistence etc.).

Examples from other domains include:

- AUTOSAR Adaptive [3], [4] includes an OS, but also middleware-type services (security, communication, persistence etc.).
- Avionics industry is using ARINC 653[8] Partitioning Hypervisor to execute safety critical applications in a PI environment. This standard has been used in railways. See also [5] for a study on application of ARINC 653 for railways (vehicles).

The above example shows that the PI concept is widespread available in software systems, adopted by different industries and needs more study, evaluation for railway specific usage.

1.6. Additional material

The following documents from the context of smartrail 4.0 provide a deeper analysis of the topic. Since they were written before or during the RCA work, they are not yet completely aligned with RCA terminology:

- [1] General Concept ETCS Interlocking Datacenter http://smartrail40.ch/service/download.asp?mem=0&path=\download\downloads\General_Concept_ETCS_Interlocking_Datacenter%20(2).pdf.
- [2] Safety-critical Applications in Data Center in the Railway System SBB http://smartrail40.ch/service/download.asp?mem=0&path=\download\downloads\Safety-critical%20Applications%20in%20Data%20Center%20in%20the%20Railway%20System%20SBB.pdf.
- [3] AUTOSAR Adaptive, https://www.autosar.org/standards/adaptive-platform/.
- [4] AUTOSAR Adaptive, the computing center in the car, https://assets.vector.com/cms/content/know-how/_technical-articles/AUTOSAR/AUTOSAR_Adaptive_ElektronikAutomotive_201809_PressArticle_EN.pdf.
- [5] Safe4Rail. Report on analysis of ‘functional distribution.

[8] ARINC, siehe https://en.wikipedia.org/wiki/ARINC_653

- Architecture frameworks and solutions. https://safe4rail-1.safe4rail.eu/downloads/deliverables/Safe4RAIL-D2.2-Analysis-functional-distribution-architecture-solutions-PU-M7.pdf.

2. Goals of Platform Independence (PI)

Platform independence pursues goals on the strategic, technical and economic level.

Strategic/Economic:

- Supporting the RCA framework: RCA provides an architecture framework based on harmonized requirements and introduction of PI in RCA will ease the implementation.
- Reduction of vendor lock in (for the computing platform and for the applications): the same application can be used on platforms of different suppliers.
- Reduce entrance barriers for new suppliers.
- Cost reduction, Lower life cycle cost.

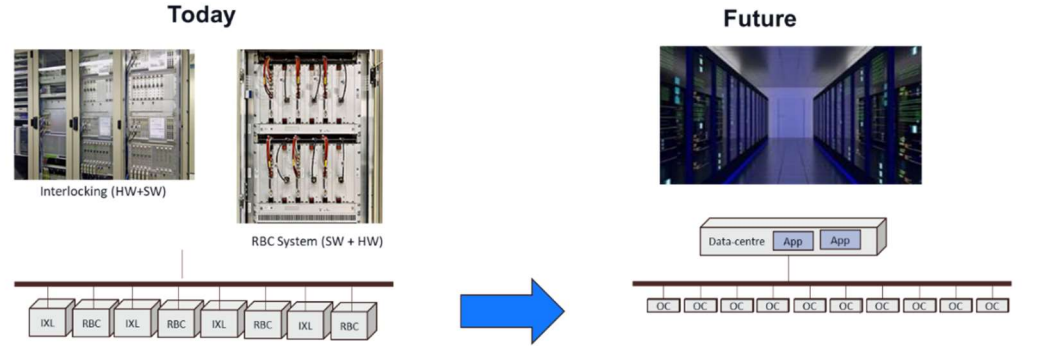
Technical:

- Separating the application from the underlying middleware and OS (operating system) increases upgradability for the application. An RCA component turns into a “piece of software”.
- Applications are also separated from the underlying hardware, easing life-cycle management. Hardware evolves independently of software; spare parts may only be available for a relatively short time.
- Allows co-locating components on the same physical node. This allows a clean separation of logical components without multiplying the number of physical nodes. This also facilitates centralizing components in (a few) data-centres.
- Opens the path to virtualization and cloud-based hosting of the RCA components.
- Opportunity to introduce COTS (commercial off-the-shelf) hardware in the platform.
- Independent evolution of applications and IT infrastructure, future-proofness. Simplifies lifecycle and obsolescence management.
- IT responsiveness and flexibility.
- Ease of maintenance: faulty hardware can be swapped easily without causing system downtime.

Issues and risks, which have to be addressed are: modular safety, attractiveness of the business model for suppliers, time needed to establish such a model in the railway domain, roles of suppliers and integrators.

2.1. Platform independence and centralization

The following picture shows one important aspect of PI, the contribution to the possibility to centralize applications physically[9].



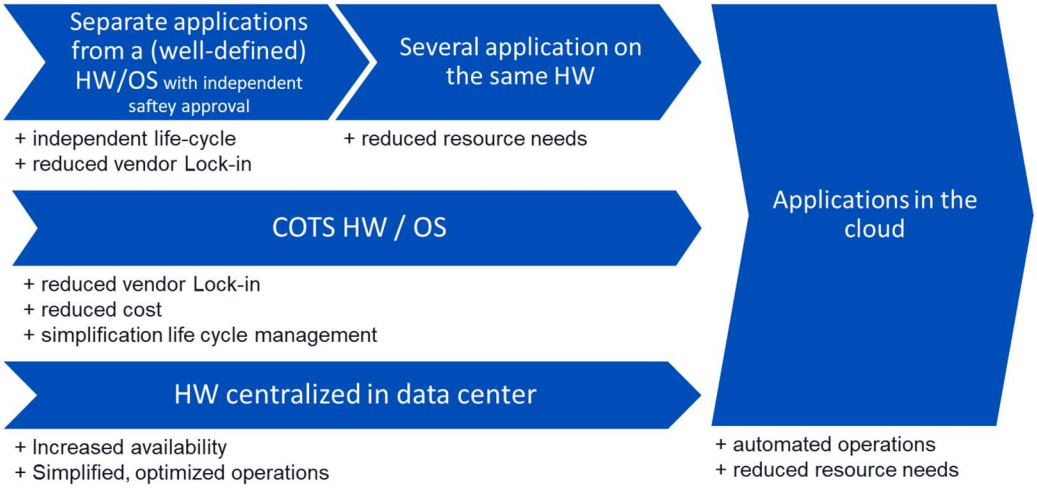
Interlockings / RBC:

- procured as «systems» (HW + SW)
- safety-critical (SIL4, multi-channel)
- deployed locally

Data-center + applications + OCs:

- HW centralized in data processing center
- safety critical applications on COTS HW
- decentralized object-controllers (OC)

The following diagram shows the expected effects:



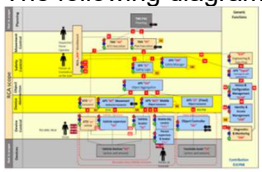
[9] Note: even with PI, decentralized models are still possible and viable. This is ultimately an issue of overall system-architecture.

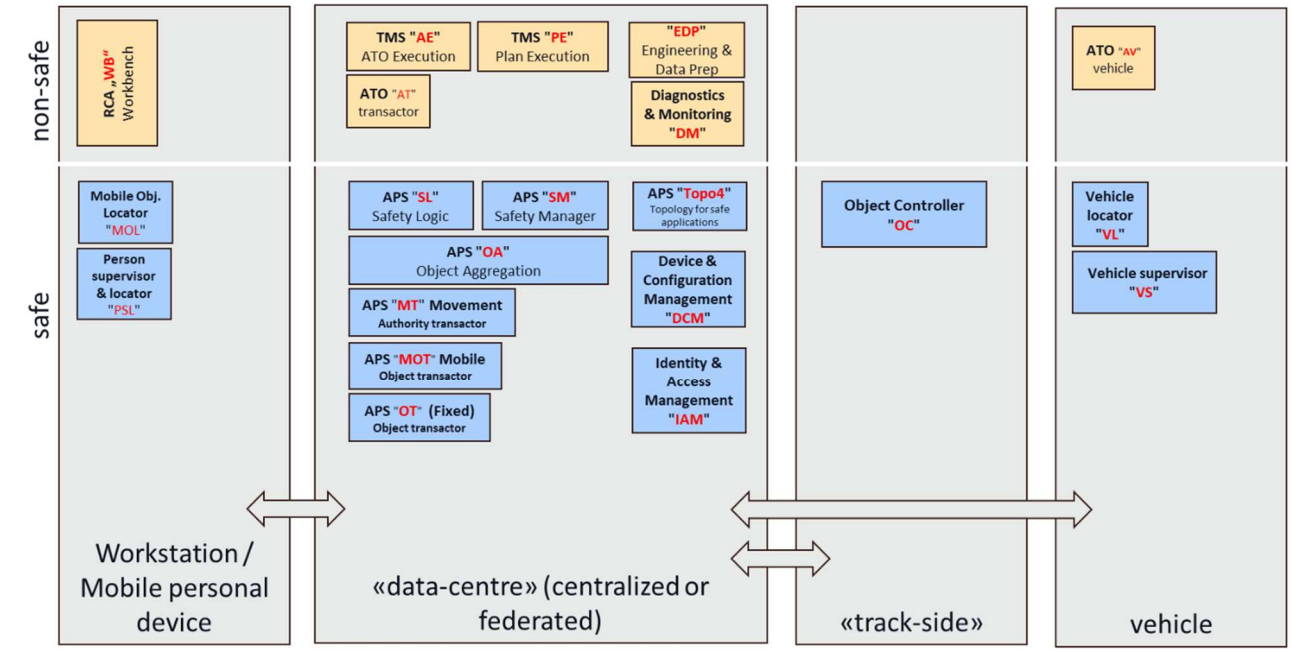
3. Where is platform independence applicable in RCA?

Platform independence is in principle applicable to all RCA components. Not all RCA components have, however, the same context and requirements with respect to the platform they will need. Differences include:

- 4 (or more) different “deployment zones” i.e. where the component is likely to be localized (data centre, work-place, track-side, vehicle). This has an influence on: available physical protection, available computing resources, system management possibilities, available communication channels.
- Different safety (and other non-functional) requirements: Safety critical, Safety relevant, no Safety relevance.
- Different resource / scalability requirements
- Different instance cardinalities (e.g. 1 (redundant) instance of SL to 1000s of instances for OC or VS.

The following diagram shows 4 identified “deployment zones” and provides a short description.





In principle the same “general abstraction” and the same platform (but differently sized) could be used for all components. There are some issues however, which have to be investigated:

- Using a platform with safety-critical capabilities for a component without safety relevance is likely wasteful for the development and the operation of the application[10]. Therefore more than one platform type (according to SIL requirements) will be necessary.
- “track-side”: The object-controllers (with interface to “the real world”) are likely to be physically more distributed than most other RCA components and probably don’t benefit from a PI. I.e. OC are likely to be procured as “systems”, including SW and HW.
- “vehicle”: potentially there are synergies with the vehicle platform. Needs coordination with OCORA, where PI for the vehicle is a main goal.

[10] The use of safety platforms for non-safety applications increases the life cycle cost and unnecessarily limits the possibility of developing and deploying changes. In an RCA context these are strong arguments for separation.

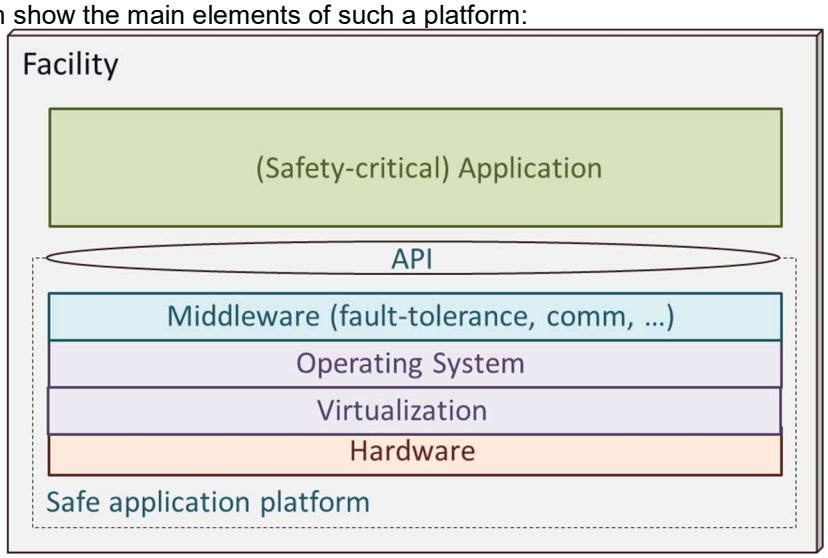
Notes:

- The diagram does not show a fixed partitioning of the components onto different platforms, but shows important criteria for the partitioning.
- We will probably have two different platform definitions for safe and non-safe applications. Since usable abstractions for non-safe applications already exist, this is considered the "easier" problem and our focus in on a platform definition for safe applications.
- Therefore, our main work on PI for RCA targets the safe applications in the data-centre, with a potential synergy on the vehicle (→ OCORA).

4. Topics under investigation (for safe applications)

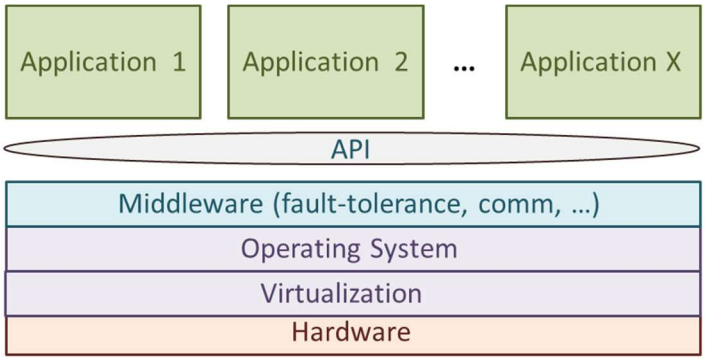
4.1. Scope and structure of a platform for safe applications

The following diagram show the main elements of such a platform:



- **Facility**: Building infrastructure that provides an optimal environment to the IT and telecom equipment by minimizing negative external influences such as power outages, temperature changes and physical security.
- **API** (Application Programming Interface). Abstraction layer to insulate the applications from the operating system, the virtualization layer and finally from the hardware. Provides the mechanisms to run the safety critical, functional applications in a fault-tolerant manner.
- **Middleware** implements technical functions commonly used by the applications such as mechanisms for fault-tolerance, communications, logging, etc.
- **Operating system (OS)**
- **Virtualization**: Hardware/Software Abstraction Layer to abstract hardware from the rest of the platform.
- **Hardware**: Hardware (including firmware) for data processing and storage, plus network infrastructure including IT security measures.
- **Safe application platform:** the combination of hardware, Virtualization, OS, Middleware proving the API

The following diagram shows that such a platform can support several applications (potentially from different suppliers). Such a platform also supports: independent lifecycles between applications and platforms and among applications.



4.2. Requirements for such a platform

For a collection of requirements refer to [1], chapter 5.

4.3. Design concepts proposals

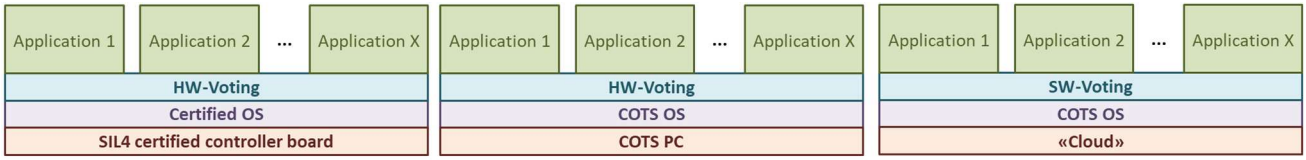
A market research conducted by ESG Elektroniksystem und Logistik GmbH [2] confirms the lack of Commercial off-the-shelf (COTS) servers or high-performance computers with SIL4 capabilities. Yet SIL4 pre-certified hardware is available as embedded systems. In the railway field those embedded computers are mainly used for onboard applications such as an ETCS - On Board Unit. Components that are pre-certified to EN 50129 are therefore designed for the operation in harsh environments. These supplementary requirements, yielding the operation in harsh conditions are not needed for the targeted application in the data-centre and imply therefore only an unjustifiably cost factor.
Moreover, the embedded solutions, provided by different vendors are all based on dated CPU architectures and must often be operated in restricted conditions to fulfil the SIL4 demands. Typically, the boards are operated in lockstep mode, where the clocks of two CPU are synchronized and the safety function is processed in parallel and validated with a 2oo2 voting. Thus, the potential of modern-day multicore CPUs cannot be exploited in such a setup. That performance is highly desired to design a performant Safe Data Centre Application Platform. To overcome this dilemma, new innovative architectures for the Safe Data Centre Application Platform are proposed in this section. Some of them are based on disruptive principles and can therefore most likely not be certified by strictly following the routes denoted in the CENELEC standards. Thus, the proposed designs must be evaluated against their ability to being certified. This has to be done in close collaboration with the certification authorities.

In [2] several approaches are described, including:

1. Pre-certified safety. Using hardware specially designed and pre-certified to CENELEC EN 50129.
2. Hardware-centred safety mechanisms. Multi-Channel MooN (M out of N) based on diverse COTS components.
3. Mixed safety mechanisms. Applying arithmetic codes to detect hardware malfunctions.
4. Software-centred safety mechanisms. Use replicas with cross-voting.

We currently focus on approaches 3 and 4 i.e. without need of specific SIL-4 hardware, applying techniques at the software-level to achieve the necessary safety levels.

Different concepts will have to be evaluated. If a stable API can be defined, the concept even allows different implementations of a safe platform and / or the evolution of such a platform. The following diagram show such a scenario.

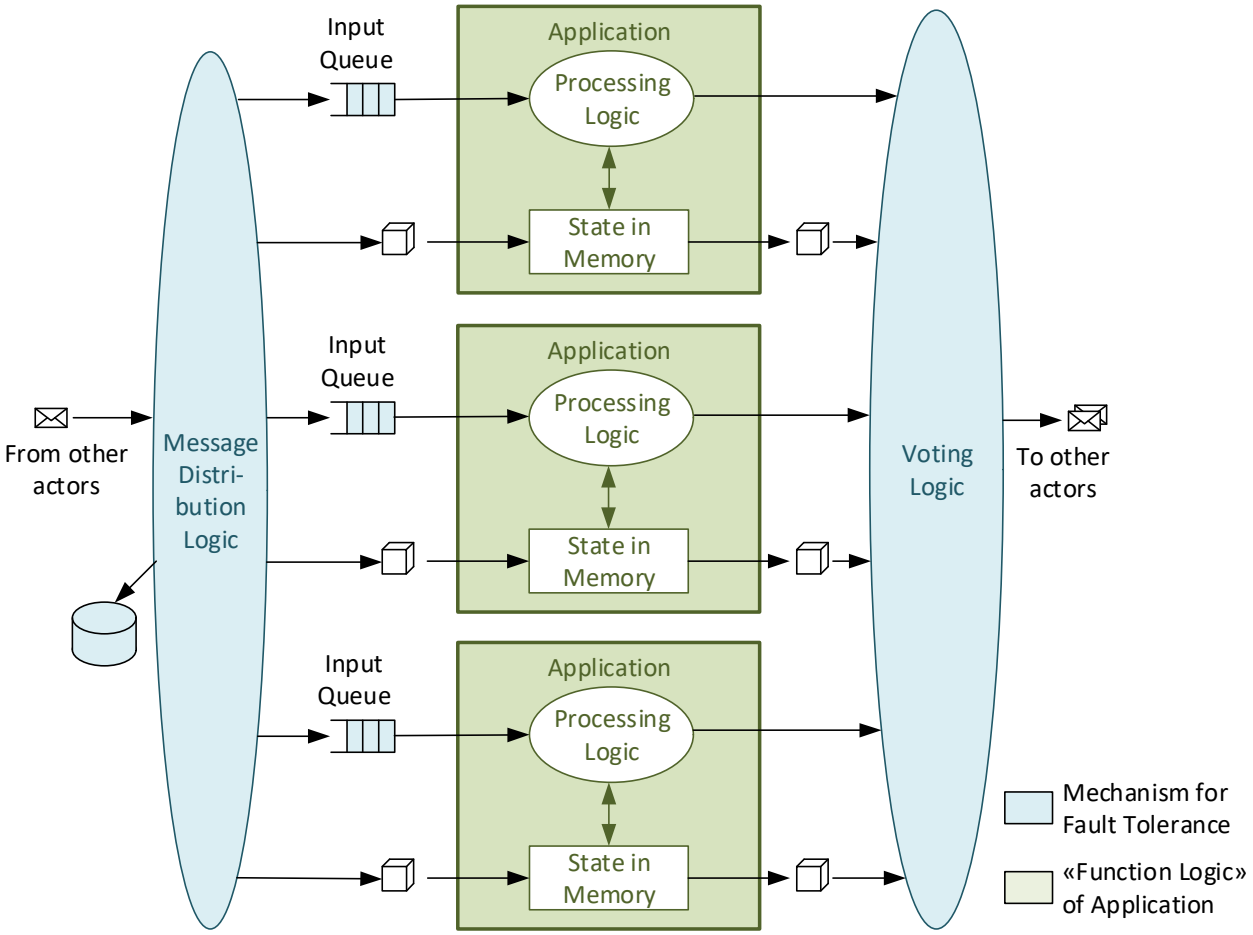


4.4. For illustration: safety mechanisms in the platform provided by software

The following diagram shows an architecture (based on state machine replication) under investigation with the following characteristics:

- Applications can run in different deployments.
- "Message Distribution Logic" ensures an identical sequence of messages "Atomic Broadcast".
- A deterministic application in the sense that:
  - a given "state in memory" and
  - a given received input message

  always results in exactly:
  - the same change of the "state in memory" and
  - the same sequence of sent output messages.
- "Voting Logic" compares output messages.
- Restart a replica:
  - Export snapshot of other replicas (special control message).
  - Voting on Snapshot.
  - Import snapshot in newly started replica.
  - Replay of all messages since "Export" message.



4.5. For reference: Safety mechanisms recommended in EN50129:2016

Table E.2 – Architecture of system, subsystem or equipment

Techniques/Measures	SIL 1	SIL 2	SIL 3	SIL 4	
1 Independence of safety-related function from non-safety-related function	HR	HR	HR	HR	(a)
2 single electronic structure with self-tests and supervision	R	R	NR	NR	(b)
3 single electronic structure based on inherent fail-safety	R	R	HR	HR	(c)
4 single electronic structure based on reactive fail-safety	R	R	HR	HR	(d)
5 Dual electronic structure	R	R	NR	NR	(e)
6 Dual electronic structure based on composite fail-safety with fail-safe comparison	R	R	HR	HR	(f)
7 Diverse electronic structure with fail-safe comparison	R	R	HR	HR	(g)
For a given SIL all techniques of the grey shaded group are alternatives, i.e. at least one of these techniques should be chosen. For a given safety related function, several techniques may be combined so as to address all kinds of faults affecting the function. Technique 5, 6 and 7 includes also architectures where additional electronic structures are used for availability purposes (e.g. 2oo3).					

Techniques for SIL4:

3: Inherent Fail-Safety

- Allows a safety-related function to be performed by a single channel.
- **All** credible failure modes of the item are **non-hazardous**.

4: Reactive Fail-Safety

- Allows a safety-related function to be performed by a single channel.
- Safe operation is assured **by rapid detection and negation** of any hazardous fault.
- Checking/testing/detection function can be regarded as a **second channel**.

6: Composite Fail-Safety

- Each safety-related function is performed by **at least two channels**.
- A fault in one channel shall be **detected and negated** in sufficient time.

Therefore, the platform for safe applications has to provide mechanisms of type 6.

5. Summary and outlook

Achieving platform independence, together with the capability to centralize / virtualize applications from SIL1-SIL4, promises important benefits. Other sectors have already made this step.

While the concept seems new for the railway domain, it has already been applied internally by (at least some) vendors of safety-critical railway equipment, because it makes sense from a re-use and lifecycle-management point of view.

Achieving platform independence for all players in the railway sector will require substantial efforts on several questions:

- Technology: what architecture is mature and future-proof?
- Business: what is the business model for a supplier bringing their (internal) platform to market?
- Regulations: how to structure safety cases in a modular fashion and learn as a sector (suppliers, IM, regulator) to efficiently and safely apply the new concepts?
- Business and regulations: the responsibility of integration needs to be redefined (SW has other legal implications than a device).

The RCA group judges that “platform independence” is a very important enabler for the RCA goals and it will be pursued in parallel to the specification of the RCA interface architecture.

Next steps:

- This topic is a candidate for a S2R project.
- Starting by the end of 2019 the program smartrail 4.0 (SBB) will start working together with suppliers on “platform independence”.
    - Start with most difficult aspect i.e. platform independence for safety critical applications. Current focus: data-centre.
    - Define independent platform interface (with supplier input).
    - Perform trial with 2 or more suppliers.
    - Evaluate feasibility (technically and commercially) of “open” interface for 3rd party app developers and 2 or more suppliers.
    - Investigate feasibility of using platform for other “zones” than the data-centre.