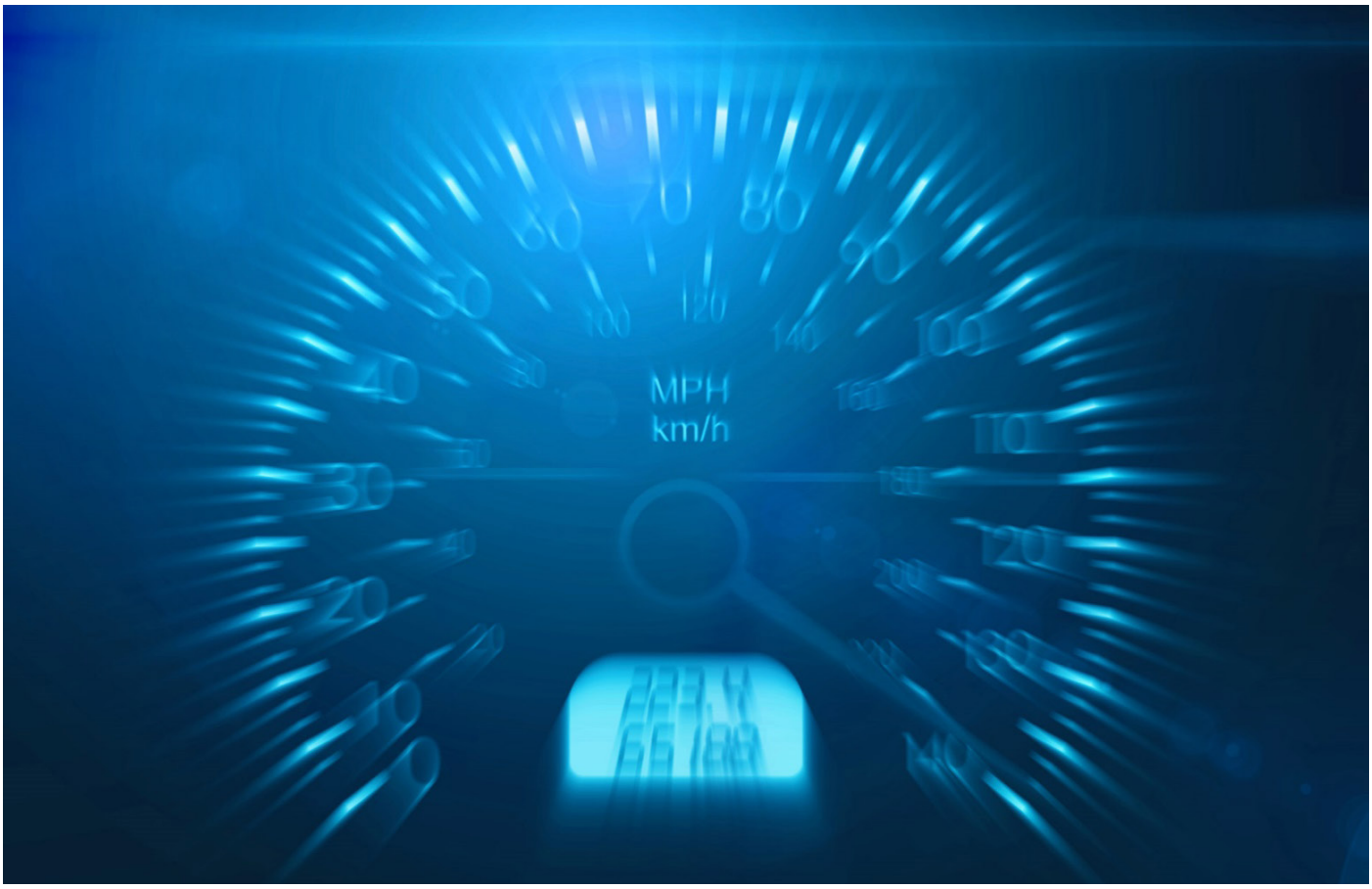
Elektrobit

Tech paper

Streamlining functional safety on your automotive software platform with EB corbos Hypervisor

Summary of Tech paper contents



In today's rapidly evolving landscape of automotive software platforms, functional safety managers and software platform architects face a daunting task. They must design mixed-critical platforms that seamlessly accommodate both quality managed and safety-related functions while adhering to rigorous safety standards. Mixed-critical platforms introduce a multitude of challenges, requiring stakeholders to make critical decisions that impact resource usage, safety fulfillment, and overall platform efficiency. Balancing these competing goals is a complex endeavor that demands innovative solutions.

This tech paper highlights the impact of these challenges on the overall safety and quality of the software platform and proposes a powerful solution—utilizing a bare-metal hypervisor—to streamline functional safety processes. A hypervisor virtualizes hardware resources, allowing software stacks tailored for their applications to run concurrently on a single hardware platform.

On a technical level, a bare-metal hypervisor isolates safety-critical software from non-safety-related software. This minimizes interference from non-safety-related software. On a process level, a bare-metal hypervisor minimizes the safety-related scope of a software platform. This enables functional safety managers and software platform architects to streamline functional safety processes by avoiding unnecessary development and verification efforts for non-safety-related software as well simplifying the development of safety-related software. In total, a bare-metal hypervisor enables all stakeholders to effectively and efficiently achieve functional safety.

Effectively achieving functional safety in the automotive domain means designing and developing a vehicle's systems to sufficiently minimize the risk of malfunctions that could potentially lead to accidents, injuries, or fatalities. Effectively achieving functional safety goes beyond simply checking boxes or meeting regulatory requirements.

For a software platform, it means understanding the risks on the software platform for all phases of the development cycle and implementing measures to sufficiently avoid and mitigate these risks.

This process impacts the underlying software infrastructure upon which vehicle systems and their applications are developed, integrated, and deployed.

Efficiently achieving functional safety in the automotive domain means implementing safety measures and processes in a cost-effective, resource-conscious, and time-sensitive manner without compromising safety objectives. This involves optimizing safety-related activities throughout the vehicle development lifecycle to reduce development overhead. You cannot successfully achieve functional safety when the resulting process is so time intensive that software and safety engineers cannot perform the process in the available time or budget. Even worse, a process with too much overhead may lead to corner cutting which will negatively impact the effectiveness of the relevant safety processes.

Achieving functional safety effectively and efficiently on a consolidated software platform can be a complex endeavor. The integration of various systems and components introduces a multitude of potential points of failure, making it challenging to anticipate and mitigate faults that may arise unexpectedly. Additional challenges such as rapidly evolving technologies, changing requirements on the platform, and competing goals between safety, performance, and costs complicate the process.

In this tech paper we will examine a **proven strategy** that ensures that your software platform is fit for functional safety.

1. Reducing your safety footprint is the key to efficiently and effectively achieving functional safety

To achieve functional safety efficiently and effectively, the goal should always be to reduce the amount of code with a safety allocation running on a platform. This means only assigning a safety-level to a component when absolutely necessary. This reduces the need to perform painstaking development and integration of components that are not inherently safety-related. A poorly designed system may introduce unnecessary safety processes that are time-consuming, complex, and costly which can lead to an overall degradation of safety.

2. Not all elements of a safety-related platform must be safety elements

The ISO 26262 standard for functional safety in automotive systems does not require systems to develop every software platform element as a safety element. It is important to understand that the standard provides a risk based approach for determining what safety engineers must consider as a safety-critical component.

The ISO 26262 standard provides a process for risk assessment and classification using Automotive Safety Integrity Levels (ASILs), ranging from ASIL-A (lowest) to ASIL-D (highest). The safety-level assigned depends on the severity, exposure, and controllability of potential hazards of vehicle systems. Systems with higher ASIL level are subject to more stringent design, verification, and validation processes. Software components inherit the highest ASIL level of the system functions to which they contribute.

So, not all software components in a vehicle will have the same ASIL level. Some components might not even be assigned a safety-level at all. For example, a software component in an infotainment system would be considered less safety-critical than a component in a vehicle's steering system.

By focusing safety development efforts on those software components that are truly safety-related, we can avoid unnecessary work and at the same time ensure that the platform architect prioritizes the quality needs of non-safety-related software.

Now that we understand that safety engineers must not develop all software components as safety-related components, the next section will describe which strategies you can take to reduce the safety footprint of your software platform.

3. Reducing your safety footprint means minimizing safety-related elements and isolating them from other elements of the system

The process of reducing the safety footprint involves four steps. Here is a breakdown of each step:

1. **Identify safety-related components:**
   Start by identifying the components of the system that are safety-related. Platform architects can achieve this through requirements tracing, which involves tracking and documenting how requirements flow from the vehicle level down to the unit level. By doing so, you can identify all components that contribute to a safety function.

2. **Identify dependencies:**
   Once you have identified the safety-related components, it is essential to identify their dependencies to other software platform artifacts. This requires having a comprehensive documentation of the software architecture. Platform architects can also use software tools to analyze interfaces and dependencies to other software artifacts in order to determine how components interact and depend on each other.

3. **Isolate the safety stacks:**
   The next step is to isolate the safety-related components from the rest of the software platform. This isolation must use mechanisms specifically developed as safety elements. By isolating the safety-related components, you minimize the potential impact of failures or errors in other parts of the software platform.

4. **Verify freedom from interference:**
   Once the platform architect has implemented isolation mechanisms, it is necessary to verify that a malfunction or failure in a quality managed component will not impact the safety of the platform and its components. It is crucial that platform architects take a systematic approach when identifying potential sources of interference. This includes considering communication, execution, and memory aspects of the safety-related components.

In the diagram below, we can see an example of a consolidated software platform where multiple system functions are allocated. These system functions may be safety-related functions such as software update over the air (SOTA), automated vehicle path control or collision avoidance systems. Non-safety-related functions of the infotainment and IoT domains may also be allocated to this software platform:

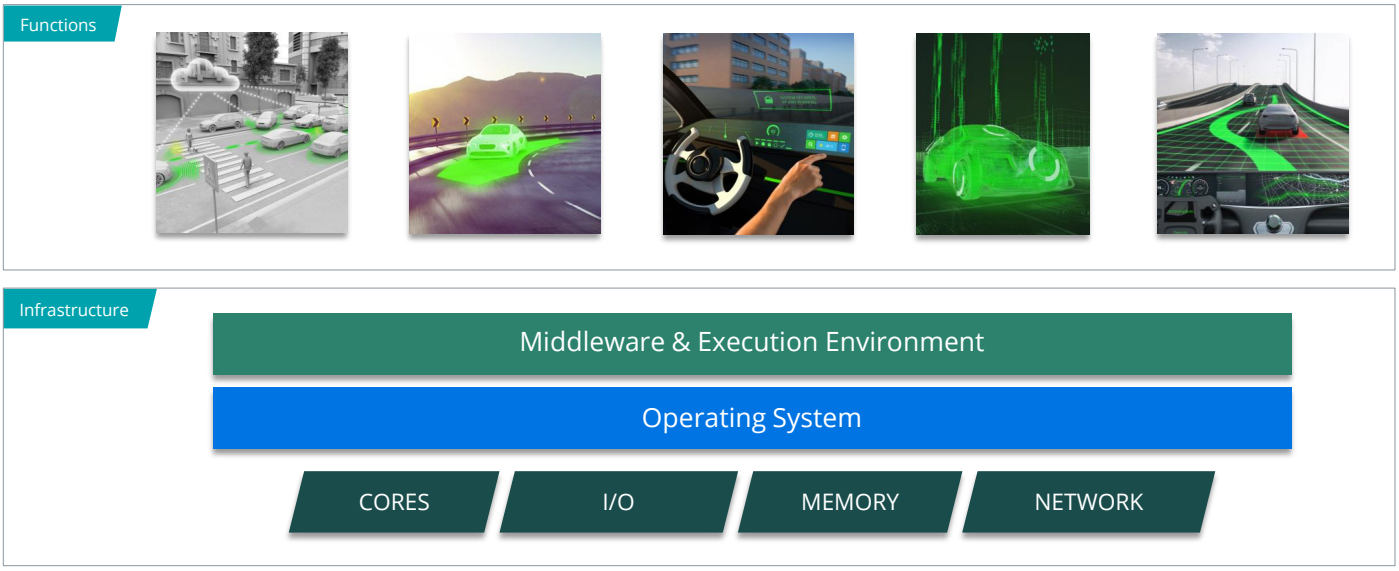


Figure 1 Monolithic Software Platform

After identifying the safety related components and identifying their dependencies to elements of the middleware, execution environment, operating system and hardware, platform architects can establish the software stacks that the hypervisor isolates:

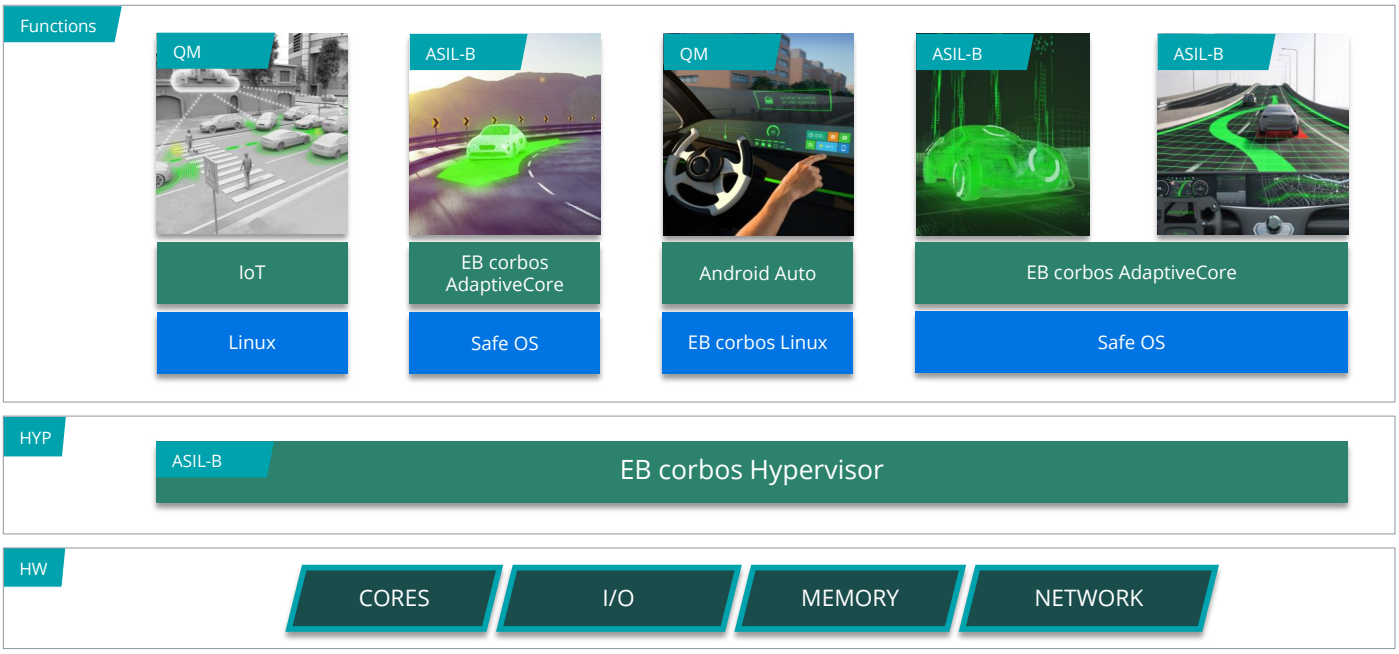


Figure 2 Minimized Safety Footprint

4. Minimizing your safety footprint can save development time compared to a monolithic software platform

Compared to a monolithic platform where QM and safety-related components run in the same execution context, a platform with a minimal safety footprint can provide better development efficiency, cost-effectiveness, and development agility. By focusing on safety components and isolating them from non-safety-related components, it is possible to strike a balance between safety, process performance, and development agility:

- **Safety:**
  In a monolithic platform where all components run in the same context, there can be numerous potential sources of interference at the interface, execution environment, and operating system levels. Isolating safety-related stacks helps minimize the potential sources for interference, as failures or errors in non-safety-related components are less likely to affect the isolated safety components.

- **Process performance:**
  Assigning a safety-level to all platform elements can be expensive and may not even be feasible. By focusing safety efforts only on the necessary safety-related components, safety engineering resources can be allocated more efficiently. This reduces the cost of developing and maintaining the system while still ensuring that the software platform fulfills safety requirements allocated to it.

- **Development agility:**
  Isolating safety-related components allows for more flexibility and agility during the development process. It enables independent development, testing, and verification of quality managed components. This can help streamline the development timeline and accelerate time-to-market for the software platform.

Figure 3 below shows how development and deployment of quality managed functions can be decoupled from safety-related functions. Development and verification of safety-related functions can be isolated to a single software stack.

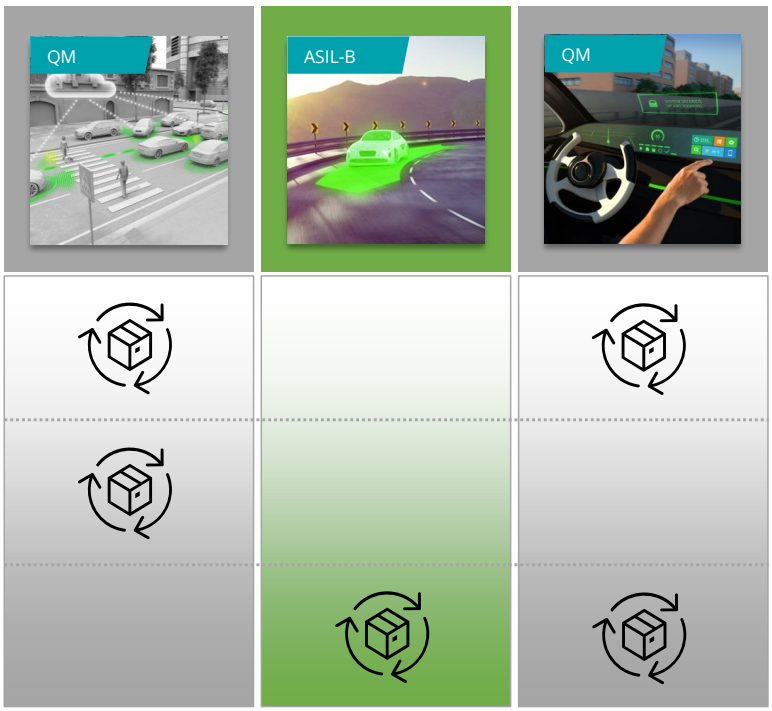


Figure 3 Separating the development cycles of vehicle functions

5. A hypervisor provides a safe and secure foundation for safety-related software stacks

A type-1 hypervisor, also known as a bare-metal hypervisor, provides a deep separation of software stacks from the hardware up to the operating system and its execution environment. A bare-metal hypervisor runs directly on the hardware, allowing multiple software stacks to run as isolated virtual machines (VMs) on top of it.

There are several features and characteristics of a bare-metal hypervisor that are interesting for a safety-related software platform:

- **Isolation:**
  A primary function of a bare-metal hypervisor is to enforce strong isolation between different VMs. A bare-metal hypervisor isolates a failure in an operating system in one VM from operating systems and applications running in other VMs.

- **Resource management:**
  A bare-metal hypervisor allocates and manages hardware resources, such as CPU, memory, storage, and network interfaces, among the VMs. It provides mechanisms for efficiently sharing as well as isolating these resources to ensure fair and secure utilization. This allows the platform architect to efficiently manage the allocation of platform resources to software stacks.

- **Security:**
  The deep separation provided by a bare-metal hypervisor contributes to enhanced security. Since a bare-metal hypervisor isolates each VM, any compromise or vulnerability in one VM does not directly impact the security of other VMs or the underlying hardware.

- **Performance:**
  By running directly on the hardware, a type-1, bare-metal hypervisor typically offers better performance compared to type-2 hypervisors, which run on top of a host operating system. Direct access to hardware resources allows for efficient and low-latency communication between the VMs and the underlying hardware.

- **Flexibility:**
  Bare-metal hypervisors can support a variety of operating systems and software stacks as guest VMs. This flexibility enables running different operating systems that may better support safety-related or quality managed applications.

Overall, a bare-metal hypervisor provides a robust and secure virtualization platform by separating software stacks from direct hardware access and creating isolated execution contexts for different VMs. This architecture offers flexibility, performance, and enhanced security for various virtualization scenarios.

6. The EB corbos Hypervisor provides a foundation to effectively and efficiently create a safe and secure software platform

The **EB corbos Hypervisor** is a powerful tool for your software platform. It provides a comprehensive solution for managing hardware resources and ensuring a safe execution environment for safety-related software stacks:

- **Type-1 hypervisor:**
  The EB corbos Hypervisor is a bare-metal hypervisor that supports both native applications and virtualized operating systems such as Linux. It manages the allocation of hardware resources to native applications or virtual machines (VMs) as per system requirements.

- **ASIL-B Safety Element out of Context (SEooC):**
  The EB corbos Hypervisor is developed as an ASIL-B Safety Element out of Context, which means it fulfills the safety requirements needed to support the isolation of safety stacks. It provides a safety manual that assists platform architects in designing a safe software platform.

- **Microkernel architecture:**
  The EB corbos Hypervisor adopts a microkernel architecture, where the amount of code running in a privileged mode is minimal. Hypervisor services run in their own execution context, with some services being safety-related and others being quality managed. The microkernel architecture approach is one example of reducing the safety footprint of a software element.

  The hypervisor, built on the open-source L4Re Operating System Framework, serves dual functions: It provides virtual machines and acts as an operating system that runs native applications (called MicroApps). Both functions make use of resource access controls using state-of-the-art object capabilities.

- **Built on open-source technology:**
  The EB corbos Hypervisor runs on the L4Re operating system framework developed by our partner Kernkonzept. The L4Re is used extensively outside of the automotive domain, and has a strong focus on safety and security.

- **Support for safety-related and quality managed OSs:**
  The EB corbos Hypervisor can run both safety-related operating systems (OSs) and native applications, as well as quality managed OSs. This flexibility allows for comprehensive coverage of all requirements on a software platform, accommodating safety-critical components alongside other non-safety-critical software.

In the diagram below, we can see the distinct advantage of a microkernel architecture provided by the L4Re. Only those services run in the L4Re kernel that are absolutely necessary. Compared to Linux, L4Re dramatically reduces the size of its kernel. In total, ten thousand lines of code run in a privileged processor mode, in contrast to over one million lines of code for Linux.

This not only reduces the attack surface of the L4Re kernel, but also increases the robustness of the hypervisor; L4Re isolates failures in applications and services such as I/O, memory, and device drivers in userland processes that run in a de-privileged processor mode:

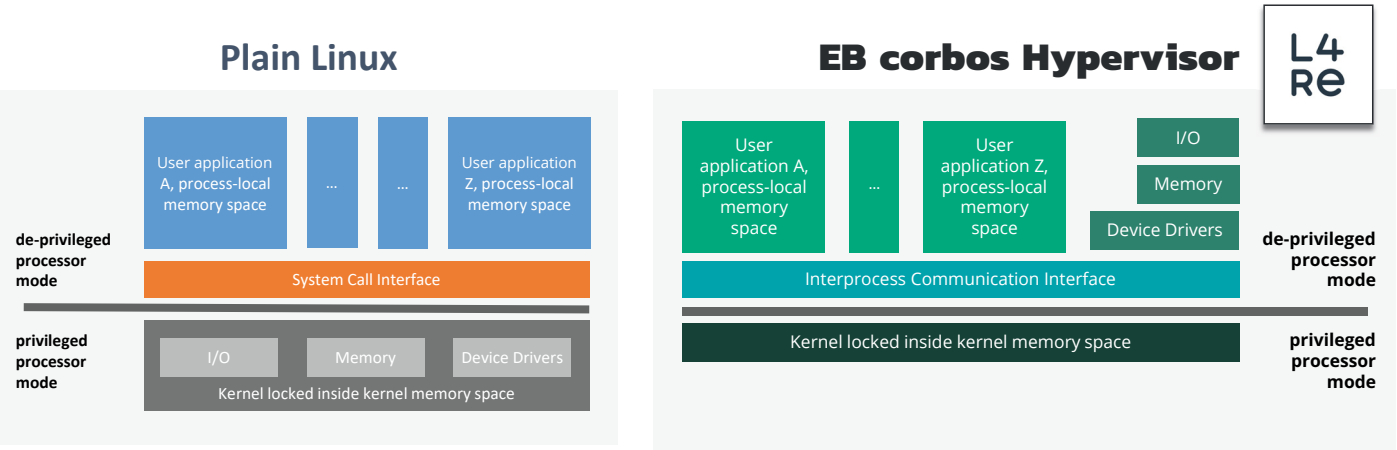


Figure 4 Microkernel architecture of the L4Re

The L4Re Microkernel follows the principle-of-least-privileges. Its applications are implemented through a state-of-the-art capability system and access control. Accessing any service or resource in the system requires a dedicated capability that needs to be granted to the application. Initially, an application starts with a well-defined and minimal set of capabilities that have granted resources to get started. Any other capabilities need to be actively granted to the application, either through the applications' startup configuration or during runtime. This way an application only has the access rights it needs to fulfil its duties, and possible malfunctions or exploits are limited to using only this minimal set of options. An example for a resource accessed via a capability is a communication channel through which two applications can exchange messages.

A further benefit of the microkernel-based L4Re Operating System Framework is the small trusted computing base (TCB) of the application.

A TCB is the set of code a function depends upon to work correctly. This is the application code as well as all the libraries it uses, as well as an any other service in the system that the application uses. Due to the modular, microkernel-based structure of the L4Re technology, system architects enjoy freedom of choice on the services they need for the application.

A prime example are device drivers that are often plagued with bugs. On a microkernel-based system device drivers run in their own modular native L4ReMicroApps. Consequently, a malfunction caused by a bug in a driver will not affect the other applications that do not use this particular driver.

This driver is also not part of the application-specific TCB of the application.

Small TCBs are beneficial for both security and safety considerations as the amount of code that needs to be inspected is considerably smaller than that of a monolithic design like Linux. Modular microkernel-based software architectures practically enable safety and security certifications and are therefore a cornerstone for any safety and security relevant use-case that brings multiple, distrusting application onto a single system.

With the L4Re Microkernel/Hypervisor system architects can set up a minimal TCB and also application specific TCBs for a system, enjoying maximum flexibility, getting maximum safety, and minimizing costs and effort for certification.

7. What can the EB corbos Hypervisor do for you?

The EB corbos Hypervisor supports the development of safety-related software platforms in several significant ways:

- **Basis for your safety architecture:**
  The EB corbos Hypervisor isolates safety stacks from interference caused by quality managed components. This reduces the efforts required to identify potential sources of interference. More importantly this avoids having to develop quality managed components as safety elements when they can interfere with safety-related components. By providing this isolation, the EB corbos Hypervisor helps reduce efforts for the analysis of interference while also creating a robust safety architecture for the software platform.

- **Allocation of platform resource organizational units:**
  The EB corbos Hypervisor enables the allocation of platform resources to different organizational units, providing each business unit with its own isolated environment or "walled garden." This segregation of resources ensures that each business unit can focus on its specific tasks without worrying about interference from other business units. This approach simplifies the work of safety managers, who can concentrate their efforts on the safety-related components, and also for software architects working on quality managed components who can focus on achieving the quality goals of the system functions they support.

- **Support for safety-related and quality managed hypervisor services:**
  The microkernel architecture of the hypervisor allows it to provide advanced services to quality managed virtual machines (VMs) while at the same time offering core services that are highly dependable to safety-related VMs. This versatility enables the hypervisor to accommodate both safety-related functions and quality managed functions, ensuring complete coverage of all platform requirements. This also allows for the integration of different software stacks, each serving their respective purposes while coexisting within the same software platform.

By providing a basis for safety architecture, allocating resources to different units, and accommodating both safety-related and quality managed functions, the EB corbos Hypervisor facilitates the development of a comprehensive and well-structured software platform. It allows for efficient management, isolation, and integration of different components, enhancing both the safety and quality aspects of the system.

8. Conclusion

In conclusion, to effectively and efficiently achieve safety in automotive software platforms, it is critical to identify and isolate safety-related components. The emphasis should be on reducing the safety footprint – focusing on those elements genuinely contributing to safety and then managing potential interference from non-safety components. This approach can result in significant benefits in terms of reduced sources of interference that a software platform must address, as well as cost efficiency and development agility, leading to a safer automotive software platform.

The value of the **EB corbos Hypervisor** lies in providing a foundation for construction of a safe and secure software platform. With its microkernel design, it creates a trustworthy base for safety architectures, effectively separating and allocating platform resources while maintaining the necessary isolation between safety-related and quality managed services of the hypervisor as well as virtual machines. The hypervisor's microkernel architecture contributes significantly to reducing the safety footprint, demonstrating a practical and powerful approach to achieving a secure and safety-oriented software environment.

Furthermore, by providing deep isolation of VMs, efficient resource management, and enhanced security, the EB corbos Hypervisor is ideally suited for a mixed-critical software platform. Its support for multiple operating systems allows flexibility and adaptability, while the compliance to ISO 26262 standard for ASIL-B safety elements guarantees its readiness to uphold high safety standards. This powerful foundation efficiently streamlines the management of system functions and resources, leading to a robust software platform that can accommodate various safety requirements and performance expectations.

Finally, the EB corbos Hypervisor presents an opportunity for the automotive industry to revolutionize the design and execution of software platforms of mixed-criticality. By mitigating interference and facilitating efficient resource allocation, it offers a versatile solution that fosters an environment conducive to the harmonious integration of diverse system components. Thus, the EB corbos Hypervisor emerges as a powerful tool that promotes safety, security, and quality, shaping the future of software platforms in the automotive industry and beyond.

About the author

Joel Thurlby
Senior Expert, EB corbos Safety, Elektrobit

Joel is a Senior Expert at Elektrobit, specializing in system architecture with a strong focus on functional safety in the automotive sector. He is passionate about blending cutting-edge automotive innovation with safety standards. Joel's expertise in software architecture and his deep understanding of safety standards bring a unique perspective to the forefront.

About Elektrobit

Elektrobit is an award-winning and visionary global vendor of embedded and connected software products and services for the automotive industry. A leader in automotive software with over 30 years of serving the industry, Elektrobit's software powers over 5 billion devices in more than 600 million vehicles and offers flexible, innovative solutions for car infrastructure software, connectivity & security, automated driving, and related tools, and user experience. Elektrobit is a wholly-owned, independently-operated subsidiary of Continental.

For more information, visit us at **elektrobit.com**

Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany

Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333

sales.automotive@elektrobit.com