**Title:**

Safeguarding Over-the-Air-Updates

**Authors:**

Dr. Kim Grüttner kim.gruettner@dlr.de

Henning Schlender henning.schlender@dlr.de

Keno Schwarze keno.schwarze@dlr.de

Dr. Bernd Westphal bernd.westphal@dlr.de

Dr. Ralf Stemmer ralf.stemmer@dlr.de

**Conference:**

11. Internationales Symposium für Entwicklungsmethodik von AVL

# 1. Motivation

In recent years, software complexity in automobiles has increased immensely. The figures for recalls in recent years show a growing cause in software errors and they are set to increase [2]. This means that the number and scope of software updates will also rise in the future. Clearly Over-the-Air Updates (OTA) should be favoured in the future, mitigating the burden and effort of visiting a repair workshop. OTAs can be used to fix bugs, close security gaps, but also to expand functions or increase performance. The automotive sector is undergoing a significant transformation process, shifting from traditional software architectures towards a centralised architecture in the Software-Defined Vehicle (SDV) [1, 28], which is well prepared for OTA.

However, such updates outside repair workshops are not without risks. After an OTA, the updated components may be impaired. This can have an impact on performance or even the safety of the vehicle and finally invalidate its roadworthiness. The UNECE R 156 standard [3] requires regular updates for vehicles built from 2024 onwards and specifies rules for the operation of a Software Update Management System.

There is a global trend towards a centralized architecture in the context of the SDV [1]. Here, the tasks of many individual electrical control units (ECUs) are combined in a few high-performance zonal or centralized computers. On the one hand, this approach reduces the variance in the hardware, offers greater flexibility and easier updatability. On the other hand, it harbours risks in the shared use of resources. For this reason, OEMs will probably restrict access to these resources through techniques like partitioning and containerization.

As future applications are subject to continuous optimization and updates, a transition of the development process to the DevOps cycle is inevitable. Part of this idea is that data from the fleet is collected during the operations phase and fed back into development.

## 2. Problem Statement

All of these factors named above, such as increasing update frequency in a DevOps cycle, and the trends towards the centralised SDV architecture, complicate the implementation of OTAs. There is a risk that the functional safety of the system can no longer be guaranteed. Suitable methods must be used to ensure that the components continue to work together correctly.

## 3. Assume / Guarantee Contracts of Extra-Functional Properties

To verify the correct operation of the components in terms of extra-functional properties, so-called Assume/Guarantee Contracts [4] can be used as part of a virtual integration test (VIT)[10]. Due to formally defined composition and refinement operations, it is possible to provide this proof for individual modules as well as entire partitions.

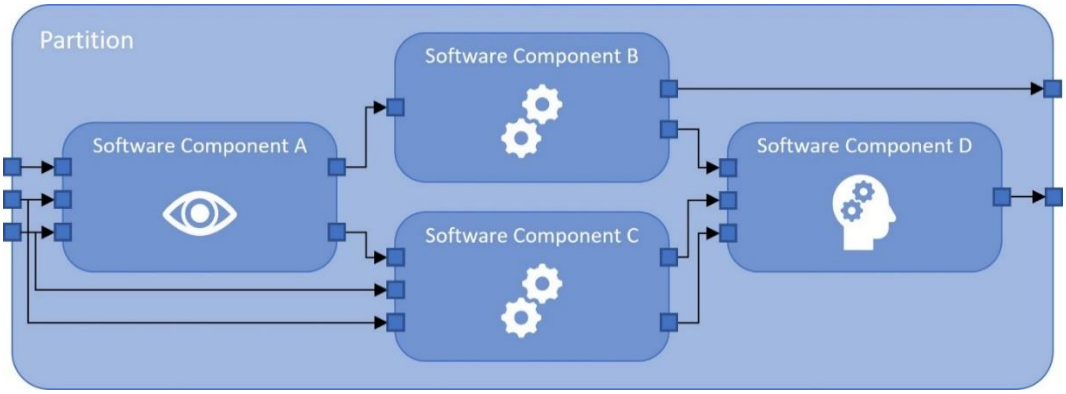


*Figure 1: Example System with 4 components labelled from A to B. [5]*

In this example (Figure 1) a partition includes software components (A – D) which performs image processing and decision making. Every component has well defined interfaces to communicate with the other internal components inside the partition or with external components. In Figure 2 is shown that every component specification now includes an assumption about the environment of the component and a guarantee, that describes the behaviour of the component in that environment [4]. To specify that behaviour the MTSL specification language is used [8, 9]. For example, the provision of an output event after an input event within a specified time. The refinement rules allow to derive specifications for subcomponents. This timing specification can then be used to check the system for coherency during the design phase and also during run-time.

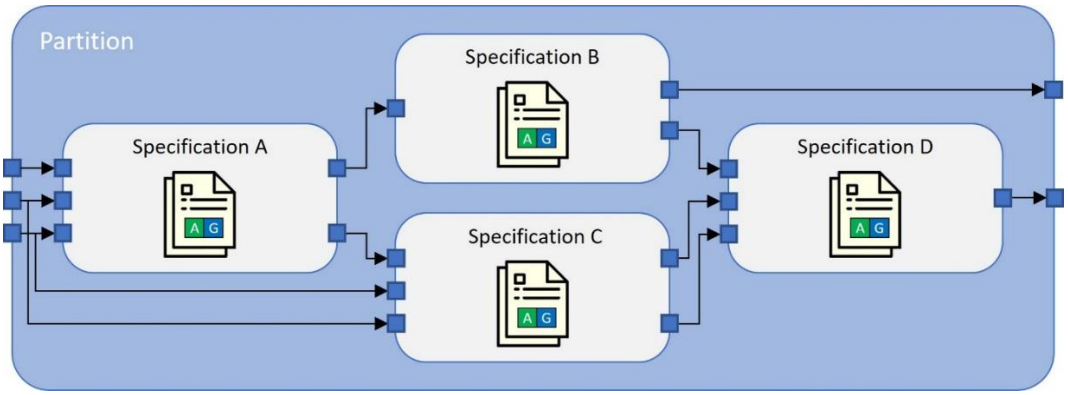


*Figure 2:The Example system from figure 1 with Assume / Guarantee contracts. [5]*

The temporal aspect of software system isn't the only factor when it comes to safety. Every application has individual resource requirements. To describe the hardware requirements of such a system, a module-based architecture formalism is proposed [6, 7]. It is able to express the influences of different hardware platforms, hypervisors, partitions, middleware layers, and applications. It supports hierarchical connected hard- and software layers, but one of the main features is the support of dynamic reconfiguration.

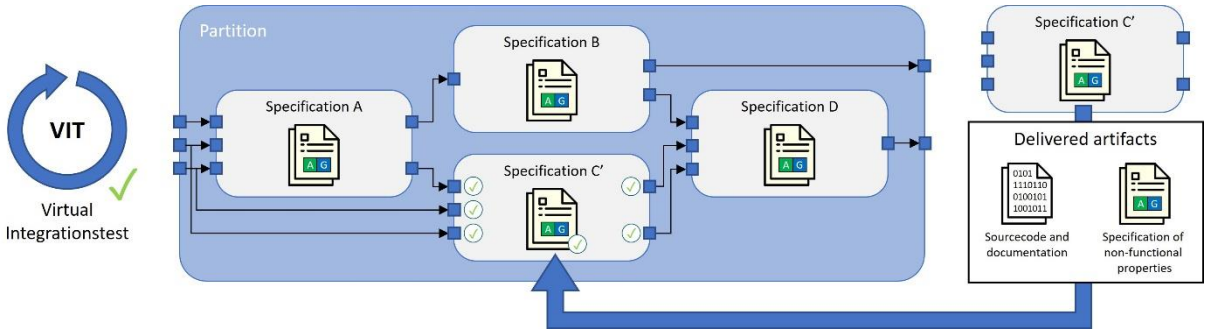


*Figure 3:The component C should be updated. With a VIT the Assume / Guarantee contracts are checked if the safe execution of the new component can be guaranteed. [5]*

Based on these formalisms with the help of Virtual Integration Tests (VIT) it is possible to guarantee the safe execution of complex SDV systems [10]. In Figure 3 the previous example is shown. This time the component C should be updated. The new version of the component is named C'. It is delivered with source code and the specification of the non-functional properties like timing and resources. During the VIT all contracts are checked. If and only if all checks were successful the correct execution of component C' can be guaranteed in this system. Providing this evidence is a complex and time-consuming process and should therefore be automated with the help of CI/CD pipelines.

# 4. Modularized Architecture

Centralizing software onto fewer, more powerful ECUs brings many benefits, but also challenges to be solved. A vehicle manufacturer should be able to source software components from external suppliers in a similar way as purchasing a complete and tests hard- software package in an ECU. This requires at the very least a description of which interfaces such software components need to have in order for the external supplier to develop and later the vehicle manufacturer to test against. Sharing an execution platform between rigorously tested safety critical components and possibly less tested non-critical applications requires guarantees of fault isolation from the underlying platform. Safe operation can be achieved by systemically

performing integration tests during development and runtime monitoring during deployment. Being able to this requires adequate means to model and specify the involved hardware and software platforms in respect to their behaviour and their resources.

Modelling tools for embedded systems like the [12, 13, 14] allow the modelling of automotive architectures, including resource assignments. They do however not provide the required semantic interpretation for modular, updatable platforms in design and update processes utilizing automated integration testing and monitor synthesis. Other approaches like [15] demonstrate the synthesis of software architectures based on resource requirements, though it does not consider the runtime middleware stack. The AUTOSAR Adaptive Platform [16] provides extensive platform and health monitoring and diagnosis tooling and extends platform health management [17] and POSIX OS support [18] in the AUTOSAR Adaptive specification. This approach does however not support dynamic reconfiguration and is not generic enough to model interfaces and modules in a uniform way across hard- and software configurations. An approach which provides modularity is shown in [19] by decoupling system integration information from their implementation and allows integration into an existing system at runtime. An explicit modelling of resource provision and consumption is not part of this architecture. An update-compatibility concept was introduced in [20], which allows the specification and checking of Resource- and Metadata (RMD) as well as Functional Event-Timing (FET) properties.

In [6] an approach is presented to describing software components dynamically reconfigurable modules. A module encapsulates specific functionality. It is defined and interacts with the overall system with interfaces which give a detailed description of the resources it requires and can provide. And it defines monitors which ensure adherence to the specifications given at the module's interfaces. The approach also covers the specification and analysis of resources to more dynamic and reconfigurable systems. A similar concept of annotating software modules with their resource requirements and provision is presented in the MADAM project [21] which shows an architecture model for adaptive systems.

## 4.1 The Modular Architecture Template

To close these gaps a modular architecture template is being developed at DLR-SE and first presented in [6]. It aims to provide a template by which to specify software components as dynamically reconfigurable modules with the primary focus being the modelling of resource allocation.

The definition aims to give a description irrespective of the module's type or which abstraction layer it exists on, so that the module definition encapsulates abstraction layers such as hypervisors, operating systems, virtualization or container engines as much as classical software applications.

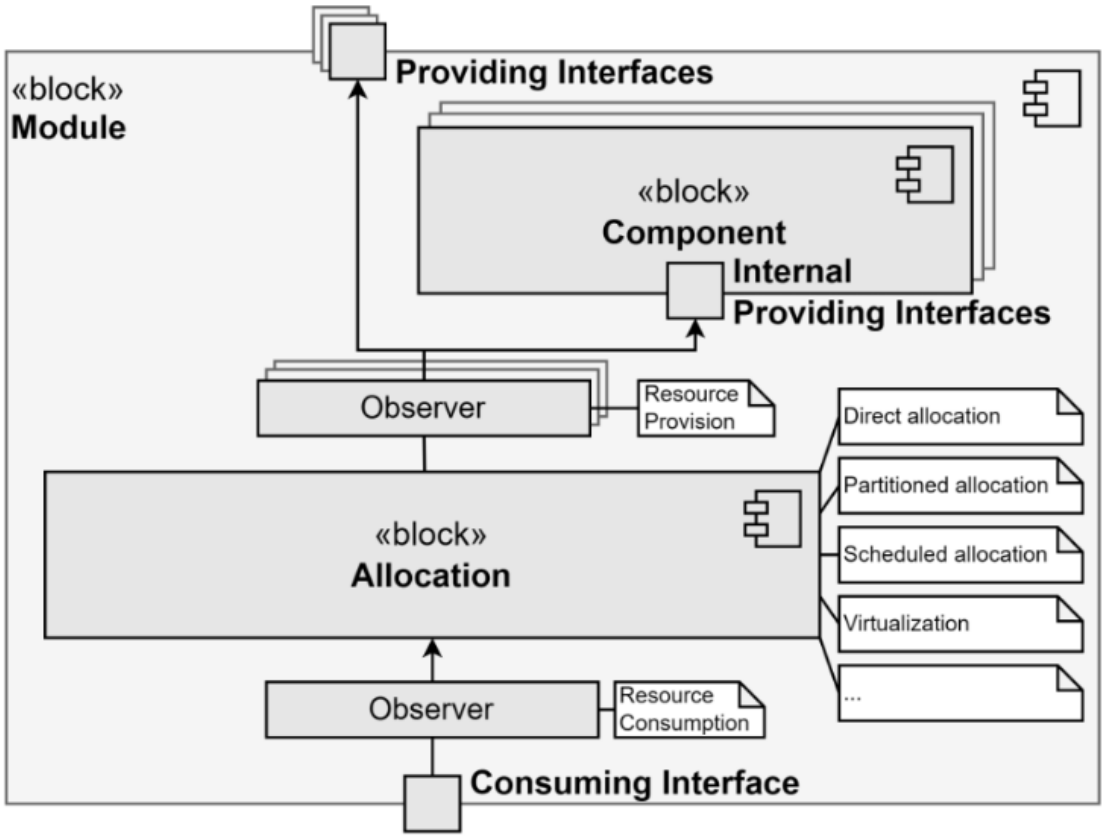


*Figure 4: The module architecture template as presented in [6]. The Allocation block claims and further distributes resources specified at the consuming interface. Components implement the functional behaviour of the module.*

A module as depicted in Figure 4 interacts with the rest of the system through one consuming interface at which the resource requirements to the underlying architecture are specified and an arbitrary number of providing interfaces. Each interface specifies the resources it consumes or provides. A resource is described by a property-value tuple. Properties can be roughly categorized into computation, memory, device access or interface metadata. Computation or memory properties are mainly relevant for the module-internals and performing resource availability checks before integration. Interface metadata describe the interface itself. This allows for checking compatibility between multiple modules when new modules are to be integrated or existing ones updated.

Each module contains an allocation block which claims the resources specified for the consuming interface. The implementation of this allocation depends of the type of the module and can either directly provide access to the claimed resource or do so in a scheduled or partitioned way. In order to give additional safety guarantees, the model explicitly defines resource observers to be synthesized after the consuming interface and after the allocation block. Should a module not adhere to its resource specifications at any time while deployed in the field, an alert can immediately be issued, providing an extra layer of assurance.

The component block encapsulates the implementation of a specific software functionality. A module contains a special type of component to manage its execution and requirements. This can for example be the provision of a runtime environment for other components or the provision of resources at the providing interface. Otherwise, components represent classical, service-oriented software applications (AUTOSAR: Composition; ISO 26262 Software Component) encapsulating specific functionality. A nested composition of components is also possible.

Each module is characterized by a type in addition to its interfaces. The type defines what sort of software component a module is implementing and can be e.g. a hypervisor or operating system. Depending on the actual type, its description may also be further narrowed down.

## 4.2 Isolation of Software-Components

The implementation of a modular architecture on a shared ECU, especially when implementing safety-critical modules requires their reliability must be guaranteed, as the shared hardware platform could lead to cascading faults with catastrophic consequences. One method of doing so is providing proper isolation between components [22].

In order to provide isolation, there needs to be a module which is able to partition and isolate the system like a type-1-hypervisor. Active research in this area shows the ability of existing open-source solutions to provide isolation [23]. The requirements to the isolation layer also depend on the underlying hardware. The more hardware that can be shared by modules, the more steps have to be taken to ensure isolation.

For example, assuming sufficient hardware resources, a safety critical component can be allocated exclusive access to processor cores, memory regions and outside interfaces. However, in some cases these resources may have to be shared. This can be due to an insufficient number of processor cores, shared cache between cores or shared communication buses. In these cases, the shared resources also need to be partitioned by way of e.g. fixed time slicing for processor and bus access and cache-colouring [24].

# 5. Run-Time Monitoring of Extra-Functional Properties

Run-time monitoring is a critical aspect of SDV development, as it enables the continuous observation and analysis of the vehicle's software and systems in real-time and ensures the system’s adherence to its specifications. While contract-based design lays a necessary foundation for building a secure and updateable system, monitoring the system’s real-world behaviour provides an extra layer of safety.

In the previous chapter we introduced a modular architecture template which is depicted in Figure 4. This template defines Observers which monitor resource consumption and provision before and after the module’s allocation block. These provide valuable safety assurances.

In addition to monitoring a software component’s adherence to its resource specifications it is also critical to ensure that at all times, timing constraints are met. This is accomplished by introducing observers for run-time monitoring of timing behaviour. To these ends, we actively pursue research in the domain of timing specification and observer synthesis.

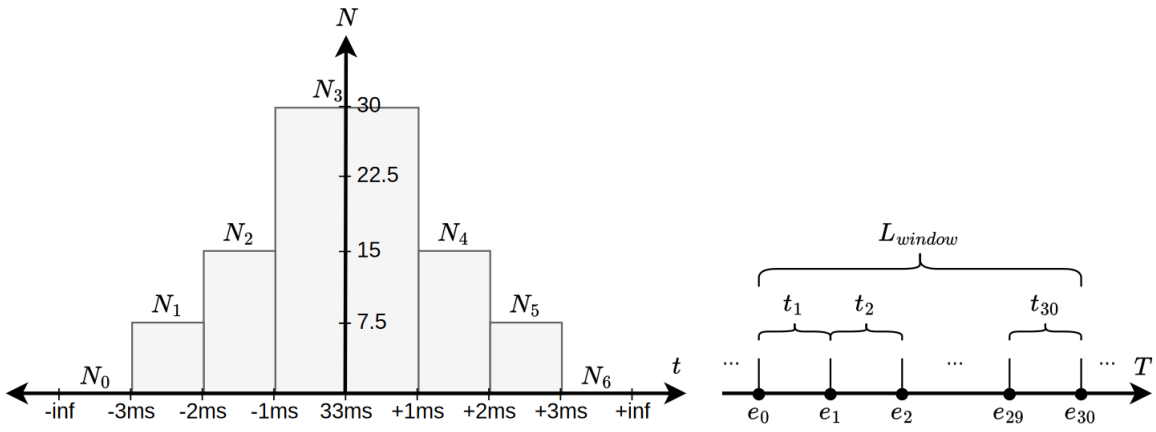


*Figure 5: Example of timing distribution. Left: A graphical representation of a histogram. Right: Timed event trace with time differences and window size*

Software running on any modern machine is unlikely to have a fixed execution time. This may be due to branching logic in the software itself where different branches take different amounts of time, the processor itself choosing different execution paths or resource contention with other processes. Consequently, to give a verdict on whether a software components execution times are behaving as expected, it is not enough to specify just one value for the execution time. Instead, we use statistical modelling to characterize an applications behaviour.

Figure 5 depicts an example of what such a characterization may look like, with the right side showing a timed event trace and the left side a histogram of observed timings. Whether or not an application is still behaving within its expected parameters is assessed by the difference between measured and specified behaviour with the specification determining how large of a delta is allowed for the system to still be considered in a safe state.

In order to analyse the timing behaviour, timed events must be recorded at runtime. Lightweight observers are integrated into the system for this purpose. This means, the impact of the observer execution should impact the monitored software's behaviour as little as possible. These influences can steam from the required CPU time or network bandwidth of the observers. To further optimize, any computationally intensive operations like the actual analysis should be offloaded to a dedicated machine. The implementation of an observer can depend on the nature of the platform and the needs of the application.

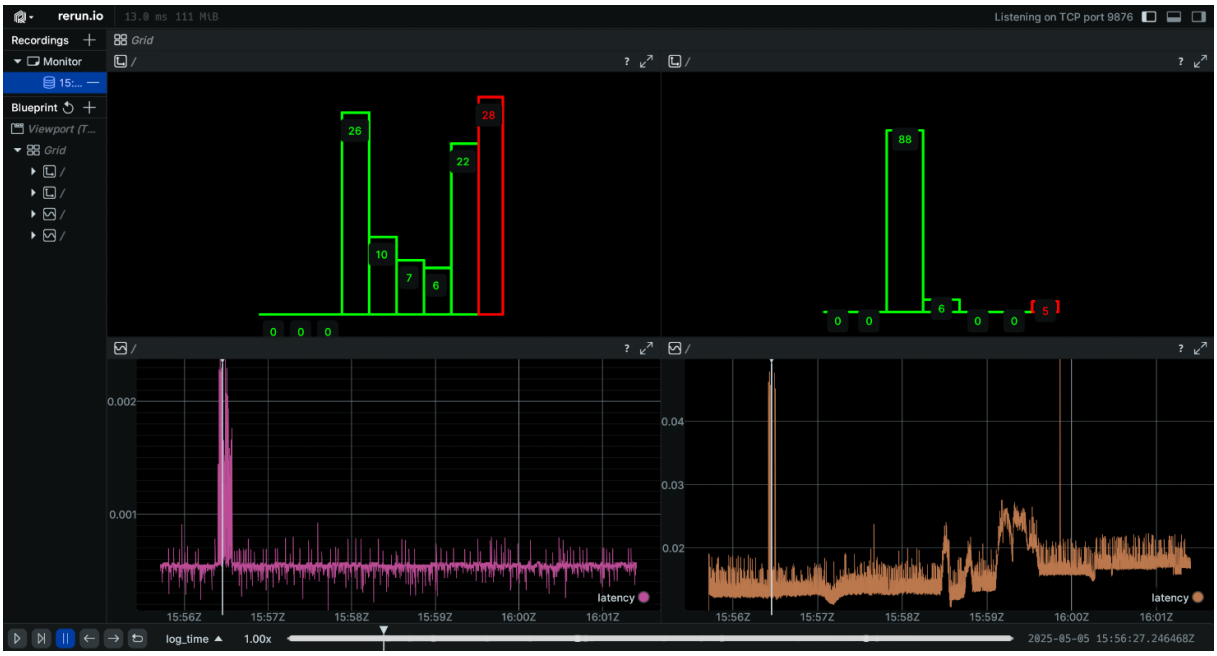


*Figure 6: Visualization of two monitors during the run of a demo scenario in our testbed. Left camera image latency over the network. Right: image processing time.*

Figure 6 shows the monitoring of two applications using Ethernet observers which examine the timing of applications at their communications interface. These rely on the networking implementation of the Linux kernel and utilize libcap [25] in order to inspect network packets.

More fine-grained information about a program's execution can obtained by observing the execution itself. This can be achieved by utilizing a processor architecture's inbuilt monitoring features such as the ARM Performance Monitoring Unit [26] which provides access to cycle and instruction tracking. Accessing the corresponding registers to retrieve measurement values will however require extra instructions and thus inevitably affect the measured times.

Going even further, open architectures like RISC V enable the definition of custom instruction sets like presented in [27]. From these, processors which enable monitoring at the hardware level can be synthesized.

Run-time monitoring is a critical aspect of SDV development, enabling the continuous observation and analysis of the vehicle's software and systems in real-time. By implementing run-time monitoring, developers can improve safety, enhance reliability, increase security, and reduce maintenance costs. While there are several challenges and opportunities to consider, the benefits of run-time monitoring in SDVs make it an essential component of modern vehicle development.

# 6. Conclusion

The challenges in this transformation process are immense for both OEMs and suppliers. The responsibility for this transformation process lies at the OEMs, but only stronger networking within the supply chain can lead to success. In addition, the decoupling between hardware and software and the software integration & verification processes on a unified hardware abstraction layer must be automated due to the increasing frequency of updates. Our approach represents an important building block in this complex transformation process.

# 7. ACKNOWLEDGMENT

This work is funded by the German Federal Ministry for Economic Affairs and Energy under grant number 16THB0009.

# 8. References

[1] C. Bodei, M. D. Vincenzi und I. Matteucci, „From hardware-functional to software-defined vehicles and their security issues," *IEEE 21st International Conference on Industrial Informatics (INDIN)*, 2023.

[2] J. Pancik und et al., „Auto recalls and software quality in the automotive sector," *EAI Endorsed Transactions on Scalable Information Systems 5.17*, 2018.

[3] UNECE, „Proposal for a New UN Regulation on Uniform Provisions Concerning the Approval of Vehicles with Regards to Software Update and Software Updates Management System," *ECE/TRANS/WP.29/2020/80,* 2020.

[4] J. S. Becker, B. Koopmann, I. Stierand und L. Westhofen, „Providing Evidence for Correct and Timely Functioning of Software Safety Mechanisms," *Software Engineering 2023 Workshops,* Bd. Workshop Automotive Software Engineering, pp. 66 - 77, 2023.

[5] H. Schlender, B. P. Koopmann und K. Rothemann, „Automotive Software Updates," *Webinar-Reihe des Transformations-Hubs TASTE,* 2023.

[6] P. Uven, R. Stemmer und G. Nitsche, „A modular architecture template for resource modeling in Software-Defined Vehicles," *Dependable Computing -- EDCC 2025 Workshops,* 2025.

[7] I. Yraza, I. Agirre, I. Mugarza, G. Nitsche, P. Uven und J. M. Orbegozo, „Towards a Contract-Based Definition of Update-Compatibility – Modelling Safety Integration Criteria," *IEEE 26th International Conference on Intelligent Transportation Systems (ITSC),* 2023.

[8] E. Böde, et al., "Design Paradigms for Multi-Layer Time Coherency in ADAS and Automated Driving (MULTIC)", FAT Schriftenreihe 302, VDA, 16.10.2017

[9] E. Böde, et al., "MULTIC Tooling", FAT Schriftenreihe 316, VDA, 11.07.2019

[10] Damm, Werner, et al. "Using contract-based component specifications for virtual integration testing and architecture design." *2011 Design, Automation & Test in Europe*. IEEE, 2011.

[11] Kröger, Janis & Koopmann, Björn & Stierand, Ingo & Fränzle, Martin. (2023). Contract-based specification of mode-dependent timing behavior. Innovations in Systems and Software Engineering.

[12] Object Management Group, "OMG SysML", accessed 10.09.2025, [Online]. Available: https://www.omgsysml.org/

[13] Eclipse Foundation, "Eclipse APP4MC", accessed 10.09.2025. [Online]., Available: https://eclipse.dev/app4mc/

[14] R. Hottger, H. Mackamul, A. Sailer, J.-P. Stegh ¨ ofer, and J. Tessmer, "APP4MC: Application platform project for multi- and many-core systems", Information Technology, vol. 59, no. 5, pp. 243–251, 2017., [Online]. Available: https://doi.org/10.1515/itit-2017-0019

[15] S. Kugele, P. Obergfell, and E. Sax, "Model-based resource analysis and synthesis of service-oriented automotive software architectures", Software and Systems Modeling, vol. 20, no. 6, pp. 1945–1975, Sep. 2021

[16] AUTOSAR, “Specification of health monitoring,” AUTOSAR, Tech. Rep.,2021, accessed 10.09.2025, [Online]. Available: https://www.autosar.org/fileadmin/standards/R21-11/FO/AUTOSAR_ASWS_HealthMonitoring.pdf

[17] AUTOSAR “Specification of platform health management,” AUTOSAR, Tech., Rep., 2024, accessed 10.09.2025, [Online]. Available: https://www.autosar.org/fileadmin/standards/R24-11/AP/AUTOSAR_AP_SWS_PlatformHealthManagement.pdf

[18] AUTOSAR “Specification of operating system interface,” AUTOSAR, Tech., Rep., 2024, accessed 10.09.2025, [Online]. Available: https://www.autosar.org/fileadmin/standards/R24-11/AP/AUTOSAR_AP_SWS_OperatingSystemInterface.pdf

[19] A. Kampmann, B. Alrifaee, M. Kohout, A. Wustenberg, T. Woopen, ¨ M. Nolte, L. Eckstein, and S. Kowalewski, "A dynamic service-oriented software architecture for highly automated vehicles", 2019 IEEE Intelligent Transportation Systems Conference (ITSC). IEEE, 2019, pp. 2101–2108.

[20] I. Yarza, I. Agirre, I. Mugarza, G. Nitsche, P. Uven, and J. M. Orbegozo, "Towards a contract-based definition of update-compatibility – modelling safety integration criteria", 2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC), Bilbao, Spain, 2023, pp. 710-717

[21] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven, "Using architecture models for runtime adaptability", *IEEE software*, vol. 23,no. 2, pp. 62–70, 2006

[22] Irune Agirre, Irune Yarza, Imanol Mugarza, Jacopo Binchi, Peio Onaindia, Tomasso Poggi, Francisco J. Cazorla, Leonidas Kosmidis, Kim Grüttner, Patrick Uven, Mohammed Abuteir, Jan Loewe, Juan M. Orbegozo, and Stefania Botta., "Safe and secure software updates on high-performance mixed-criticality systems: The UP2DATE approach.", Microprocessors and Microsystems, Volume 87, 2021

[23] Martins, José, and Sandro Pinto., "Shedding light on static partitioning hypervisors for arm-based mixed-criticality systems", 2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS), San Antonio, TX, USA, 2023, pp. 40-53

[24] Suzuki, N., Kim, H., De Niz, D., Andersson, B., Wrage, L., Klein, M., Rajkumar,R, "Coordinated bank and cache coloring for temporal protection of memory accesses", 2013 IEEE 16th International Conference on Computational Science and Engineering, Sydney, NSW, Australia, 2013, pp. 685-692

[25] libpcap. Home | TCPDUMP & LIBPCAP, accessed 10.09.2025, [Online]. Available: https://www.tcpdump.org

[26] ARM, “Performance Counters”, accessed 10.09.2025, [Online], Available: https://developer.arm.com/documentation/109847/9-

3/Performance-analysis-concepts/Performance-counters

[27] N. R. Nanjundaswamy, G. Nitsche, F. Poppen and K. Grüttner, RISC-V Timing-Instructions for Open Time-Triggered Architectures, 2023 53rd Annual IEEE/IFIP International Conference on Dependable, Systems and Networks Workshops (DSN-W), Porto, Portugal, 2023

[28] MÁTÉ, Bálint, et al. "Software-Defined Vehicles: Bridging Industry and Research Perspectives". In: 2025 IEEE 34th International Symposium on Industrial Electronics (ISIE). IEEE, 2025. S. 1-6.

# 9. Summary

In recent years, software complexity in automobiles has increased immensely. The figures for recalls in recent years show a growing cause in software errors and they are set to increase. Clearly Over-the-Air Updates (OTA) should be favoured in the future, mitigating the burden and effort of visiting a repair workshop. Such updates can have an impact on performance or even the safety of the vehicle and finally invalidate its roadworthiness. The automotive sector is undergoing a significant transformation process, shifting from traditional software architectures towards a centralised architecture in the Software-Defined Vehicle (SDV), which is well prepared for OTA. Yet, it harbours risks in the shared use of resources and hence there is a risk that the functional safety of the system can no longer be guaranteed after an update. Suitable methods needs to be used to ensure that the components continue to work together correctly.

One approach to ensure such extra-functional properties on centralised architectures are so-called Assume/Guarantee Contracts and virtual integration tests. Due to formally defined composition and refinement operations, it is possible to provide proofs for individual modules as well as entire partitions. The idea behind partitions is to minimise the influence between software components and thus to decouple them in terms of resource utilisation. In order to confirm that there is no impairment even after the update process, runtime or onboard monitors are generated from the specification, which later recognise errors or an exceeding of the resource budget during operation.