THE
AUTONOMOUS
Working Group Safety & Architecture

Safe Automated Driving: Requirements and Architectures

Second Edition

AUTHORS

tttech auto

Lead Contributor:
Moritz Antlanger

Working Group Lead:
Sascha Drenkelforth

Christian Mangold

Georg Niedrist

NXP

Shailesh More

DENSO
DENSO AUTOMOTIVE Deutschland GmbH

Justin-Kiyoshi Tiele

KTH VETENSKAP OCH KONST

Martin Törngren

Sina Borrami

Naveen Mohan

Infineon

Abhijit Ambekar

Udo Dannebaum

UNIVERSITÀ DEGLI STUDI FIRENZE
Da un secolo, oltre.

Andrea Bondavalli

Nahla Ben Mosbeh

Gabi Escuela

Marius Weiss

FORMER CONTRIBUTORS:

Lucas Fryzek
Ayhan Mehmed
Friedrich Reisenberger

Jens Rosenbusch
Christoph Schulze
Chaitanya Shinde

Matthew Storr
Neil Stroud
Kazuhito Takenaka

REVIEWERS

The Working Group Safety & Architecture would like to thank the reviewers of the report for their valuable input and feedback.

Bernhard Kaiser
Ansys Germany GmbH

Philip Koopman
Carnegie Mellon University

Andrei Terechko
NXP Semiconductors N.V.

Jan Toennemann
Vector Informatik GmbH

Rasmus Adler

Simon Fürst

CONTENTS

VERSION HISTORY

Version	Date	Revision description
2.0	15.09.2025	Second edition
1.0	01.12.2023	Initial release

KEY UPDATES OF THE SECOND EDITION

Since the release of the first edition report in late 2023, the following major changes and additions have been implemented:

- **Candidate architectures**
  Three new candidates for the conceptual system architecture of an AD Intelligence have been identified, described, and evaluated, namely the Cross-Checking Pair, Daruma, and AD-EYE architectures.
- **Relevant safety standards**
  The analysis and discussion of safety standards relevant in the development of an AD Intelligence has been greatly extended. The following regulations and standards have been considered: UNECE R157, ISO/IEC TR 5469, ISO/PAS 8800, UL 4600, and ISO/TS 5083.
- **Sufficient Independence**
  A detailed discussion how to achieve Sufficient Independence between the subsystems making up an AD Intelligence has been added. This includes a semi-quantitative scheme "Independence Coverage" for evaluating whether Sufficient Independence has been achieved.
- **Implementation considerations**
  Additional aspects relevant for the mapping of conceptual system architectures to hardware and software implementations have been considered, e.g., crypto agility for post-quantum cryptography.
- **Architecture descriptions**
  The diagram style and level of depth of the architecture descriptions have been made more consistent to allow easier comparison between different candidate architectures.
- **Terminology**
  The terminology used throughout the document has been improved and made more consistent.

REPORT SUMMARY

The Autonomous is an initiative and open platform bringing together leading executives and experts of the mobility ecosystem to align on subjects relevant to the safety of autonomous driving (AD); in its Safety & Architecture Working Group, members of international research institutes and industrial companies came together to investigate what the system-level *conceptual* architecture of an automated vehicle could look like, in order to address the safety challenges of automated driving.

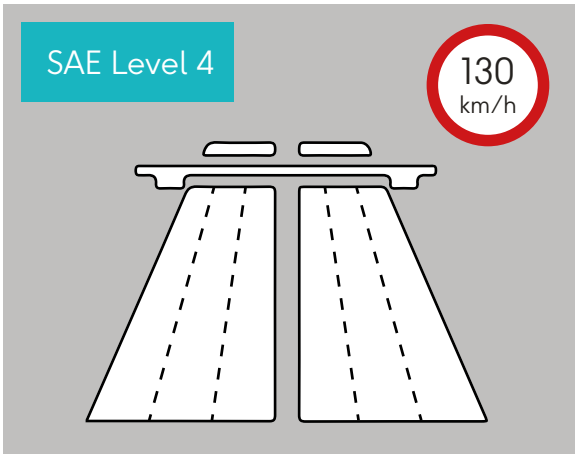
This report was compiled from June 2021 to December 2023 for its first edition, and from 2024 through 2025 for the second edition which is presented here; both editions were accompanied by external reviewers from industry and academic experts.

Our work is structured as follows: we start by outlining the reference use case of an SAE L4 Highway Pilot (HWP), derive key system requirements, and establish general constraints and well-known design principles for implementing such a use case in an AD system. We continue with candidate architectures from market and literature research and derive their properties, then compare the architectures with respect to a set of criteria that we consider crucial (such as system availability, robustness, and scalability). Finally, we conclude with development and implementation considerations (like the challenge of Sufficient Independence between redundant elements), and discuss the impact of relevant standards and of security requirements on the architectures.

The intended readers are system owners who make architectural decisions and ensure consistency on many different abstraction levels, from high-level conceptual architectures to low-level physical implementations. Our intention is to support them in making such decisions and building up a safety argumentation.

ABSTRACTION LEVEL AND REFERENCE USE CASE

It is commonly understood and accepted that the development of a safe automated driving system for complex driving tasks is a big challenge. Even when developed to the highest standards, complex HW and SW elements will exhibit faults and functional insufficiencies that can materialize anytime. Still, the overall autonomous driving system needs to tolerate malfunctions and keep up operation at least for a minimum time frame – i.e., it needs to be fail-degraded. A well-chosen architecture is indispensable to manage the complexity of autonomous driving systems and to ensure fault tolerance in an effective and efficient way.



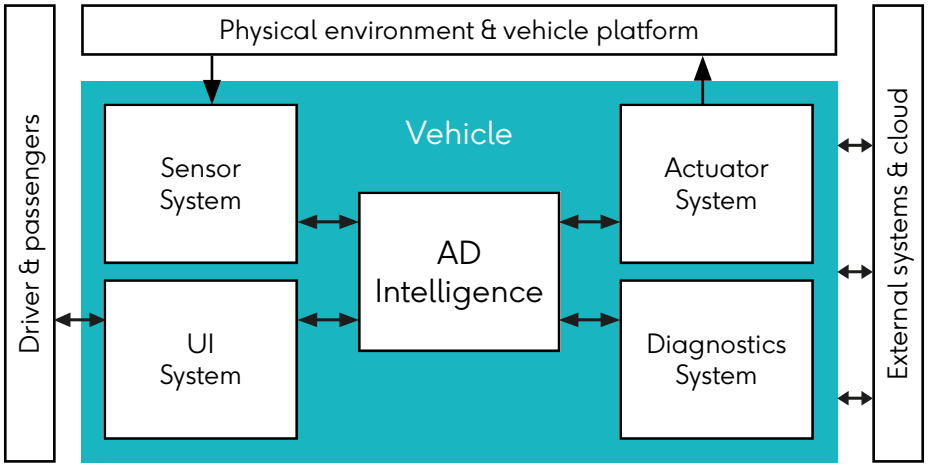
For our analysis, we have chosen what we call the conceptual system architecture level: we consider the system as a set of well-encapsulated subsystems that can comprise up to an entire processing channel, or at least a dedicated subset of the overall functionality.

As our reference use case, we chose an SAE Level 4 Highway Pilot to act as a backdrop for compiling assumptions and deriving system requirements and design principles applicable to conceptual system architectures. Similar features are being offered in a limited way on

the market today (as of 2025, L2 Highway Assistant systems and L3 systems with restrictions on speed, lane changes, and lead vehicles) and a (user-owned) SAE L4 HWP with full functionality is expected within the next few years; it will need to deal with complex traffic situations and will necessitate non-trivial system architectures due to high availability requirements and complexity.

SYSTEM REQUIREMENTS, DESIGN CONSTRAINTS AND DESIGN PRINCIPLES

The Safety & Architecture Working Group focuses on a part of the AD system providing the central driving algorithms, which we call the **Automated Driving Intelligence** (ADI). This system covers all cognitive tasks previously performed by the driver. A simplified representation is shown on the left, illustrating the four other systems the ADI is connected to, as well as the elements that "close the loop" with the physical environment.



A set of key system requirements (summarized in the table on the right) should be applied to the ADI to ensure the safety of commands to the actuators. Besides the expected timeliness, correctness, and consistency of commands, their availability is highly safety-relevant for an SAE L4 function. Additionally, an ADI architecture shall foresee self-diagnostic mechanisms and shall support detecting and handling functional insufficiencies (including but not limited to the perception functions).

S1	ADI output timeliness
S2	ADI output availability
S3	ADI output correctness
S4	ADI output consistency
S5	Perception malfunction detection
S6	ADI diagnostics

When coming up with conceptual system architectures intended to satisfy these system requirements, technological limitations constrain how high-reliability systems can be designed, built, and tested using realistic HW and SW components. Such general constraints need to be addressed by an architecture for automated driving, e.g.: it is impossible to avoid design faults and single-event upsets in large and complex monolithic systems, and it is impossible to achieve high availability by testing or to specify all edge cases that an AD function must cope with.

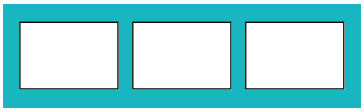
In addition, well-established practices should be respected in a sound conceptual system architecture. We identify such design principles, e.g., including using well-encapsulated independent subsystems ("Fault Containment Units", FCUs), applying diversity and redundancy, preventing emergent behavior by limiting interactions between subsystems, and mitigating hazards by adopting the Swiss cheese model.

CANDIDATE ARCHITECTURES

From industry publications, academic papers, and patent publications we have identified candidate architectures and grouped them into three basic categories of conceptual system architectures:

1. MONOLITHIC ARCHITECTURES

represent the status quo for SAE L2 systems and are a natural basis for incremental development to L4 systems.



2. SYMMETRIC ARCHITECTURES

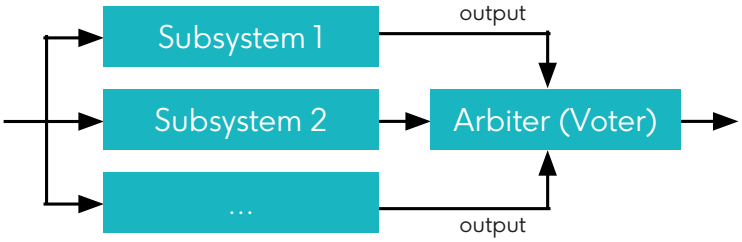
rely on multiple channels providing the same or similar functions, often with some voting mechanism for arbitration.

3. ASYMMETRIC ARCHITECTURES

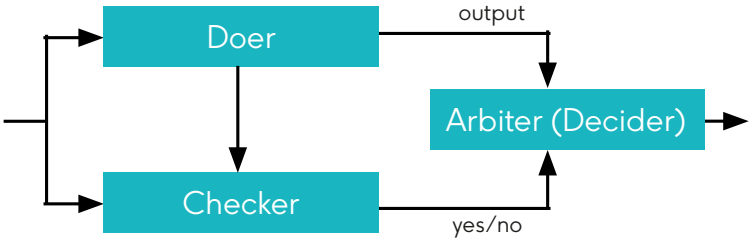
employ diverse decompositions to reduce the complexity of some subsystems, e.g., via monitoring elements or fallback channels with reduced functionality.

These architectures employ several underlying patterns the most significant ones are as follows:

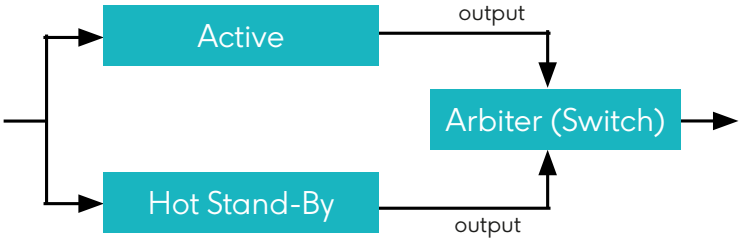
- **The Arbitration pattern** manages redundancy by deciding (e.g., voting) between equal channels. The Agreement pattern is similar, but without an external arbiter.



- **The Doer / Checker pattern** asymmetrically decomposes (for correctness) a channel into a Doer performing the intended function and a Checker approving it.



- **The Active / Hot Stand-By pattern** asymmetrically decomposes (for availability) into a preferred Main channel and – if that is not available – a Fallback channel.



A representative of monolithic architectures is the **Single-Channel** architecture, where a single ECU performs all tasks of the AD Intelligence, i.e., processes the sensor data into a consistent environmental model, generates trajectories and, finally, set points for the actuators.
Examples of this architecture are the AUDI zFAS originally intended for an SAE L3 Traffic Jam Pilot (2017), Tesla's "Full Self Driving" (FSD)[1], or monolithic end-to-end AI systems.

The **Majority Voting** architecture as a representative of the symmetric architectures implements a set of channels (three or more), each of which can perform the full nominal function. The voter compares (exactly or inexactly) the channels' results and forwards the majority opinion to the actuators. If all three results differ, no decision can be made. To achieve fault tolerance, multiple instances of the voter may be necessary.

The **Cross-Checking Pair** is another symmetric architecture and is inspired by the deceptively simple approach of constructing a fail-operational system by "making two channels that check

[1] As far as can be judged from available documentation.

each other". The mutual cross-check of the (complex) channels is augmented by two simpler selectors that collect actuator commands and validation results from the channels and forward the best-ranked actuator commands that are considered safe by both channels.

The **Daruma** architecture can be considered an extension of the cross-checking pair architecture; it consists of three complex channels that each output a trajectory (possibly with different focus, e.g., nominal vs. safety) and their environmental model. A "Daruma" subsystem cross-checks all output trajectories against all environmental models and ranks the channel outputs according to safety and performance criteria. Finally, two redundant selectors forward the best ranked results to the actuator system.

The first of the considered asymmetric architectures is the **Channel-Wise Doer/Checker/Fallback** architecture, where a Doer performs the nominal driving function and can resemble an SAE L2 system, while a Fallback performs only Minimal Risk Maneuvers. A Checker validates both the Doer's and the Fallback's output. A Selector receives the Checker's verdict and forwards either the trajectory from the Doer or from the Fallback to the actuators. Doer/Checker/Fallback are complex subsystems, and each of them can fail arbitrarily. They are implemented in a diverse way to minimize common cause failures, to ensure sufficient independence. The Selector is simple, has low performance requirements, and can be fully verified to preclude systematic faults. To achieve fault tolerance, it consists of redundant instances.

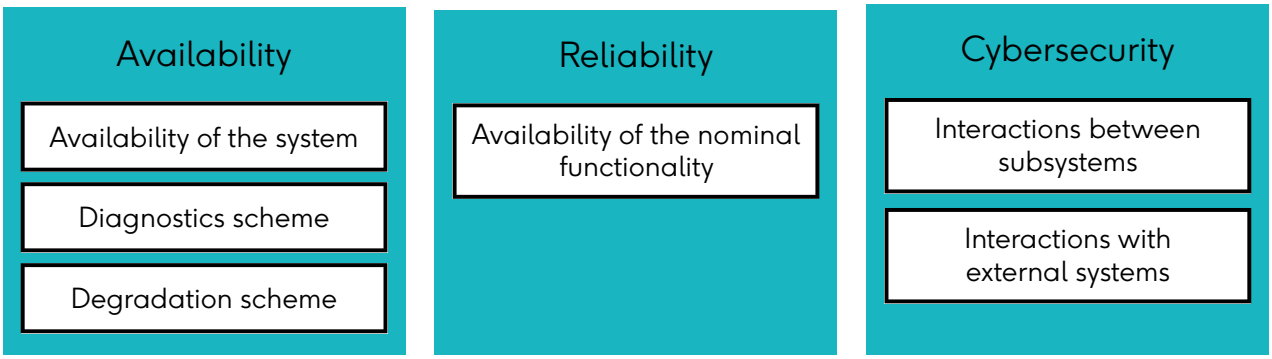
Another asymmetric architecture is the **Layer-Wise Doer/Checker/Fallback**, a dual-channel approach with at least one primary and one safing channel, which provides a degraded mode of operation in case the primary channel fails. Each channel consists of Doer/Checker pairs, arranged in multiple layers of the Sense-Plan-Act model. A Priority Selector determines the output to be sent to the actuators, depending on the states of the channels. The Priority Selector is a high safety integrity component, simpler than the Checkers. It must continue to operate in the presence of failures to deliver either the primary or the safing output, or to trigger an emergency stop. It may fail silently so long as that failure triggers an emergency (blind) stop as a very rare last resort.

The **Distributed Safety Mechanism** architecture can be seen as a more complex, distributed variant of the Doer/Checker/Fallback approach. The architecture is composed of three channels, each of them containing safety monitors – a Nominal Channel, consisting of the function itself and controlled by a Function Monitor, an Emergency Channel, which is controlled by a Controller Safety Mechanism, and a Safety Channel, which is controlled by a Vehicle Safety Mechanism. The Function Monitor is checking for SOTIF issues, the Controller Safety Mechanism is responsible for monitoring all the function controllers (including hardware and software platforms) and the Vehicle Safety Mechanism. The Vehicle Safety Mechanism is responsible for monitoring the communication networks and the Controller Safety Mechanism. It can send control commands to the vehicle actuators in case of comfort or safe stop, by using independent sensor data.

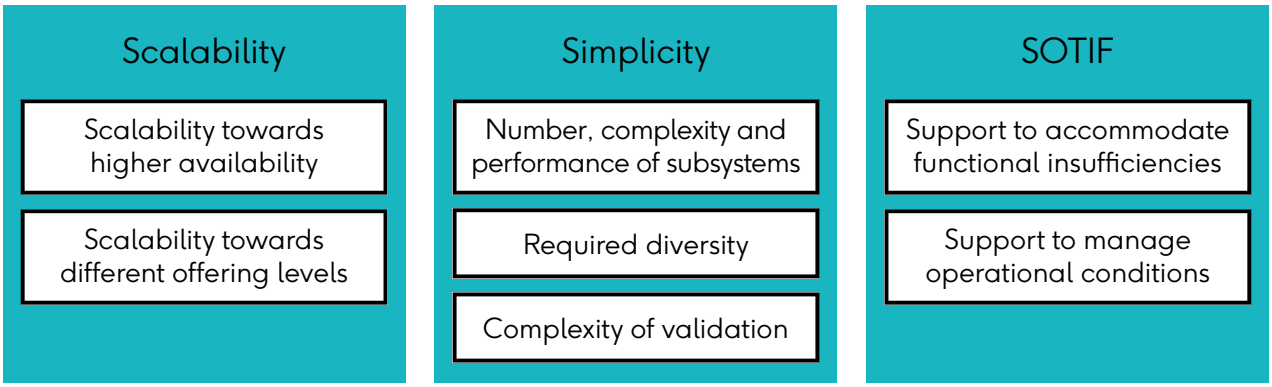
As a final asymmetric example, we study the **AD-EYE** architecture, which resembles the Channel-wise Doer/Checker/Fallback architecture. It employs a complex main channel with built-in degradation and recovery mechanisms to cope with manageable (e.g., temporary) faults, and a simpler secondary channel with a validation component for the main channel and a robust fallback capability. The validation component monitors the main channel and can restrict its operational envelope for a graceful degradation; only for the most severe faults is the fallback capability used. Finally, two redundant selectors forward the results from either the main channel or the fallback capability to the actuator system, based on the decision of the validation component.

ARCHITECTURE EVALUATION METHODOLOGY AND CRITERIA

We evaluate the presented architectures with respect to several key criteria:



- **Availability:** to what extent would the architecture support the fail-operational property, i.e., enable safe operation even in the case of unavoidable electronic or software faults?
- **Reliability:** would continuity of the nominal functionality be well supported, to help ensure a positive user experience, e.g., by avoiding function degradation?
- **Cybersecurity:** would the architecture be susceptible to security threats, or would it support resilience measures against attacks?



- **Scalability:** to what extent would cost-efficient downscaling to lower SAE levels (for vehicle options), or upscaling to higher SAE levels (for future enhancements), be supported?
- **Simplicity:** is the architecture be conceptually simple, to support modular development, verification, and validation?
- **Safety of the Intended Functionality:** would the architecture help ensure robustness and safe operation in the presence of functional imperfections and unavoidable edge cases?

For the evaluation itself, we proceed in three steps: To form an unbiased basis for the evaluation, we start with a generic evaluation of each architecture, by listing observations (properties of each architecture) related to each criterion. Next, we evaluate the relative significance of the above criteria for the selected use case of an SAE L4 Highway Pilot. Finally, we directly compare the architectures, considering the observed properties from the generic evaluation and inferring merits or weaknesses with respect to each evaluation criterion, and qualitatively ranking them under that criterion.

KEY FINDINGS

Based on detailed evaluations, we find that the **asymmetric architectures offer advantages over symmetric ones**; all of those architectures are based on the Doer/Checker/Fallback pattern, albeit in different levels and realizations - Channel-wise DCF, Layer-wise DCF, DSM, and AD-EYE have been studied in this report. By virtue of their inherent diversity of computational streams, they exhibit more robustness with respect to availability, cybersecurity, and SOTIF because the channels complement each other and tend to mutually compensate for their potential weaknesses. The asymmetric architectures also offer more options with respect to scalability, as omitting channels quite naturally leads to lower SAE level functionality, and higher levels can be reached by adding channels. Superficially, they might appear more complex than symmetric architectures. However, their diversity also facilitates modular development and independent verification of the channels, which in turn is expected to lead to lower development costs and enhanced availability.

Some symmetric architectures such as Cross-Checking Pair or Daruma can be implemented with strongly diverse channels; in the case of Daruma this is even explicitly foreseen. If so, these architectures can reap the benefits of diversity and asymmetry; they can be regarded as clever (multiple) realizations of the Doer/Checker/Fallback pattern.

Other symmetric architectures (based on voting and agreement approaches) are seen as much more susceptible to common cause deficiencies that might impact all channels at the same time – be it from the functional safety, SOTIF, or even the cybersecurity perspective. If this problem is addressed by heterogeneous channel implementations, e.g., diverse chipsets or diverse algorithms, then voting becomes unwieldy since channels might come to different but equally valid solutions; for minor discrepancies between channel outputs, tolerance ranges can be allowed for ("inexact voting"). Still, fundamentally different but correct solutions (e.g., pass an obstacle on the left vs. right) cannot be covered by voting with tolerance ranges and cannot be distinguished from situations where a channel is faulty.

Finally, the monolithic Single-Channel architecture is not seen as a feasible solution: it does not fulfill any of the criteria without additional internal redundancy and supervision mechanisms that are introduced during implementation – which would make it evolve into one of the other architectures.

IMPLEMENTATION CONSIDERATIONS AND SUFFICIENT INDEPENDENCE

We consider selected topics related to the further refinement of the conceptual system architecture into a hardware and software safety concept. This includes a discussion on different software architectural styles – depending on the use case – as well as common safety measures and cybersecurity aspects. For further refinement of the conceptual system architecture into combined hardware/software solutions with redundant channels, we need to consider dependent failures of the elements – because redundant elements, like proposed in all architectures except the monolithic Single-Channel, must not fail simultaneously. In other words, *Sufficient Independence* of the channels (which includes freedom from interference) and the absence of single points of failure need to be ensured. We discuss coupling factors that can lead to dependent failures and provide hints how to overcome them.

In practical implementations, there is a natural tension between the desire for mostly homogeneous redundancy (by deploying the same kind of SoC, operating system, application algorithms, etc. to reduce development cost) and the perceived need for fully heterogeneous

redundancy (i.e., maximum diversity to avoid systematic common cause failures). To this end, we propose a new methodology for evaluating Independence of an intended implementation which avoids subjective and maybe wishful reasoning, and instead systematically assigns an "independence coverage" to each dependent failure initiator. The methodology has been derived in analogy to ISO 26262's *diagnostic coverage* approach and gives a solid semi-quantitative evaluation if Sufficient Independence is achieved.

ARCHITECTURE DESIGN AND STANDARDS COMPLIANCE

To achieve a sound safety argumentation for the chosen architectures, we refer to the relevant safety standards, in particular ISO 26262 and ISO 21448. In addition, we propose advanced methods like formal verification on the architecture level and for the logical-to-physical mapping, as well as Markov modeling to quantify the overall system availability, to meet an ASIL target.

Furthermore, relevant standards for the implementation of an ADI have been analyzed, and the relevance of architectural design to achieve standards compliance is described in this report:

- Implementation according to **ISO 26262** (Functional Safety) will realistically require ASIL decomposition to achieve the prescribed limits on failure rates, so as to be able to integrate HW and SW modules that are not available or feasible in ASIL D quality. The functional partitioning of the architectures described in this report provides a solid basis for such a decomposition, i.e., for achieving ISO 26262 compliance of an ADI. Such compliance is likely not achievable with a monolithic system approach.
- Although dedicated architecture design and evaluation steps are not explicitly prescribed by **ISO 21448** (SOTIF), the standard is structured around the sense-plan-act paradigm and frequently refers to a fallback entity in case of faults, both implying an architectural system partitioning that is provided by all except the monolithic approach. ISO 21448 also acknowledges that a suitable modular architecture will reduce the effort for V&V.
- **UNECE R157** defines functional and safety requirements for type approval of vehicles that provide SAE L3 or higher HWP functionality and is applicable for many countries. It does not prescribe a specific architecture, but clearly a well-designed architecture will support fulfilment of the safety requirements of R157 by design and will enable to systematically prove compliance, instead of seeking evidence of compliance via trial-and-error based testing of a monolithic black-box system.
- **ISO TR 5469** and its automotive specific variant **ISO/PAS 8800** have both been published in 2024 to address Functional Safety for AI-based systems. Besides extensively specifying development measures for AI components to enhance system properties like reliability, predictability, and robustness, both also propose architectural measures. The concrete patterns suggested are a voting approach among diverse AI channels, a Doer/Checker approach including limitation of the output, and a Doer/Fallback approach; while the Doer is an AI-based component, the Checker is potentially AI-based, and the Fallback is non-AI-based.
- The "UL Standard for Safety for Evaluation of Autonomous Products", **UL 4600** aims to define a comprehensive safety case for autonomous vehicles both in urban and highway use cases. It explicitly requests a logical and corresponding physical system architecture, mandates redundancy and even a layered approach by requesting observability of perception results – which effectively rules out monolithic (end-to-end AI) systems. Several of the architectures discussed here provide a sound basis to achieve compliance with the standard. Besides the direct architectural impact of UL 4600 there are many additional clauses which can be supported indirectly by a suitable ADI architecture.

- **ISO/TS 5083** is a very recent (2025) addition to the standards landscape; it gives guidance on how to develop and validate an AD system for road vehicles and how to structure the safety case. ISO/TS 5083 mandates fallback mechanisms to cope with faults, but remains architecture-agnostic. Only in the context of AI models are architectural patterns (of ISO/PAS 8800) referred to.

INTRODUCTION AND PURPOSE

THE AUTONOMOUS

The Autonomous is the global community shaping the future of safe autonomous mobility. Initiated by TTTech Auto in 2019, The Autonomous is an open platform building an ecosystem of all actors involved in the development of safe autonomous mobility. Ecosystem partners range from car manufacturers, technology suppliers and regulatory authorities to disruptors, thought leaders, academia, and government institutions. The goal of The Autonomous is to generate new knowledge and technological solutions in the field of autonomous mobility, thus accelerating the transition to market readiness and series development of safe self-driving vehicles. To achieve this, The Autonomous has put in place two strategic streams:

1. **Event Stream** – facilitates discussions and networking for leading executives and experts from the autonomous mobility ecosystem.
2. **Innovation Stream** – facilitates cooperation across the industry to work on global reference solutions for safety challenges. These reference solutions conform to relevant standards and will facilitate the adoption of safe autonomous mobility on a global scale. As part of the Innovation Stream, The Autonomous launches and facilitates Working Groups and Expert Circles in order to develop pre-competitive concepts, concrete technical solutions, best practices, and recommendations in key areas of autonomous driving – from E/E architectures and artificial intelligence to regulatory frameworks and societal acceptance.

The findings of The Autonomous Working Groups are presented yearly at The Autonomous Main Event.

WORKING GROUP SAFETY & ARCHITECTURE AND ITS SCOPE

The first initiated Working Group of the Innovation Stream of The Autonomous is the one on "Safety and Architecture": International research institutes and industry leaders come together to address the fundamental question of what the conceptual system architecture of an automated vehicle (SAE level 4 and higher) should look like, i.e., how the system's partitioning into computational streams, for instance for safety and redundancy purposes, could be performed (for further explanations, see section 1.4). The present report produced by the Working Group "Safety & Architecture" addresses this topic.

It is commonly understood and accepted that the development and implementation of a failure-free automated driving system for complex driving tasks is an extremely tough challenge. Even having been developed to the highest standards, complex HW and SW elements can exhibit malfunctions that can materialize in an arbitrary way. Still, the overall autonomous driving system needs to tolerate these and keep up operation at least for a defined time frame – i.e., needs to be fail-operational or at least fail-degraded. Regarding faults, this study very generally discusses how to achieve a dependable computational system architecture and is thus not limited to faults like the ones caused by a lack of functional safety or to functional insufficiencies due to a lack of "safety of the intended functionality". A good summary of dependability aspects that need to be considered can be found in [1].

The chosen level of conceptual representation is on the one hand sufficiently specific to be useful as a reference and on the other hand sufficiently generic to allow for different implementations. More details on this conceptual representation will be given in section 1.4. This report focuses on the computational unit between the sensors and actuators which will be called "Automated Driving Intelligence"[2] (ADI), see Figure 1. This includes sensory processing, fusion, trajectory finding and decision making, but excludes raw data sensors and the actuators. Detailed hardware and software architectures are topics for potential follow-up activities of the Working Group after this report.

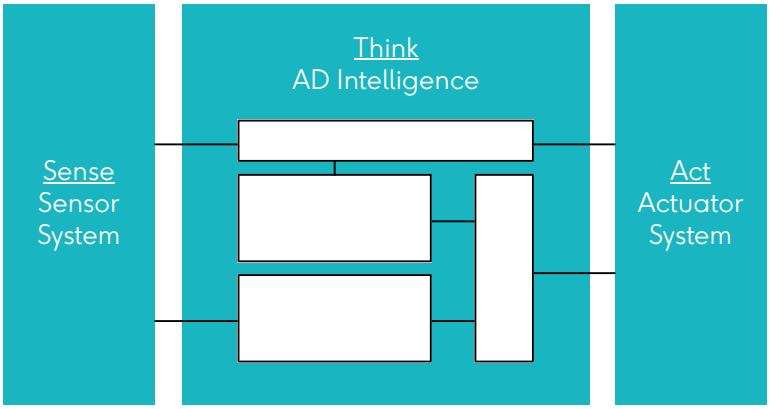


Figure 1: Automated Driving Intelligence (ADI)

We would like to make it explicitly clear that this Working Group focusses solely on the architecture aspect and does not cover the full lifecycle of an ADI. For instance, aspects like V&V or field monitoring are only considered indirectly, insofar as we point out if a given architecture fosters a modular and effective validation or rather impedes it. To this end, we also restrict ourselves to the use case of an SAE Level 4 Highway Pilot (see section 1.1 for detailed considerations), because it well reflects the typical complexity of an AD system and represents a reasonable design target for OEMs in the near future. ADI architectures for other use cases like Urban Pilots will pose similar challenges and adhere to similar design principles, even though the concrete system implementation in terms of sensor set and algorithms will be different.

[2] We use this term [50] instead of the more generic "AD System" to indicate that it excludes other systems such as raw data sensors or actuators.

PURPOSE AND STRUCTURE OF THIS DOCUMENT

Architecture and design occur on multiple different abstraction levels (see Figure 2). It lies within the responsibility of "system owners", whom we consider the intended readers of this document, to ensure a consistent design across all such levels. System owners (see also section 2.1.1) may work for OEMs, mobility companies, or their suppliers and need to make architectural decisions both on a high, abstract level and on a lower, implementation level (see also section 2.1.2).

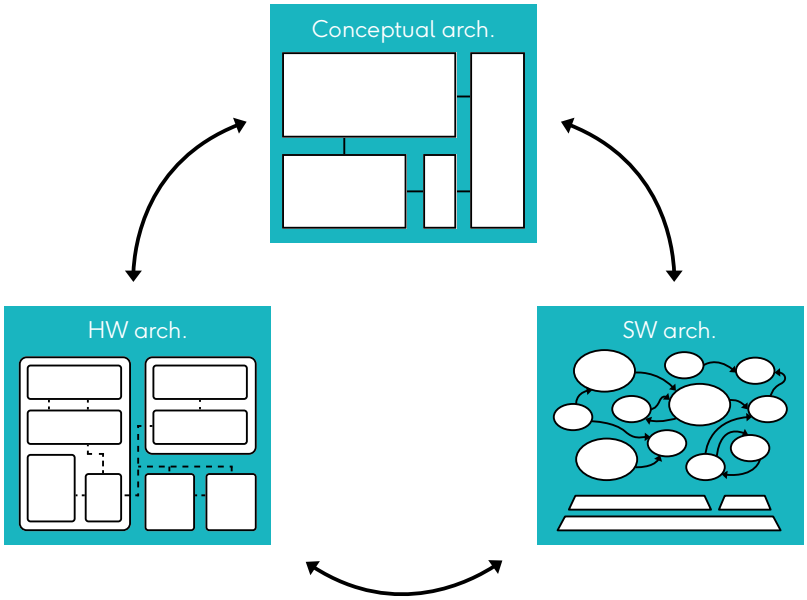


Figure 2: Ensuring consistency between architectures on different abstraction levels.

This document is intended to support system owners in making high-level architectural decisions and mapping these to low-level implementations. It aims to provide a structured analysis of high-level architectures in the Automated Driving (AD) context, as well as supportive arguments for demonstrating that crucial requirements are satisfied and important KPIs are well met.

This document is structured as follows (see also Figure 3):

- **In section 1 – Background and Premises**, we define the context in which we look at high-level system architectures.
  - We start by outlining a reference AD feature that captures the challenges regarding safety and availability. For this, we chose an assumed version of an SAE L4 Highway Pilot feature (see section 1.1).
  - The reference AD feature is assumed to be provided by an AD system. The system boundary of this "AD Intelligence" is described in section 1.2.
  - Based on these, we then derive high-level system requirements for the "AD Intelligence", with a focus on safety and availability (see section 1.3).
  - The architectural abstraction level that we consider is described in detail in section 1.4.
  - Finally, we also collect general constraints and design principles relevant to system architectures within our chosen context and on our chosen abstraction level (see section 1.5).
- **In section 2 – Architecture Evaluation Criteria**, we define evaluation criteria relevant to high-level system architectures.
  - In order to choose evaluation criteria relevant to our intended readers, we start by

describing the architectural choices they may need to make (see section 2.1).
  - Many attributes of a well-made AD system do not directly depend on the high-level architecture. We thus summarize these attributes and assume that they are covered (see section 2.2).
  - Attributes that are more closely linked to the choice of high-level system architecture are collected in sections 2.3 to 2.8 and summarized in tabular form in section 2.9. Each of these attributes is broken down into multiple evaluation criteria (and associated key questions) that we later apply in the architecture evaluation.
- **In section 3 – Candidate Architectures**, we collect and describe different high-level system architectures.
  - We start by describing the process we used to collect candidate high-level system architectures (see section 3.1). Many of these architectures will implicitly take the system safety requirements, general constraints, and design principles into account.
  - Since some of these share certain basic principles, we chose to extract these and describe their intention and mechanism in a generic way (see section 3.2).
  - The candidate conceptual system architectures are clustered in three major groups and described in a comparable way in sections 3.3, 3.4, and 3.5.
- **In section 4 – Architecture Evaluation**, we evaluate the collected conceptual system architecture candidates.
  - Our evaluation methodology is described in section 4.1.
  - We use the evaluation criteria defined earlier to make a series of general observations on each candidate architecture (see section 4.2). These are not specific to an AD use case.
  - This is then followed by considering these observations in the context of our reference AD use case (see section 4.3).
- **In section 5 – Implementation Considerations**, we provide considerations for mapping conceptual system architectures to specific HW and SW architectures.
  - Considerations for mapping a conceptual system architecture to a physical HW architecture are collected in section 5.1.
  - Considerations for mapping a conceptual system architecture to a physical SW architecture are collected in section 5.2.
  - Standards relevant to the development of safe AD systems and their impact are discussed in section 5.3. This section also outlines the process for constructing a safety argumentation.
  - Finally, considerations related to defining and evaluating sufficient independence are collected in section 5.4.

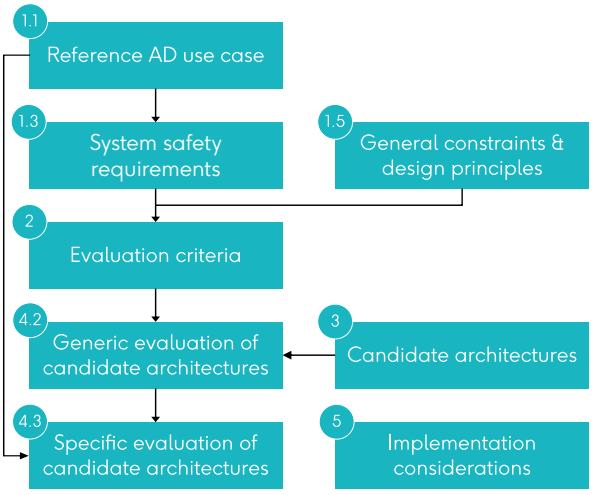


Figure 3: Structure of this document.

1 BACKGROUND AND PREMISES

The requirements, general constraints, and design principles as introduced here relate to a system of interest, referred to as the ADI (recall Figure 1), and more specifically, to the computational system architecture of the ADI. Ensuring the safety of automated driving requires a system safety perspective that takes the AD intelligence, the vehicle platform, the behavior of surrounding actors as well as the traffic environment into account, including the full set of responsibilities previously assumed by the driver. The work described in this report does not take on this entire grand challenge, but rather focuses on architectural aspects of the AD Intelligence and their contributions to safety.

With the overall goal to propose and evaluate architectures for AD systems, a main emphasis is placed on meeting appropriate functional safety requirements, with considerations of system and software complexity, and hardware reliability (referring to ISO 26262 [2]). With the introduction of advanced environmental perception and machine learning, additional safety hazards must be addressed for automated driving, as traditionally, functional safety standards have assumed that "requirements are known" and "nominal operation" with no software or electronics failures, is safe. This has led to new standards, such as ISO 21448 "Safety of Intended Functionality" (SOTIF) [3], which attempts to address these challenges. SOTIF is part of the considerations for this work regarding causes of failures and qualitative aspects of diversity (further elaborated in section 1.5.1).

For highly Automated Vehicles (AVs), the increasing complexity and risks of failures lead to open issues, including what constitutes safe road behavior and what measures are needed to assure a "positive risk balance" [4] such that an AV would at least perform better than an average driver. These are topics being treated in recent standardization work, such as ISO TS 5083—Safety of Automated Driving Systems. A positive risk balance is considered for the architectural work in terms of, for example, reliability goals. Behavioral aspects such as tactical safety [5] are beyond the scope of this work.

Cybersecurity will also be key for automated vehicles and their relation to safety, as manifested by standards like ISO/SAE 21434:2021 [6] for automotive. Cybersecurity aspects are, however, not covered extensively in this release, partly due to the chosen abstraction level. Some aspects of cybersecurity that have an indirect impact on security considerations will be covered through a few evaluation criteria, see section 2.5.

1.1 REFERENCE AD USE CASE

1.1.1 MOTIVATION

This section outlines the reference AD use case targeted by the "Safety & Architecture" Working Group of The Autonomous. This may later be supplemented by additional AD use cases in future iterations of the Working Group.

This reference AD use case **shall serve** the following purposes:

- As an input for defining reasonable assumed requirements (see section 1.3). In ISO 26262, safety-related requirements are ultimately derived from item-specific safety goals, e.g., that the system shall avoid collisions and loss of vehicle control.
- As an input for establishing what level of algorithmic complexity is required to perceive varied environments and handle different and dynamic traffic scenarios.

- As an input for defining general assumptions (see section 1.5).
- As an input for defining and quantifying dependability goals[3].

This reference AD use case **may help** with the following purposes:

- To derive a rough estimate of what computational resources are required.
- To derive a rough estimate of currently achievable failure rates for the computational resources as well as the estimated rate of hazardous behavior of the intended functionality [10][4] from software (application and infrastructure code).
- To refine requirements that are related to vehicle-level use cases and scenarios into more detailed requirements on the algorithm level, e.g., perception, activation/deactivation, degradation, or warnings[5].

Note: This reference AD use case is intended to give the reader a general understanding of what such a feature could look like. While some of these descriptions have direct relevance for architectural considerations later on (marked in **bold font** in the following reference AD use case sub-sections), many others merely serve as a background to outline the many different aspects and perspectives involved.

1.1.2 CHOICE OF REFERENCE AD USE CASE

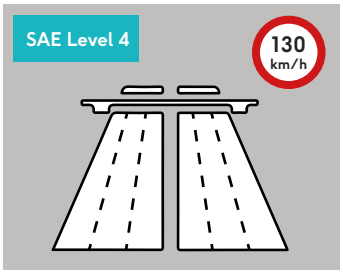
OEMs are working on a number of different AD use cases, each having different architectural implications. We have screened these on a high level according to several criteria to identify a suitable reference AD use case:

- Timeline
  The reference AD use case should (likely) become technically and commercially feasible in the near future.
- Complexity of the Operational Design Domain (ODD)
  The reference AD use case should apply to an ODD of at least medium complexity. This implies that complex algorithms and high-performance hardware are necessary.
- System availability
  The reference AD use case should have high integrity and availability requirements, i.e., require resilience against faults (fail-operational/fail-degraded) to render the vehicle at least as safe as if it was controlled by a careful, competent human driver. This implies that a non-trivial conceptual system architecture is necessary to compensate for the weaknesses of complex algorithms and powerful hardware.

[3] Dependability goals are a complex topic and can cover a wide variety of aspects. Some of these, e.g., continued operation after an MRC has been reached, go beyond the scope of this report. In this context, remote assistance can be beneficial for some AD use cases.
[4] Malfunctioning behavior can arise due to faults (e.g., bugs), due to functional insufficiencies (e.g., environmental aspects neglected in the specifications), due to operational disturbances (e.g., environmental conditions), or due to misuse. In systems involving machine learning, this may also be caused by bad or biased training data. There are some empirical estimates for the number of undiscovered bugs per line of code remaining despite using proper development processes.
[5] This may also include performance-related aspects such as timing, accuracy, and detection reliability.

Table 1: List of AD use cases under discussion (not comprehensive).

AD use case	Timeline	ODD complexity	System availability
Traffic Jam Pilot	**In series production**	**Low**	**Medium**
Highway Pilot	**High**	**Medium**	**High**
Mobility as a Service (MaaS)	**Medium**	**High**	**Medium**
Valet Parking	**High**	**Very low**	**Low**
Low-speed AD (shuttle)	**High**	**Low**	**Low**



The AD use case we consider the most suitable (see Table 1) is an **SAE Level 4 Highway Pilot (HWP)** feature.

We explicitly target the "High Automation" level / "SAE L4" (according to the classification scheme proposed by the Society of Automotive Engineers [7][6]) over SAE L3 (see also [8, 9]). This entails the following:

- The system assumes full responsibility for the Dynamic Driving Task (DDT) in all dimensions, i.e., the driver can be "hands-off".
- The system assumes full responsibility for its own supervision, i.e., the driver can be "eyes-off" and "brains-off".
- System safety will never depend on the driver to take back control, which would be both difficult to achieve [9] and would also have a pronounced detrimental effect on the "quality time" gained from an AD feature.
- The system may only request that the driver take back control within more than a few dozen seconds to allow a smooth transition to user-operated mode. If the driver does not take back control when asked to, the system needs to bring the vehicle to a minimal risk condition on its own.

If the AD system encounters a fault and/or if the driver does not respond to a request to intervene (exact time frame subject to concrete system specifications), the vehicle will perform a DDT fallback operation. We assume that this consists of the execution of a Minimal Risk Maneuver (MRM) [7] to enter a Minimal Risk Condition (MRC), e.g., pulling over to the right side / emergency lane and coming to a controlled stop or (if this is no longer feasible) coming to a controlled stop in the current lane, but excludes recovery, i.e., the AD system does not attempt to continue driving without a full reset after entering the MRC.

1.1.3 FUNCTIONALITY PROVIDED TO USER

In the following, we define an assumed version of an HWP feature, similar to proposals from different OEMs. These allow the driver of a passenger car (sedan, SUV, crossover, or similar vehicle with relatively simple vehicle dynamics) to take their eyes off the road and perform other tasks while on a highway, with the AD system performing the entire DDT (lateral and longitudinal vehicle motion control and complete Object and Event Detection and Reaction (OEDR)) and assuming full responsibility.

[6] These classification schemes are still evolving, which is why we consider a more detailed outline of the AD use case (including feature activation and deactivation) necessary.

ID	Statement
U1	**The HWP feature supports lane keeping.**
U2	**The HWP feature supports lane changes.**
U3	The HWP feature supports traffic jams (stop & go traffic and enforcement of an emergency corridor).
U4	The HWP feature can be set to continue driving on the current highway.
U5	The HWP feature can be set to go to a target highway exit.
U6	**The HWP feature supports speeds of up to 130 km/h.**
U7	The HWP feature visually presents its status (e.g., off / on / defective) as well as its world model and motion plan to the passengers.

The Operational Design Domain (ODD) of the HWP feature is outlined in more detail in appendix ODD outline of reference AD use case.

1.1.4 FEATURE ACTIVATION, DEACTIVATION, AND REQUESTS TO INTERVENE

ID	Statement
U8	We assume that "regular activation" of the HWP feature could proceed as follows: • The driver presses the "activate HWP" button. • The system checks that all conditions for its activation are fulfilled (see appendix ODD outline of reference AD use case) and indicates the result to the driver. • The system gradually offers more resistance to steering wheel and pedals.
U9	We assume that "regular system-initiated deactivation" of the HWP feature could proceed as follows: • The system visually represents the automated driving system's world model, motion plan and diagnostics to the user to simplify the (requested) control takeover for the user. • The system indicates that it is approaching a point where the conditions for activation will no longer be fulfilled (end of the mission, change of external circumstances, detected failure, etc.). • The driver presses the "acknowledge deactivation" button. • The system checks that the driver is capable of driving (attentive and hands on the steering wheel) and indicates the result to the driver. • The system gradually offers less resistance to steering wheel and pedals. • If the driver fails to resume control, the system executes an MRM when the conditions for activation are no longer fulfilled.
U10	We assume that "regular driver-initiated deactivation" of the HWP feature could proceed similarly to "regular system-initiated deactivation", but without the first two steps.
U11	We assume that "fast driver-initiated deactivation" of the HWP feature could proceed as follows: • The driver puts their hands on the steering wheel and/or feet on the pedals. • The driver overrides the resistance offered by the system. • The system indicates to the driver that it has relinquished control.

ID	Statement
U12	We assume that “driver-initiated emergency deactivation” of the HWP could proceed as follows: • The driver presses the "pull over" button. • The system indicates to the driver that it will come to a controlled stop. • The system executes an MRM.

1.1.5 DEGRADED FUNCTIONALITY

ID	Statement
U13	**The HWP feature has a nominal mode (routine/normal operation), during which it is capable of executing the mission.**
U14	**The HWP feature has a degraded mode (see also Figure 4), during which it will execute an MRM (pulling over, controlled stop, or emergency stop) [7].**
U15	**The HWP feature will enter degraded mode if any part of the AD system encounters errors seen as critical to the ADI functionality or if the ODD is violated.**
U16	After entering degraded mode (unable to continue mission), the HWP feature will not activate again without a full reboot.
U17	**In degraded mode, the HWP feature will signal the emergency (e.g., via hazard lights) and try to come to a controlled stop in (what is understood as) a safe enough location (i.e., emergency lane or right-most lane). [First choice]**[7]
U18	If this is not possible, the HWP feature will try to come to a controlled stop in the current lane of travel. [Second choice]
U19	If this is not possible, the HWP feature will try to come to an emergency stop. [Third choice]
U20	The HWP feature **does not** have a limp-home mode, during which it is capable of continuing the mission with reduced functionality (e.g., reduced speed) and/or trying to restore full functionality (e.g., partial reboot while continuing to drive).

[7] The HWP feature should be at least as safe as a competent human driver, i.e., it should aim for the first choice and only resort to the second or third choices if the circumstances require it.

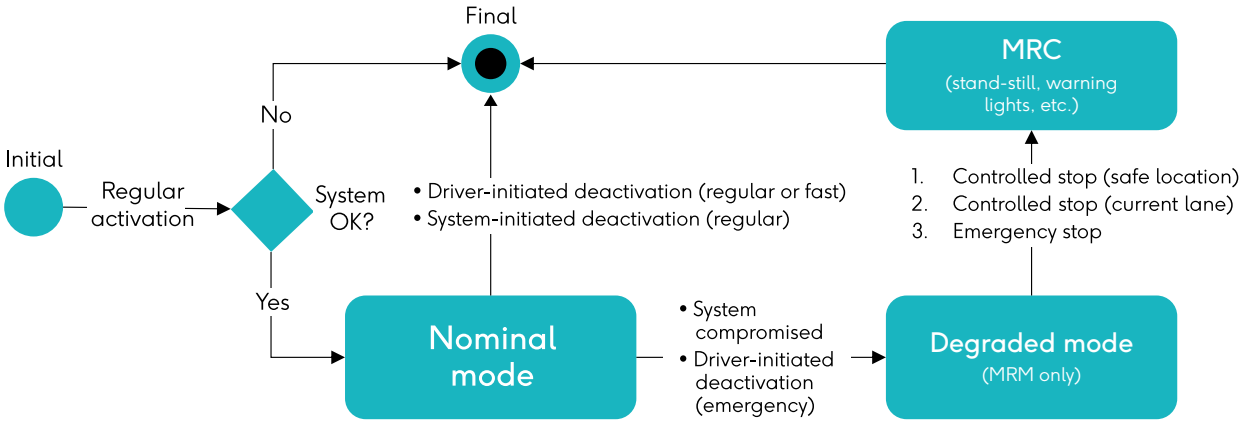


Figure 4: State diagram of different operating modes. While a minimal risk maneuver (MRM) only brings the vehicle to a stop, additional measures such as activating warning lights, etc.[75][8] are necessary to reach a minimal risk condition (MRC).

1.2 SYSTEM BOUNDARY

1.2.1 OVERVIEW

The Working Group "Safety & Architecture" primarily considers a system providing AD functionality, i.e., the Automated Driving Intelligence introduced previously (recall Figure 1). In this section, we lay out the boundary of this AD Intelligence and its interactions with other systems outside this system boundary. Due to our focus on system conceptual architectures (as opposed to detailed SW or HW architectures), we only describe the data and control flow on interfaces and omit HW-related aspects such as concrete network topologies, power supply or cooling. Figure 4 shows such a layout, providing a simplified representation including the elements that "close the loops", i.e., the physical vehicle and the human making use of the User Interface (UI) system.

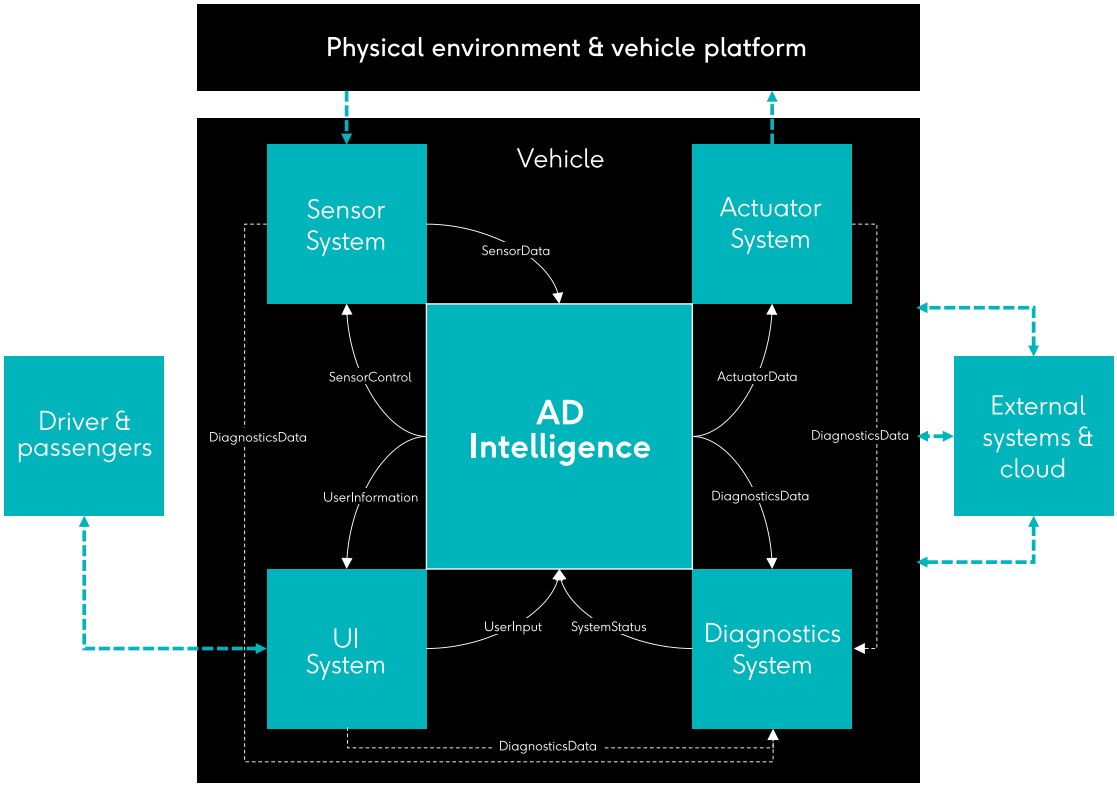


Figure 5: AD Intelligence and its interfaces to surrounding systems.

[8] UNECE R157 [75] distinguishes between procedures and maneuvers, e.g., in the context of lane changes. While maneuvers cover only the physical movement, procedures also cover accompanying actions such as indicating the physical movement to other traffic participants.

The AD Intelligence is connected to four other systems (see Figure 5), which are described in more detail in the following subsections. The main data flow is from the Sensor System to the AD Intelligence and then from the AD Intelligence to the Actuator System (receivers). The AD Intelligence's main service interface is to the Actuator System. The other service interfaces of the AD Intelligence are mainly for sensor control and diagnostics.

1.2.2 SENSOR SYSTEM

The Sensor System provides the main inputs to the AD Intelligence. It consists of a set of sensors and/or related ECUs (e.g., zonal controllers).

- The Sensor System provides measurement data from a sensor set (SensorData). This interface must be capable of real-time behavior and must be fail-operational (e.g., redundant with absence of dependent failures, encompassing common cause and cascading failures).
- The Sensor System also provides diagnostic information to the Diagnostics System (DiagnosticsData).
- The Sensor System receives calibration and control data (SensorControl).
- The sensor set must be sufficient[9] for the AD Intelligence to create a detailed environment model[10] and offer its service (nominal and degraded functionality). The sensor set comprises "outward-looking" sensors (e.g., radar, camera, lidar, or ultrasonics), "inward-looking" sensors (e.g., IMU), and digital information (e.g., V2X or HD Maps).

1.2.3 ACTUATOR SYSTEM (RECEIVERS)

The Actuator System is the consumer of the service provided by the AD Intelligence. It consists of a set of "receivers", which may be actuator control ECUs and/or smart actuators.

- The Actuator System receives a set of waypoints (ActuatorData), which describe the desired movement of the vehicle[11]. This interface must be capable of real-time behavior and must be fail-operational. The Actuator System itself must be capable of fail-operational or fail-degraded operation.
- The Actuator System also provides diagnostic information to the Diagnostics System (DiagnosticsData).
- The actuator set must be sufficient to control the vehicle even in the presence of a single fault.

1.2.4 USER INTERFACE (UI) SYSTEM

The UI System allows the user to control the AD Intelligence. Some parts of the UI are safety-critical, e.g., to prevent unintended activation/deactivation or driver monitoring.

- The UI System provides commands such as activation/deactivation requests, acceleration, steering and brake requests, destination input, or pull-over request (UserInput).
- The UI System receives requests and status information such as take-over request, or environment model for the Heads-Up Display (HUD) and presents those to the user (UserInformation).
- The UI System also provides diagnostic information to the Diagnostics System (DiagnosticsData).

[9] "Sufficient" covers multiple aspects, which go beyond the scope of the "Safety & Architecture" Working Group. Among these are that the sensor set needs to have a sufficient coverage area (detection range) for the intended functionality, needs to be able to detect all relevant objects (e.g., via employing different sensor types), and needs to be robust to faulty sensors (e.g., via redundancy).
[10] The ADI can process raw sensor data using perception and/or sensor fusion components.
[11] On the implementation level, which goes beyond the scope of this report, there are many ways to represent this intended movement, e.g., waypoints, curvature and velocity, or torque and acceleration values. The actuator control ECUs then process these in a high-frequency control loop into the corresponding actuator setpoints, e.g., the current driving the electric motor in the steering column.

1.2.5 DIAGNOSTICS SYSTEM

The Diagnostics System collects status information from all systems in the vehicle and may also contain data recording functionality (logging and/or black box). In contrast to traditional automotive diagnostics, the Diagnostics System we refer to here is focused on the AD operation and should be seen as an abstraction of existing and required (new) features. At least part of this system needs to be on board the vehicle.

- The Diagnostics System provides status information such as detected malfunctions in other systems (SystemStatus).
- The Diagnostics System receives status information from all other systems (DiagnosticsData).

1.3 SYSTEM SAFETY REQUIREMENTS

While section 1.1 describes the functionality offered by the AD Intelligence from a user perspective, we define assumed high-level safety requirements regarding the services offered by the system from a technical perspective in this section.

S1: AD Intelligence output timeliness

ID	Statement	Notes
S1	The AD Intelligence shall provide outputs to the Actuator System (receivers) in a timely manner.	"Timely manner" is here used to refer to fast enough (for the dynamics at hand) and predictably (e.g., with sufficiently low jitter, and in every cycle)

S2: AD Intelligence output availability

ID	Statement	Notes
S2	The AD Intelligence shall provide outputs to the Actuator System (receivers) in a fail-operational way to each receiver.	"In a fail-operational way" means that the AD Intelligence continues to perform its nominal function or a degraded function in the presence of any one single fault.

S3: AD Intelligence output correctness

ID	Statement	Notes
S3	The AD Intelligence shall not provide erroneous outputs to the Actuator System (receivers), implying that appropriate error detection, error handling or masking of uncritical faults should be introduced to reduce the likelihood of propagating failures (stemming from errors within the AD Intelligence).	Allowing an erroneous output to reach the actuators would lead to potential harm to the passengers or other traffic participants, e.g., due to a collision.

S4: AD Intelligence output consistency

ID	Statement	Notes
S4	The AD Intelligence shall enable the Actuator System (receivers) to ensure the consistency of executed actuator setpoints.	This applies to consistency between the setpoints executed by different and redundant actuators (like steering and braking), in particular for the case where multiple communication channels are used, possibly connecting to multiple receivers.

S5: Detection of perception-related faults and output insufficiencies

ID	Statement	Notes
S5	The AD Intelligence shall implement strategies to detect and react to perception malfunction and performance limitations due to environmental conditions or other causes related to the Sensor System.	This is not expected to be a differentiating factor between different conceptual architecture candidates. Sensor deficiencies as well as functional aspects related to perception, localization, prediction, ODD detection and planning are outside the scope of this study. Still, support for SOTIF by architectural measures is described in section 2.8 and in the candidate architectures' evaluation.

S6: AD Intelligence diagnostics

ID	Statement	Notes
S6	The AD Intelligence shall report its status to the Diagnostics System.	This is not expected to be a differentiating factor between different conceptual architecture candidates.

1.4 ABSTRACTION LEVEL

The discussion of system architecture can occur on several abstraction levels, which may be suited better or worse to the consideration of certain issues. In the following, we outline the levels relevant to the Safety & Architecture Working Group.

On a **high abstraction level**, we talk about **"conceptual architectures"**. Here, the system is composed of a small set of well-encapsulated subsystems that fail independently (so-called "Fault Containment Units" or with an additional absence of common cause failures). Each subsystem can comprise parts of a processing channel or even an entire processing channel (sensors to actuators). A point of particular interest on this abstraction level is how to achieve and manage sufficiently independent redundancy within the system.

On a **low abstraction level**, we talk about **HW and SW architectures**. Here, the system is composed of a potentially large set of HW and SW components, which may be highly particular to the specific implementation and system vendor.

The Safety & Architecture Working Group focuses on the discussion of conceptual architectures for two main reasons:

- Conceptual architectures are sufficiently non-trivial, i.e., a reference solution and industry-wide cooperation can provide benefits to system owners. Identifying the underlying principles for achieving high integrity and high availability and combining them in a transparent way leads to a better understanding.
- Conceptual architectures are sufficiently generic, i.e., a reference solution can be applicable to most system owners. Taking vendor-specific constraints, e.g., commercial considerations or integration with legacy systems, into account is shifted to the specific HW and SW implementation.

Specific HW or SW architectures are not considered. Not only would implementation-specific considerations constrain applicability and distract from the underlying principles of the conceptual system architectures, but they are also likely to quickly become obsolete. However, we will identify considerations that apply when mapping a conceptual architecture to HW and SW in order to ensure the desired system properties.

1.5 GENERAL CONSTRAINTS AND DESIGN PRINCIPLES

When coming up with conceptual system architectures intended to satisfy the system safety requirements in section 1.3, several aspects should be considered:

- There are certain basic technological limitations which constrain how very high reliability systems can be designed, built using realistic HW and SW components, and tested. Such general constraints are summarized in section 1.5.1.
- In addition, there is a set of empirical best practices that should be respected in a sound conceptual system architecture. Such design principles are summarized in section 1.5.2.

1.5.1 GENERAL CONSTRAINTS

G1: Design faults in large and complex monolithic systems

ID	Statement	Notes
G1	We assume that it is impossible to find all design faults in a large and complex monolithic SW system.	Including sample omissions and biases in machine learning training.

Rationale

- A SW system with more than ~10k lines of code will statistically contain at least one SW fault despite adequate testing [10] [11]. This does not mean that a SW system with fewer lines of code will necessarily be free from faults with adequate testing (e.g., control flows can still be complex). Heisenbug[12] type faults [12], which may appear to "alter" its behavior when attempting to investigate or reproduce it, can prove particularly hard to detect and eliminate. A typical example of a Heisenbug fault is a race condition in concurrent software.
- The ADI can be assumed to contain several subsystems that each contain more than a million lines of source code.

[12] See also https://en.wikipedia.org/wiki/Heisenbug

G2: Single-event upsets in non-redundant HW

ID	Statement	Notes
G2	We assume that it is impossible to mitigate single-event upsets (SEUs) in non-redundant HW.	While the architectural evaluation in this report will not go to a detailed hardware level, the implication is that errors due to SEUs must be considered in the system-level architecture design.

Rationale

- SEUs are caused by ionizing particles, e.g., cosmic radiation, which affects electronic devices such as processors or memory. The impact of this depends on the executed SW, but in the worst case, e.g., for brittle neural networks, even a single bitflip can lead to misclassification [13].

G3: Need for safety by design

ID	Statement	Notes
G3	We assume that it is impossible to establish the very high safety-related availability of a large monolithic system by testing and simulation alone.	The order of magnitude considered here is similar to the rate of random HW faults for ASIL D.

Rationale

- Depending on the testing assumptions, it would take hundreds of millions to hundreds of billions of miles driven to demonstrate the reliability of autonomous vehicles [14] [15]. Additional methods like simulations can only alleviate this to some extent.

G4: Specification of critical scenarios

ID	Statement	Notes
G4	We assume that it is impossible to precisely specify all critical scenarios that can be encountered within the ODD specified for automated driving.	The corresponding risks can be reduced by guidance from relevant standards such as SOTIF and UL 4600, including through ODD and field monitoring.

Rationale

- The challenge relates to the "open" environments of many ODDs and moreover to the fact that traffic behavior will change as AVs are introduced [16].
- The field of automated driving is relatively new compared to the aerospace industry. Even in the comparably "simple" environment of the sky, it took several decades of collecting and studying rare and exceptional situations to establish similarly high dependability.
- Examples for such edge cases are rare traffic participants (costumed pedestrian, vehicle with odd shape, vehicle with sky blue paint or mirror finish, etc.), rare events (complex traffic accident, confusing lost load on road, etc.), or rare environmental conditions (moon in ash cloud, etc.).

G5: Frequent switching

ID	Statement	Notes
G5	We assume that it is unsafe to frequently switch back and forth between different trajectories.	It is known from basic control theory that "bumpless transfer" requires some form of interaction between controllers involved in switching. Trajectories generated by different subsystems might not implement the same driving strategies, e.g., with respect to passing obstacles vs. braking.

Rationale

- Under certain conditions, switching back and forth between the trajectories / sets of setpoints of two subsystems may lead to unsafe behavior, e.g., when "mixing" evasive action to the left and the right and thus never moving far from the center.

G6: Checks to determine plausibility of a subsystem's output

ID	Statement	Notes
G6	We assume that it is possible to evaluate the plausibility of a proposed trajectory, based on an independent environment model.	No subsystem will have access to ground truth but is in principle able to assess the perceived correctness of the trajectory provided by another subsystem. Comment: Special focus must be put on eliminating dependent failures. With perceived correctness of the trajectory, we refer to satisfying certain safety requirements (trajectory verification).

Rationale

- There are many ways of designing safety mechanisms that cover the essential safety goals of the AD Intelligence. Instead of the trajectory, checking may also apply at the level of a set of setpoints for the Actuator System.
  - Proposed trajectories can be checked against another environment model (than the one that was used to generate it), i.e., whether certain safety goals are violated.
  - Proposed sets of setpoints can be checked against another environment model and against the corresponding proposed trajectory, i.e., whether the two are consistent.
  - A runtime environment model can be checked for violations of assumptions or of the ODD.

G7: Rate of safety incidents

ID	Statement	Notes
G7	We assume that the target rate of hazardous behavior for the AD Intelligence functionality includes the effects of random faults, as well as the potential for hazardous outcomes arising from systematic faults and functional insufficiencies.	This assumption is included to be able to reason about architectural candidates, reflecting a failure rate that does not take into account the causes of malfunctions.

Rationale

- While the failure rate targets in ISO 26262 only apply to random HW faults, the dependability goals for the AD Intelligence apply jointly to hazards arising from random and systematic HW faults, systematic SW faults, and insufficient specifications or performance limitations (SOTIF).

G8: Impact of system failure

ID	Statement	Notes
G8	We conservatively assume that all failures lead to hazardous behavior of the AD Intelligence, leading to accidents in the worst case.	This is a pessimistic assumption, but conservativeness was deliberately chosen to be on the safe side. While some failures, e.g., a collision trajectory, will be highly hazardous, other errors may not impact risk to a large extent (e.g., a slightly altered trajectory).

Rationale

- This applies only to situations where the AD Intelligence as a whole fails. The failure of single subsystems can be compensated for by the conceptual architecture.
- While active, the AD Intelligence replaces the human driver. However, according to traffic statistics [14] [17], only a small fraction of reported accidents (human-driven cars) involves fatalities (0.1-0.2%) or severe injuries. In addition, a significant fraction of minor accidents is not even reported to authorities (25-60%) [14] [18]. Most wrong decisions made by human drivers thus do not have severe consequences. We cannot make the outright assumption that the severity distribution in accidents caused by human drivers is in any way similar to those caused by an AD Intelligence; however, it is clear that not all failures of the AD intelligence will lead to fatal accidents.
- The assumption is related (complementary) to the "improvement factor" demanded of an AD Intelligence over the average human driver.

G9: Interference from other systems

ID	Statement	Notes
G9	We assume that other safety-related systems do not interfere negatively with the AD Intelligence.	Alternatively phrased, we assume an architecture which coordinates the safety-related behavior of the vehicle.

Rationale

- Inputs from other safety-related systems, e.g., Automated Emergency Braking (AEB) or similar, can be overridden on the Actuator System side while the AD Intelligence is in operation.

1.5.2 DESIGN PRINCIPLES

D1: Fault Containment Units

ID	Statement	Notes
D1	The AD Intelligence shall consist of a set of independent subsystems that each form a Fault Containment Unit (FCU). Additionally, the set must be free of common cause failures.	Special emphasis needs to be placed on avoiding dependent failures, i.e., common cause failures and cascading failures. The appropriate strategies for achieving this depend on the complexity of the subsystem.

Rationale

- In literature, variations of the definition of "Fault Containment Unit" (FCU) can be found. In the context of this report a Fault-containment unit is a subsystem with its own hardware and software, whose faults are prevented from propagating to its receivers.
  Note 1: Fault propagation is prevented by means of FCU-internal and/or external safety mechanisms, which are designed to ensure absence of cascading failures.
  Note 2: Faults with a potentially changed semantic (by an internal safety mechanism) propagate via FCU interfaces. Therefore, each interface of an FCU should be defined so that the system can react to such faults (i.e., the failure modes of the interfaces should be made known to its receivers).
- On system level it will be a responsibility of the receivers to manage the failure modes of an FCU in a safe way.
- For system-level conceptual architectures, we assume that each FCU fails independently of other FCUs. This requires an absence of common cause failures that needs to be assured by engineering measures.
- *An arbitrary failure (including Byzantine failures) of an FCU must not lead to a failure of the complete ADI; ensuring this property is a key requirement for the system-level conceptual architectures.*
- See G1: *Design faults in large and complex monolithic systems*, G2: *Single-event upsets in non-redundant HW*, and G3: *Need for safety by design*.
- To reduce the complexity of a large system, one of the simplest and most robust techniques is to allocate separable functions to subsystems that can be shown to be as independent from each other as possible [10]. Such subsystems should form FCUs, which can be verified separately.

D2: Simple and complex subsystems

ID	Statement	Notes
D2	The conceptual architecture of the AD Intelligence shall distinguish between simple subsystems (fully verifiable – preferably with formal techniques – and deterministic, e.g., due to being formally specified, having few lines of code, and avoiding algorithmic complexity) and complex subsystems.	

Rationale

- See G1: *Design faults in large and complex monolithic systems*, G3: *Need for safety by design*, and G4: *Specification of critical scenarios*.
- Simple subsystems should be developed fully to ASIL D, be fully formally specified (to preclude Byzantine faults[13] during runtime), and contain a relatively small number of lines of code (i.e., thousands, not millions).

D3: Diversity and redundancy for complex subsystems

ID	Statement	Notes
D3	The complex subsystems of the AD Intelligence shall be diverse in design where reasonable. For any shared elements, safety-related availability requirements shall be considered during development.	Groups of redundant subsystems may have similar or identical purposes. Different designs are easier to achieve in the former case, but necessary in both.

Rationale

- See G1: *Design faults in large and complex monolithic systems*, G3: *Need for safety by design*, G4: *Specification of critical scenarios*, D1: *Fault Containment Units*, and D2: *Simple and complex subsystems*.
- Complex subsystems must be assumed to exhibit Byzantine faults, i.e., inconsistent or arbitrary behavior when faulty. Due to their size and complexity, design faults and HW failures become inevitable and must be addressed by employing redundancy and design diversity.

D4: Provable correctness for simple subsystems

ID	Statement	Notes
D4	The simple subsystems of the AD Intelligence shall be sufficiently simple such that they are fully verifiable (formally specified, few lines of code).	It is assumed that the simple subsystems will be concerned with arbitration involving logic.

Rationale

- See G1: *Design faults in large and complex monolithic systems*, D1: *Fault Containment Units*, D2: *Simple and complex subsystems*.
- It is difficult to achieve replica determinism, i.e., identical behavior from two instances of the same implementation, for complex subsystems. However, this can be achievable for relatively simple decision logic, using simple, fully verifiable SW running on fault-tolerant HW.
- As stated in G1: *Design faults in large and complex monolithic systems*, SW faults should statistically be expected for a SW system with more than ~10k lines of code [10] [11].

[13] See https://en.wikipedia.org/wiki/Byzantine_fault

D5: Avoidance of emergent behavior

ID	Statement	Notes
D5	The conceptual architecture shall minimize interactions among the different subsystems.	

Rationale

- See G3: *Need for safety by design* and G4: *Specification of critical scenarios*.
- As establishing the very high dependability of a monolithic system is not feasible, it is necessary to provide evidence of each constituent subsystem's dependability separately. Such an effort is vastly facilitated if these subsystems are coherent and avoid emergent behavior when interacting with other subsystems.

D6: Transient and permanent faults

ID	Statement	Notes
D6	If the AD Intelligence detects a large number of transient faults within one of its subsystems, it shall consider this a permanent fault in this subsystem.	

Rationale

- See G6: *Checks to determine plausibility of a subsystem*.
- When transient faults occur too often, it is reasonable to consider this a permanent fault and to react appropriately (e.g., request driver to take over and/or execute an MRM).

D7: Mitigation of common cause hazards

ID	Statement	Notes
D7	The conceptual architecture shall minimize the possible propagation paths of hazards by mitigating against common cause faults and functional insufficiencies across the design pattern [19] [20].	Adaptation of the Swiss-cheese model in Figure 6 and Figure 7 is proposed to guide awareness regarding propagation of hazards through system channels. It furthermore supports the abstract design goal formulation of minimizing overlap of the holes. An example is provided in appendix Classification of Trajectory Capability.

Rationale

- The various hazards that can lead to ADS losses are a function of the ODD and use case. Functional insufficiency is a major (if not majority) contributor to hazardous ADS behavior [21]. There is an opportunity to address them at the design level too, not just within V&V efforts.
- Each channel of the architecture pattern can be characterized by capabilities. As introduced in the SaFAD whitepaper [4], these capabilities can be understood as being the fun-

damental set of system properties that are responsible for safety (nominal/degraded functionality – implemented via elements). The channel's functional complexity is indicated by the depth of the slice, and the operational domain coverage (i.e., ODD, Operational Domain, or Target Operational Domain) is indicated by its area.

- An output insufficiency is a lack of the capability that is intended to be provided by the ADI. A shared output insufficiency between channels allows triggering conditions to manifest as losses (i.e., an undesired event such as property damage, injury, or death) and is analogous to the (un-)known unsafe area in SOTIF [21]. Likewise, shared errors between channels allow fault root causes to manifest as losses.
  - Therefore, the conceptual design goal of minimizing the shared lack of capability across channels can be formulated. The conceptual dimensioning and positioning of errors and output insufficiency across channels must be well understood for mitigation efforts (such as diverse, heterogeneous implementations) to offer complementary capability.
  - As indicated by the model, it cannot be assumed that efforts to achieve diversity with respect to fault tolerance will also satisfy the diversity required to mitigate output insufficiencies. There must be an awareness that fault-tolerant implementations alone do not exclude the possibility of functional insufficiencies leading to losses.

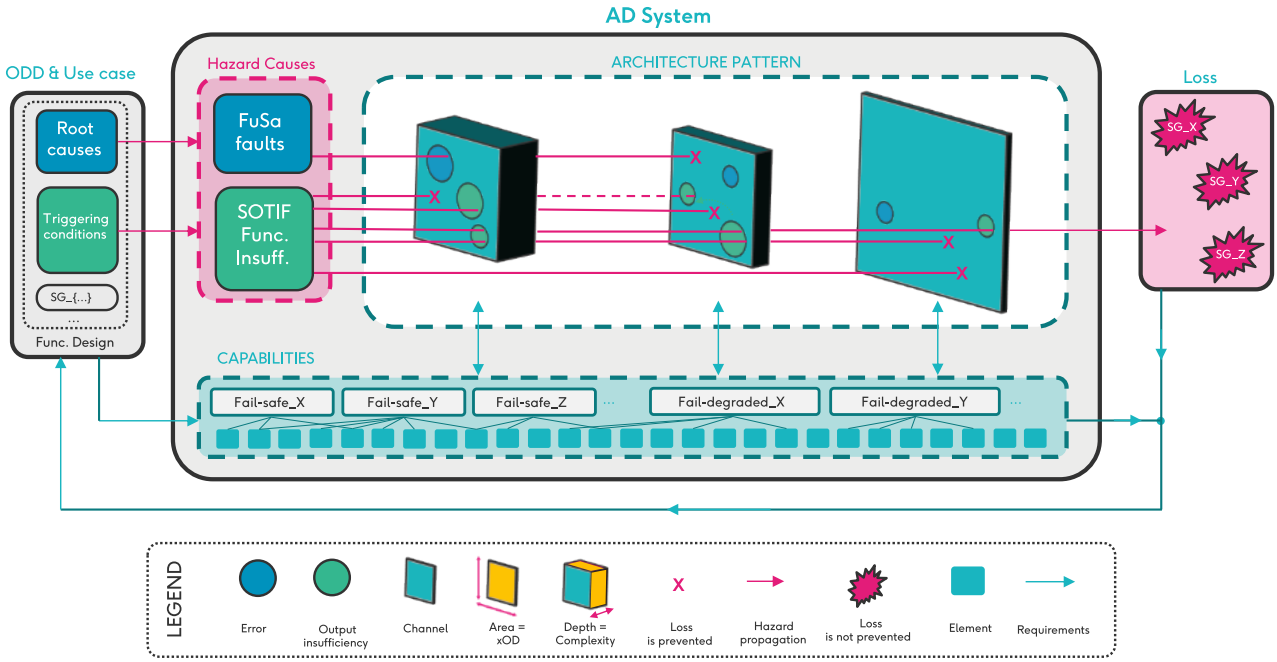


Figure 6: Conceptual architecture-level hazard propagation, as expressed via an adaptation of the Swiss-cheese model

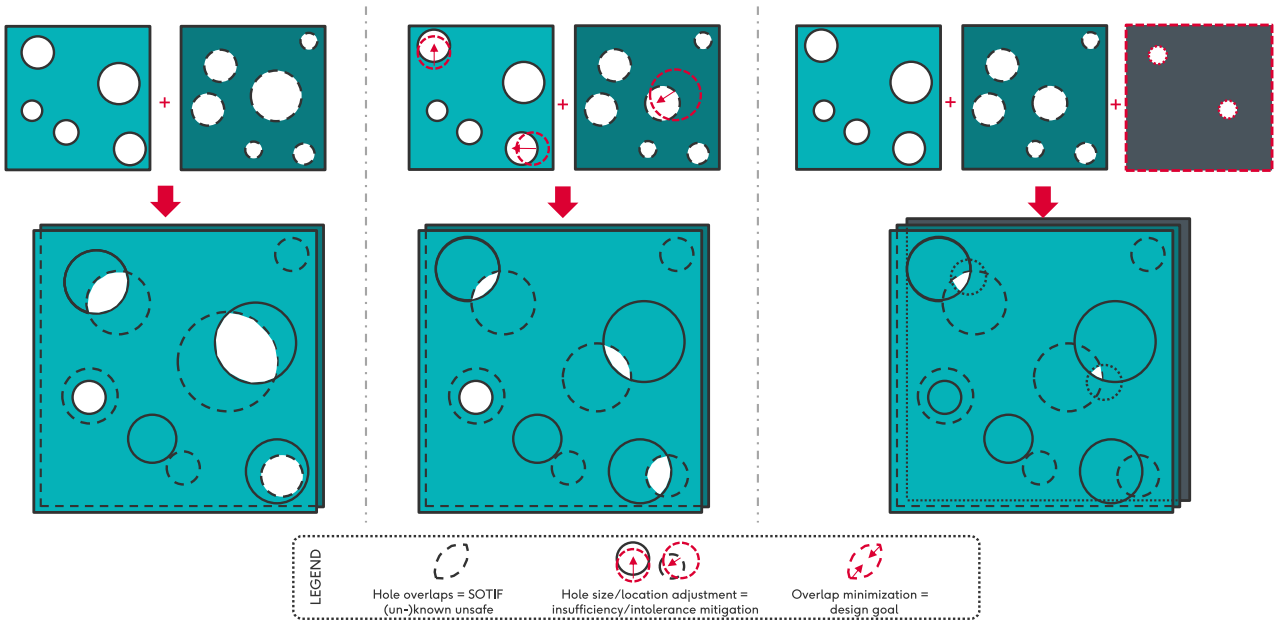


Figure 7: Bird's-eye view – Reduction of hole overlaps (errors, output insufficiency) as a design goal for channel design

2 ARCHITECTURE EVALUATION CRITERIA

2.1 ARCHITECTURAL DECISIONS AND PROCESSES

The term "architecture" can cover both commercial aspects, e.g., as business architectures, and technical aspects. In the latter, it can cover different abstraction levels, e.g., functional architectures, conceptual (or logical) architectures, down to very specific technical (or physical) HW and SW architectures.

As stated in section 1.4, the focus of the Safety & Architecture Working Group lies on the conceptual abstraction level. To make sound architectural decisions here, we first need to define a set of evaluation criteria suitable for this abstraction level. These should not exist in isolation, i.e., they should relate to relevant decisions the target reader of this document needs to make. To ensure this relevance, we first outline a persona for the assumed reader, i.e., someone responsible for making architectural decisions as part of a systems design process.

2.1.1 SYSTEM OWNER PERSONA

It lies within the responsibility of "system owners" (often system architects), whom we consider the intended readers of this document, to ensure a consistent systems design across all abstraction levels (recall Figure 2). In the following, we outline the system owner persona.

The system owner can work for an OEM, a mobility company, or for a system supplier:

- Large and/or technologically leading OEMs may try to bring most of the architectural design in-house. In this case, the system owner needs to make all architectural decisions, perform mapping between abstraction levels, and ensure consistency.
- Small and/or technologically following OEMs, as well as mobility companies, may try to buy off-the-shelf system solutions. In this case, many architectural decisions are made by the system supplier, but the system owner still needs to understand the different architecture perspectives in order to pick a suitable system solution.
- System suppliers are often focused on providing off-the-shelf HW platforms but may also extend to SW platforms and application SW solutions. In this case, the system owner may need to demonstrate to prospective customers that the offered solutions can be combined into a suitable AD system.

2.1.2 ARCHITECTURE DESIGN PROCESS AND DECISIONS

Systems design textbooks often promote a start at the top-most, user-focused level and then suggest to – step by step – become more and more detailed and specific as the design is refined. This may involve some of the following steps (see also Figure 8):

- High-level users and use cases are defined.
- Use cases are broken down into high-level system requirements[14].
- The system requirements are used to develop the high-level systems design.
- The systems design is used to derive more detailed application SW requirements.

[14] This will include both functional and safety requirements.

- The application SW requirements are used to develop the application SW design.
- The application SW design is used to derive requirements for the SW platform and HW platform.
- The platform requirements are used to develop the SW and HW platform designs.

In practice, the architecture design process is often not top-down. Several factors can contribute to this:

- An incomplete understanding of the problem space or insufficient domain knowledge may necessitate building a prototype before writing requirements.
- Emergent properties in the environment (e.g., the environment changing when exposed to the system) can also only be understood once a prototype is in the field.
- External constraints and commercial considerations (e.g., the much longer lead times and in HW development) can also shape the design before requirements are even known. In addition, legacy constraints may also come into play.

Working bottom-up can lead to situations where the design of the HW platform constrains the design of the application SW and ultimately also the conceptual architecture.

The system owner must make architectural decisions at each of the steps described above:

- What is a suitable conceptual architecture for the particular use cases?
- What is a suitable SW architecture for the particular use cases? Does it match the conceptual architecture? Is it commercially viable?
- What is a suitable HW architecture for the SW stack? Does it match the conceptual architecture? Is it commercially viable?
- Which of the available system solution offerings is suitable for the particular use cases?

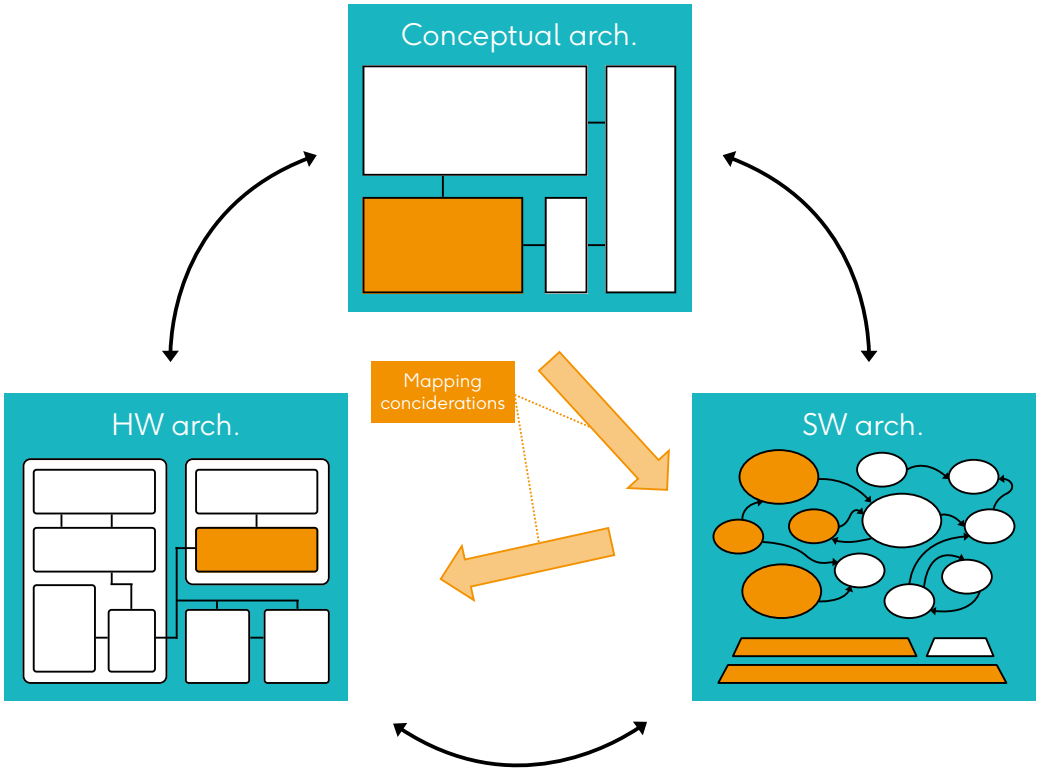


Figure 8: Idealized mapping process between different architectural abstraction levels.

2.2 GENERAL REQUIREMENTS

There are many properties that a well-designed AD Intelligence needs to have. Only some of these are suitable for differentiating different architectures on the conceptual level. Many of the attributes applicable at the physical level can be assumed to be present as long as the mapping of conceptual architecture elements to HW and SW components is done properly, and automotive development processes are followed.

For completeness, we list some of these properties in the following.

2.2.1 AUTOMOTIVE QUALITY

All components used in the AD Intelligence need to satisfy the usual automotive quality standards such as AEC-Q100 to ensure suitability for automotive use cases. This can involve robustness to shocks, high and low temperatures, etc.

2.2.2 ADHERENCE TO STANDARDS

There are several industry standards that need to be followed in the development and production of the AD Intelligence. The ones that immediately come to mind are ISO 26262 (Functional Safety) [2], ISO 21448 (Safety of the Intended Functionality) [3], ISO 21434 (Cybersecurity) [6], UL 4600 (Safety Case Assessment) [22], and SAE J3018 (Safety of On-Road Testing) [23].

2.2.3 FIELD MONITORING AND UPDATE PROCESS

Even with the most rigorous safety development process, a sufficiently complex system will almost inevitably have flaws that were underestimated or unforeseen. Therefore, it is necessary to continuously monitor vehicles in the field and analyze the collected data, e.g., to establish that assumptions made in the safety analysis continue to hold true over the lifetime of the vehicle. Flaws can be addressed by providing timely updates to minimize exposure to both safety and security vulnerabilities.

2.2.4 COMFORT AND FUNCTIONALITY

Ultimately, the AD feature needs to provide benefits to the end user. This implies that the AD function controls the vehicle in a manner that is both comfortable (e.g., low acceleration and low jerk) and beneficial to the passengers (e.g., a useful speed limit).

2.2.5 MODULARITY AND MAINTAINABILITY

Road vehicles often have an intended minimum economically viable lifetime of around 15 years[15]. Over such an extended period, it is likely that several components, particularly complex ones such as high-performance electronics, will need to be maintained or replaced. As AD systems and their components are relatively expensive, it is advantageous to design them in a modular (and thus more easily maintainable) manner.

2.2.6 PHYSICAL IMPLEMENTATION

Some attributes are specific to the physical implementation of the AD Intelligence. In general, the Electronic Control Units (ECUs) involved in the AD functionality need to be sufficiently small

[15] Of course, many vehicles continue in service much longer.

to fit inside the constrained internal space of the vehicle. They also need to have sufficiently low power consumption to not have a severe impact on the range of electric vehicles and/or cause issues with heat dissipation. Finally, the affordability of the system should also not be neglected.

2.2.7 SAFETY

The AD Intelligence must be developed to the highest applicable level as defined in ISO 26262 (i.e., ASIL D) and ISO 21448 (see also sections 5.3.2 and 5.3.3, respectively)[16]. There are two elements of safety for a fail-operational/fail-degraded system: the availability of the system, which is the probability that the system keeps operating properly when a failure occurs[17], and the safety integrity of the available outputs itself, which avoids an unreasonable risk due to their execution (e.g., collisions).

ISO 26262 uses the FIT rate (Failures in Time, i.e., per billion hours of operation) as a metric to quantify the occurrence of random HW faults. Other relevant causes for safety incidents such as systematic HW faults, systematic SW faults (bugs), and functional insufficiencies (SOTIF), are mainly addressed by prescribing safety processes[18].

To quantify the required level of safety of the system more comprehensively, we define the total rate of safety incidents (including all the underlying causes listed above) that can lead to unsafe situations (see G7: Rate of safety incidents). This rate of safety incidents for the system can be calculated through a Failure Modes, Effects, and Diagnostics Analysis (FMEDA) and a Fault Tree Analysis (FTA). Based on the reference AD use case, we propose a tentative target for the rate of safety incidents of 10-100 per billion hours of operation (10^{-8} – 10^{-7} per hour).

Different parties from industry and academia have discussed widely varying target rates [24] [25] [26]. These range from ~10^{-9} per hour (or even lower) up to ~10^{-7} per hour. These considerations are often based on the average rate of traffic accidents (or fatalities) for a particular use case (total or just highway) and an improvement factor over the average human driver.
Such a derivation roughly proceeds as follows:

- The rate of reported traffic accidents (fatal and non-fatal) can be estimated from traffic statistics [17] [14] [27]. This varies to a degree between countries and by use case, depending on the typical speed, the traffic situation complexity, and what other traffic participants are involved. The rate of fatal accidents is in the range of 1.7×10^{-7} – 5×10^{-7} per hour[19].
- The rate of reported non-fatal accidents from the same statistics is typically 100x – 1000x higher, ranging from 7.1×10^{-5} to 2×10^{-4} per hour. However, it cannot necessarily be assumed that this ratio will be similar for AD. To demonstrate a positive risk balance, we should therefore aim to build an ADI that has fewer safety incidents than humans have fatal accidents (see G8: Impact of system failure).
- The demanded improvement factor over the average human driver depends on public acceptance. Values here can range from as high as 1000x [24], which is used as a reference in aerospace, to as low as 4-5x [28], which people already find acceptable in surveys. An intermediate value of 10x – 100x may be reasonable [25].
- We also need to neglect contributions from other causes that cannot be addressed by the ADI (see Figure 9). Only causes equivalent to the cognitive tasks otherwise performed by the driver can be considered for the target rate of safety incidents.

[16] Through the use of ASIL decomposition, the ASIL for many subsystems and components can be lowered, e.g., to ASIL B(D).
[17] Loss of functionality, e.g., turning the system off in case of a malfunction, can lead to a hazard.
[18] ISO 21448 describes qualitative and quantitative criteria for the evaluation of the residual risk. A quantitative example given is the maximum number of accidents per hour.
[19] Some of these rates are given in incidents per kilometers driven. When necessary, we assume an average speed of 60 km/h for all driving and 110 km/h for highway driving to convert.

- Our tentative target of 10^{-8} – 10^{-7} per hour for the rate of safety incidents is an improvement of ~10x (1.7x - 50x) over the rate of fatal accidents and an improvement of ~1000x (710x - 20000x) over the total rate of accidents.

2.2.8 PREVENTION OF DEPENDENT FAILURES

Conceptual system architectures for the ADI depend on the assumption that at most one subsystem suffers from a fault or functional insufficiency at a particular time. Therefore, cascading failures or common cause failures (jointly known as dependent failures), which have the potential to simultaneously affect two or more subsystems of the ADI must be prevented. Section 5.4 discusses measures to address dependent failures and to evaluate how well a combination of measures prevents dependent failures.

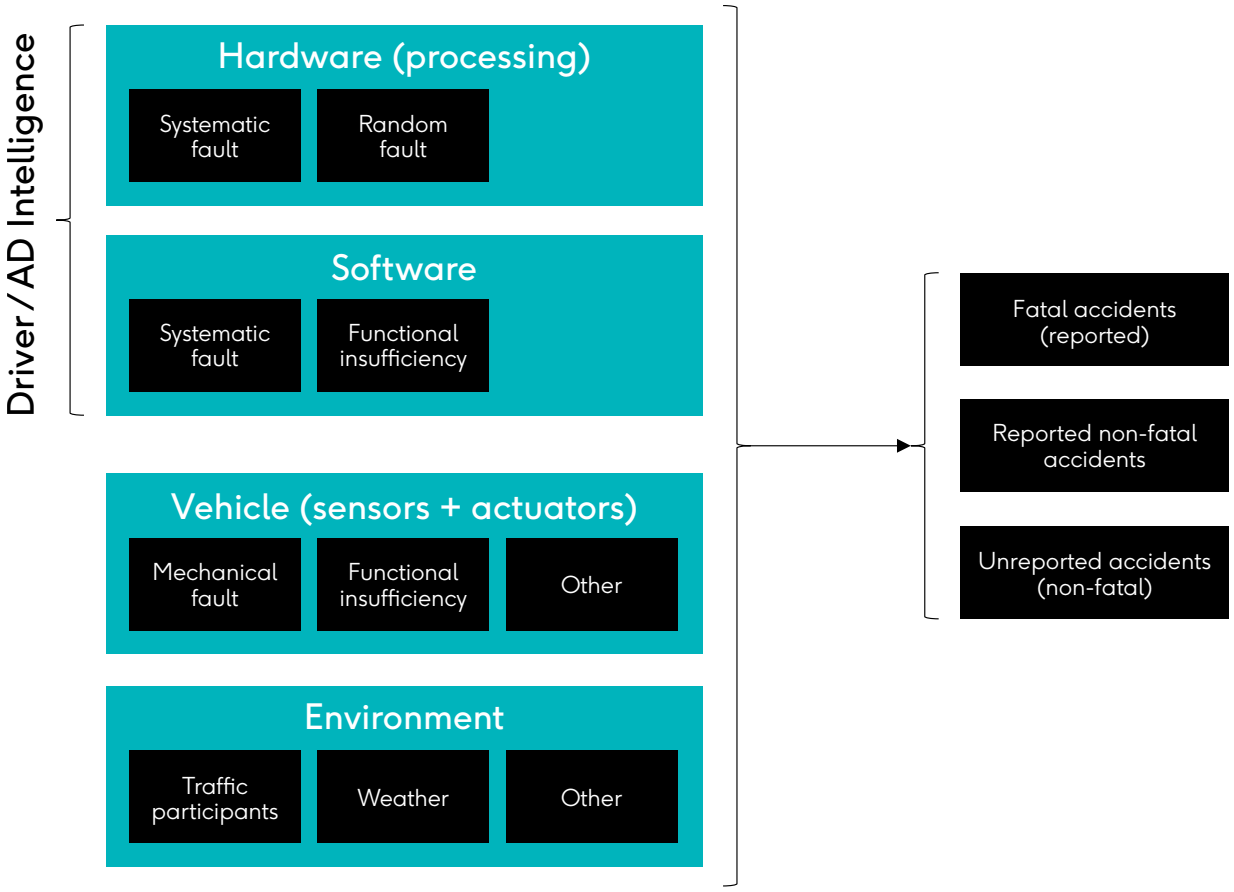


Figure 9: Coarse overview of causes contributing to accidents.

2.3 AVAILABILITY

Because there is no human driver to take over control, the availability of the system, i.e., its *readiness for correct service*, also becomes crucial. We define three evaluation criteria related to the availability attribute.

2.3.1 AVAILABILITY OF THE SYSTEM

A suitable conceptual system architecture must take safety-related availability requirements into account. This means that it is designed in such a way that no single fault can lead to the failure (or unavailability) of the entire AD Intelligence. At least some degraded functionality

needs to be available and dispatchable. Key questions related to this are:

- Does the conceptual system architecture maintain safety (correctness and availability) in the presence of any single fault or output insufficiency[20]? The output of the ADI needs to have a suitable level of capability (see appendix Classification of Trajectory Capability).
- Does the conceptual system architecture also cover all sufficiently probable dual- and multi-point faults (including common cause faults)?

If the conceptual system architecture scores badly on these questions, the system owner should consider it unsuited for AD use cases where unavailability of the system is inherently unsafe, which is most driving situations other than parking.

2.3.2 DIAGNOSTICS SCHEME

If the different subsystems have self- or cross-checking diagnostic capabilities, they can facilitate degradation schemes in the AD Intelligence (see section 2.3.3). This enables them to react dynamically to each other's condition, e.g., by proactively switching to a more cautious course of action. Key questions related to this are:

- Are the different subsystems aware of each other's condition?
- Can the different subsystems adapt based on each other's condition?

If the conceptual system architecture scores badly on these questions, the system owner should consider the increased burden on the degradation scheme.

2.3.3 DEGRADATION SCHEME

While a failure of the AD Intelligence needs to be prevented at all costs (and thus be exceptionally rare), failures of a single subsystem will be much more frequent. This can necessitate switching to a degraded mode, where the AD Intelligence executes an MRM [47][21]. If this occurs too frequently or unnecessarily (e.g., due to a transient or recoverable fault), it can adversely affect both the user experience and public safety (e.g., due to blocked public roads). Key questions related to this are:

- How noticeable is it to the end user when an error occurs in the system?
- Are different levels of degradation possible and how graceful are these?

If the conceptual system architecture scores badly on these questions, the system owner should consider the increased burden on the integrity of the implemented function, as no additional lines of defense may exist.

There may be additional practical criteria such as minimizing risk redistribution onto vulnerable population segments, but such issues are beyond the scope of this report.

[20] The definition of single-point faults in ISO 26262 only covers HW faults, whereas we also consider SW faults and functional insufficiencies.
[21] Degradation schemes can have several levels, which are progressively less safe and desirable. Schemes have been proposed to quantify such cascades and the respective acceptable probabilities of each level.

2.4 NOMINAL FUNCTIONALITY

Whenever degradation is used in the system (see also section 2.3.3), the full nominal functionality is no longer available. This has a noticeable impact on the user experience. In particular, transient faults and/or false positives should not lead to unnecessary degradation[22]. Therefore, the reliability of the AD Intelligence, i.e., its continuity of correct service, is important. We define one evaluation criterion related to the reliability attribute.

2.4.1 AVAILABILITY OF THE NOMINAL FUNCTIONALITY

A suitable conceptual system architecture is based on concepts that prevent unnecessary degradation, ensuring that the nominal functionality of the AD Intelligence is available as much as possible. This is also related to the redundancy management scheme, which is based on some kind of arbitration and ultimately decides the behavior of the system based on a limited set of inputs. Arbitration algorithms can be relatively simple, e.g., a simple silencing function in Doer/Checker, or rather complex, e.g., inexact voting algorithms. Complex arbitration algorithms may be difficult to implement in a robust way, potentially outweighing benefits from achieving a simpler conceptual architecture (see also sections 2.3.1, 2.3.3, 2.5.1, and 2.7.1). Key questions related to this are:

- Is the system sensitive to false positives[23] in one of its subsystems (e.g., a Checker "sees" an inexistent object), that make the nominal functionality unavailable (see also appendix Classification of Trajectory Capability)?
- Is the system sufficiently reliable to avoid creating nuisances like blocking public roads?
- Do the arbitration algorithms require complex and abstract decisions?
- Can these decisions be converted to pseudo-code and broken down into manageable logical statements?

If the conceptual system architecture scores badly on these questions, the system owner should consider the need for a redesign of the system or alternatively the increased burden on the quality of the primary functionality. This may require significantly higher testing efforts.

2.5 CYBERSECURITY

While the focus of the Safety & Architecture Working Group is on safety and we consider a detailed cybersecurity analysis outside our scope, some aspects of conceptual system architectures have an indirect impact on security considerations. We define two evaluation criteria related to the cybersecurity attribute.

2.5.1 INTERACTIONS BETWEEN SUBSYSTEMS

A suitable conceptual system architecture consists of several well-encapsulated subsystems that ensure that faults arising within them do not propagate to the rest of the system, i.e., Fault Containment Units (FCUs). Similar considerations apply from a security perspective, i.e., where few and well-defined interfaces between subsystems are beneficial. Key questions related to this are:

- How many communication interfaces are there between the different subsystems?
- How frequent and extensive (bandwidth) are these interactions?

[22] A system that often resorts to an MRM may well remain safe, but would be less useful.
[23] A false positive in this context indicates an incorrect activation of a safety mechanism even when it was not necessary. An example could be that a subsystem erroneously (e.g., due to a phantom object it "detects") considers another subsystem faulty and triggers an MRM.

- Are well-defined and restricted interfaces used?

If the conceptual system architecture scores badly on these questions, the system owner should consider that the security concept must more extensively consider the case where multiple subsystems are compromised simultaneously via propagation.

2.5.2 INTERACTIONS WITH EXTERNAL SYSTEMS

It is generally assumed that the AD Intelligence will need to interact with external systems, e.g., for map and traffic data, V2X, or to receive updates. Reducing the number of subsystems that are involved in this can help reduce the attack surface of the system. Key questions related to this are:

- Which subsystems need to communicate with the external systems, e.g., a backend?
- How often and for what purposes (HD maps, updates, etc.) is this communication necessary[24]? Complex subsystems providing the nominal functionality will often require access to off-board information such as HD maps, navigation data, or V2X.
- Which subsystems require updates and how often? Do they use the same update mechanisms? Complex subsystems will often require regular updates (usually OTA), while simple subsystems may only require rare updates (optionally via a different, more secure update mechanism).

If the conceptual system architecture scores badly on these questions, the system owner should consider that the security concept must more extensively consider the case where multiple subsystems are compromised simultaneously.

2.6 SCALABILITY

From the perspective of the system owner, a particular implementation of the AD Intelligence is not developed in isolation.

- Carrying over already developed systems (or components thereof) can provide huge savings in money and time.
- In addition, most OEMs aim to address different market segments and are therefore interested in multiple (and hopefully scalable) offering levels. These can range from legally required NCAP functionality to premium AD or even driverless functions (e.g., MaaS/robotaxis).

We consider both of these as parts of a scalability attribute, for which we define two evaluation criteria.

2.6.1 SCALABILITY TOWARDS HIGHER AVAILABILITY

AD features classified as SAE Level 4 and above, which are the scope of the Safety & Architecture Working Group, can vary widely, implying vastly different availability goals. For a Highway Pilot feature, remaining available for tens of seconds and coming to a controlled stop is considered sufficient. However, a fully driverless vehicle may require some limp-home functionality, i.e., continuing driving for dozens of minutes up to hours. In the ideal case, the conceptual system architecture can be scaled depending on the availability (or integrity) levels required by a particular use case. Key questions related to this are:

- Does the architecture support higher availability goals than what is necessary for the reference AD use case, e.g., for driverless use cases?

[24] This may depend on the use case, the ODD, and may also change over time.

- Which subsystems would be added to achieve this?

If the conceptual system architecture scores badly on these questions, the system owner should consider that it may be difficult to re-use it for more elaborate AD use cases at a later point in time. It may then be necessary to switch to a different conceptual system architecture.

2.6.2 SCALABILITY TOWARDS DIFFERENT OFFERING LEVELS

If multiple price segments or offering levels have to be addressed, it is highly advantageous from a cost perspective to develop all such systems jointly. Higher offering levels (offering AD features) can then be developed as extensions of lower ones (e.g., ADAS features) or vice versa. Such systems may even be similar from a functionality perspective (e.g., both performing highway driving with lane changes at up to 130 km/h) and only differ from an integrity and availability perspective (e.g., requiring supervision from an attentive driver or not). Key questions related to this are:

- Does the architecture support reusing ADAS[25] (with minor modifications) as a subsystem (role and provided functionality)?
- Which subsystems are specific to SAE L4 use cases?

If the conceptual system architecture scores badly on these questions, the system owner should consider that this may entail higher development costs.

2.7 SIMPLICITY

While we do not consider physical implementation options as part of the Safety & Architecture Working Group, some aspects of conceptual system architectures have a pronounced – though indirect – impact on this. Complex architectures with tightly coupled subsystems are generally harder to implement, validate, and verify. Ideally, architectures should be sufficiently simple such that they can be easily understood, and their subsystems can be developed and validated independently of each other. The latter is particularly important as testing a black box system to the required failure rates for AD is nigh impossible (see also G1: Design faults in large and complex monolithic systems). We define three evaluation criteria related to the simplicity attribute.

2.7.1 NUMBER, COMPLEXITY, AND PERFORMANCE OF SUBSYSTEMS

As stated before, suitable conceptual system architectures should consist of loosely coupled, cohesive subsystems (see section 2.5.1). As long as the number of subsystems and interactions is relatively low (e.g., manageable with current methodologies), emergent behavior can be more easily prevented. The development and HW costs of each subsystem depend more strongly on its internal complexity and performance requirements. This can range from essentially a smart switch with minimal logic to high-performance, AI-based subsystems for perception and planning. Key questions related to this are:

- How many subsystems exist in the system (also implying development and HW costs)?
- How complex are these subsystems (e.g., involving ML/AI-based approaches or algorithms that are hard to implement or calibrate properly, also implying SW implementation cost)?
- What are the performance requirements of these subsystems (also implying power consumption and HW cost)?

[25] It should be noted that SAE L2 systems rely on constant supervision by the driver, who must intervene immediately in case of a system failure.

If the conceptual system architecture scores badly on these questions, the system owner should be aware that the cost to implement and manufacture a corresponding physical architecture is likely to be higher.

2.7.2 REQUIRED DIVERSITY

Ensuring that multiple subsystems do not fail simultaneously due to systematic faults and/or functional insufficiencies (see also G7: Rate of safety incidents) poses a pronounced new challenge in AD. On the level of a conceptual system architecture, this generally requires asking for some level of diversity between subsystems. Exploiting asymmetries, e.g., by making use of Doer/Checker approaches, can make it easier to ensure this. Key questions related to this are:

- Between which subsystems is diversity required (also implying increased development costs)?
- Is there a large number of complex and high-performance subsystems for which not many different suppliers or approaches exist?

If the conceptual system architecture scores badly on these questions, the system owner should be aware of the additional cost and difficulty in implementing provably diverse SW.

2.7.3 COMPLEXITY OF VALIDATION

A well-known challenge in AD is how to demonstrate that the system is safe enough. To do this, testing is necessary – though not sufficient. The associated effort scales dramatically with the target failure rate of the system or subsystem. Key questions related to this are:

- Can subsystems be validated independently from each other?
- If so, does the required validation effort decrease significantly (e.g., 10^{-8} per hour/100 million hours for testing of the integrated system → 10^{-6} per hour/1 million hours for each isolated subsystem[61][26])?
- What is the complexity of ensuring the absence of correlated or common cause failures between subsystems?

If the conceptual system architecture scores badly on these questions, the system owner should be aware that testing will pose a significant challenge.

2.8 SAFETY OF THE INTENDED FUNCTIONALITY (SOTIF)

To ensure an acceptable level of dependability for AD/ADAS systems, the analysis of SOTIF aspects must be included in architectural design decisions from the beginning. Although the impact of SOTIF on the conceptual architecture of ADs has not been sufficiently examined, we propose some ideas that could help to determine whether particular architectures have the potential to better support SOTIF.

We focus on the idea that each channel must be designed to ensure a safe vehicle behavior in all the expected operational conditions depending on its functional responsibility (e.g., nominal, or fallback capabilities). Then, the safe interaction between the different architectural elements shall be ensured for system safety. For this, dedicated components supporting SOTIF-related tasks are required.

[26] The more independent the subsystems are, the easier it is to argue that there are no shared functional insufficiencies. For example, Mobileye's True Redundancy concept [61] proposes using different sensor modalities.

The analysis of the different modes of operation, ODD subsets and the intended functionality of the system may lead to the addition of sensors, components or additional channels to compensate for the functional insufficiencies.

In general, modular architectures can support SOTIF, by consisting of subsystems that complement each other functionally. This is evident for the challenges related to ODD and triggering conditions analysis, in combination with scenario-based validation. Acceptance criteria could also be defined per subsystem and in a more granular manner, reducing validation effort. Additionally, SOTIF issues are expected to require regular software updates (e.g., new traffic signs, extensions of the environmental model, safety case changes), which is facilitated by modular approaches.

2.8.1 SUPPORT TO ACCOMMODATE FUNCTIONAL INSUFFICIENCIES

As stated before, suitable conceptual system architectures can compensate not only for faults, but also functional insufficiencies. To do so, they may foresee diverse sensor modalities and may define schemes for dealing with different driving tasks within particular ODDs. Key questions related to this are:

- Is the architectural design suitable for the intended ODD and does it support an effective implementation of the vehicle's driving policy (e.g., OEDR, DDT, maneuvers, traffic rules)?
- Is the diversity of the architectural design elements sufficient to cover all the potential triggering conditions and output insufficiencies (e.g., the perception subsystem consists of diverse algorithms applying deep learning vs sensor fusion perception, avoidance of common cause false negatives when detecting/classifying objects)?
- Does the architectural design include diverse sensor modalities (e.g., vision, lidar, radar, localization) to compensate for performance limitations of the environment perception sensors?

If the conceptual system architecture scores badly on these questions, the system owner should be aware that the efforts to implement functional modifications addressing SOTIF-related risks are likely to be higher.

2.8.2 SUPPORT MANAGEMENT OF OPERATIONAL CONDITIONS

A suitable conceptual system architecture should be aware of the current ODD and react accordingly. This may also include driver monitoring. Key questions related to this are:

- Does the architecture include components to monitor adequately the ODD in different operational conditions?
- Does the architecture ensure safe usage of the driving function in all operational conditions (e.g., control takeover, activation/deactivation, degraded mode, emergency mode)?
- Does the architecture support the data collection and monitoring of safety performance indicators during field operation (e.g., to improve the set of known scenarios, data possibly collected in real-time)?

If the conceptual system architecture scores badly on these questions, the system owner should be aware that mitigating risks associated with potential functional insufficiencies and/or triggering conditions, including those that are to be uncovered during operation, will likely be difficult to achieve. This can lead to the fact that a restriction of the intended functionality must be taken into consideration more than originally planned.

2.9 TABLE OF EVALUATION CRITERIA

Figure 10 illustrates the structure of the evaluation criteria. Each attribute is split into several evaluation criteria, which in turn have several associated key questions used during the evaluation. The full set of evaluation criteria is listed in Table 2, along with related system requirements (compare section 1.3), general constraints (compare section 1.5.1), and design principles (compare section 1.5.2).

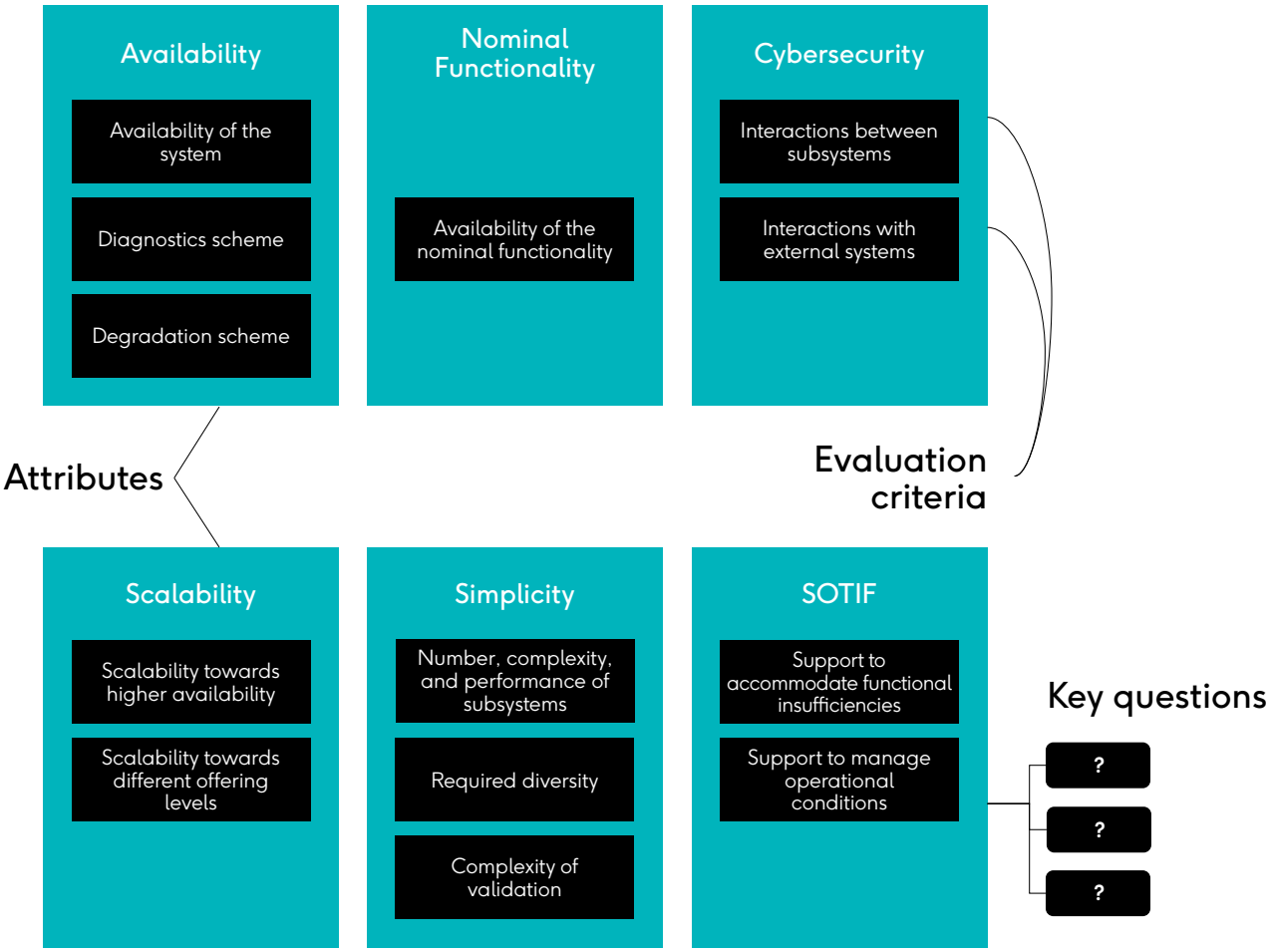


Figure 10: Structure of relevant attributes, evaluation criteria, and key questions.

Table 2: Summary of the evaluation criteria.

Attribute	Evaluation criterion	Example observations	Related
Availability	Availability of the system	• There are no obvious single-point faults in the architecture. • The architecture can deal with some multi-point faults.	S1: AD Intelligence output timeliness S2: AD Intelligence output availability S3: AD Intelligence output correctness S4: AD Intelligence output consistency G1: Design faults in large and complex monolithic systems G2: Single-event upsets in non-redundant HW G3: Need for safety by design G4: Specification of critical scenarios D1: Fault Containment Units D3: Diversity and redundancy for complex subsystems D7: Mitigation of common cause hazards
	Diagnostics scheme	• Subsystems are aware of other subsystems' status and can adapt their behavior accordingly.	S6: AD Intelligence diagnostics
	Degradation scheme	• The architecture has a defined degradation scheme. • The failure of a single subsystem does not immediately lead to an emergency reaction (e.g., MRM).	S2: AD Intelligence output availability D6: Transient and permanent faults D7: Mitigation of common cause hazards
Nominal functionality	Availability of the nominal functionality	• Frequently occurring transient faults do not lead to an emergency reaction (e.g., MRM). • The arbitration decisions can be broken down into manageable logical statements.	G5: Frequent switching D6: Transient and permanent faults D7: Mitigation of common cause hazards
Cybersecurity	Interactions between subsystems	• Subsystems only interact via well-defined interfaces.	D1: Fault Containment Units D5: Avoidance of emergent behavior
	Interactions with external systems	• Few subsystems need to communicate with external systems. • Few subsystems require frequent (e.g., OTA) updates. • Some subsystems can make use of a different, slower update mechanism (e.g., in workshop).	

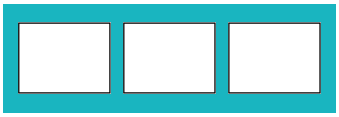
Attribute	Evaluation criterion	Example observations	Related
Scalability	Scalability towards higher availability	• The architecture can be extended by adding more subsystems to achieve higher availability or integrity.	
	Scalability towards different offering levels	• Some of the subsystems are very similar to SAE L2 ADAS systems in functionality and could be carried over with minor modifications.	
Simplicity	Number, complexity, and performance of subsystems	• The number of subsystems is small. • The number of complex subsystems is small. • The number of subsystems with high computational performance requirements is small.	D1: Fault Containment Units D2: Simple and complex subsystems D5: Avoidance of emergent behavior D7: Mitigation of common cause hazards
	Required diversity	• Diversity is required between few subsystems. • Diverse subsystems perform complementary functions (e.g., Doer/Checker). • Few complex subsystems require diversity.	S2: AD Intelligence output availability S3: AD Intelligence output correctness G6: Checks to determine plausibility of a subsystem D3: Diversity and redundancy for complex subsystems D7: Mitigation of common cause hazards
	Complexity of validation	• The different subsystems are loosely coupled and cohesive enough to be independently validated. • The target failure rate of each subsystem requires a manageable testing effort.	G1: Design faults in large and complex monolithic systems G3: Need for safety by design D4: Provable correctness for simple subsystems
Safety of the intended functionality	Support to accommodate functional insufficiencies	• The diversity of the architectural design elements (e.g., independent sensor sets) decreases the risk of unhandled output insufficiencies.	G1: Design faults in large and complex monolithic systems G4: Specification of critical scenarios G7: Rate of safety incidents G8: Impact of system failure D3: Diversity and redundancy for complex subsystems D4: Provable correctness for simple subsystems D7: Mitigation of common cause hazards
	Support to manage operational conditions	• The separation into independent channels with specific capabilities enables a high level of vehicle situational awareness.	S5: Detection of perception-related faults and output insufficiencies G4: Specification of critical scenarios D7: Mitigation of common cause hazards

3 CANDIDATE ARCHITECTURES

In this section, we collect and describe different proposed conceptual system architectures that we will evaluate in section 4, first generically, then in the context of an SAE L4 Highway Pilot as the reference use case. We first describe the process we used for collecting such candidate architectures based on publicly available sources and the experience of the Working Group members (see section 3.1). Then, we identify generic underlying principles that are shared between multiple candidate architectures (see section 3.2). Finally, we describe the structure and behavior of each candidate architecture, where we cluster them into three major types:

1. MONOLITHIC ARCHITECTURES

(see section 3.3) represent the status quo for SAE L2 ADAS and serve as the baseline for the evaluation.



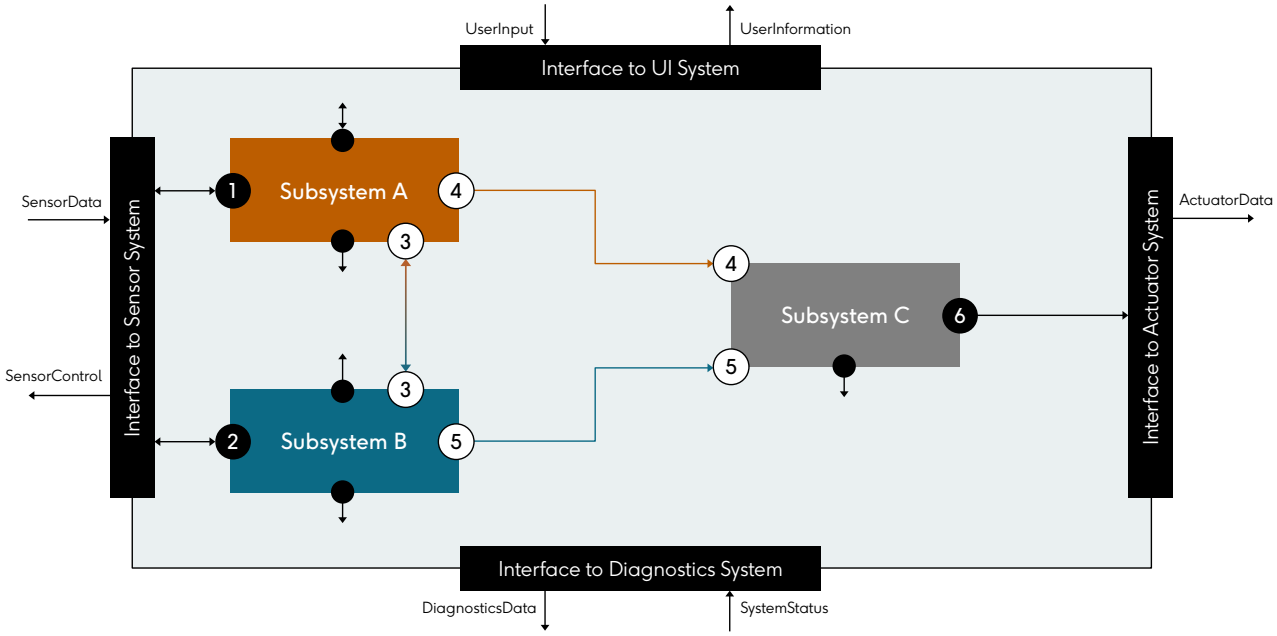
2. SYMMETRIC ARCHITECTURES

(see section 3.4) rely on multiple channels providing the same or similar functions, often with some voting mechanism (see sections 3.2.1.1 and 3.2.1.2) determining which output to use.

3. ASYMMETRIC ARCHITECTURES

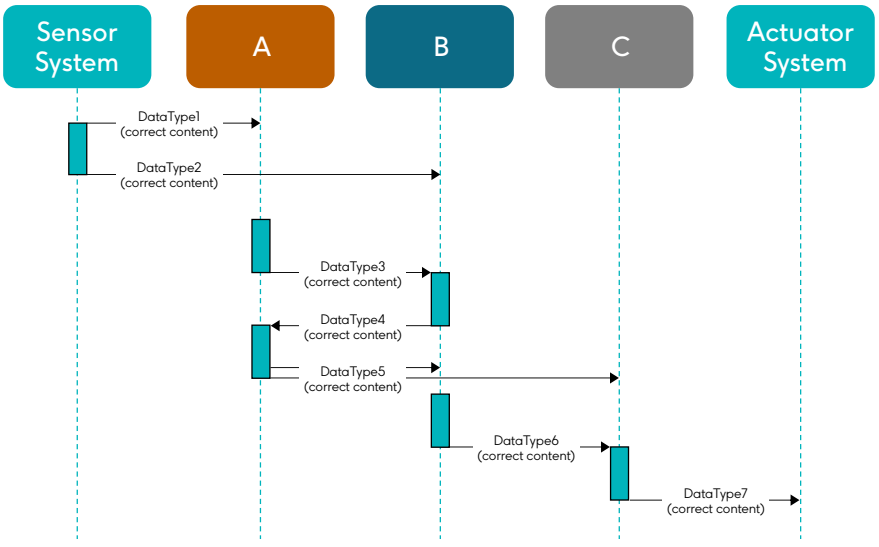
(see section 3.5) employ asymmetric decompositions to reduce the complexity of some subsystems, e.g., via Doer / Checker (see section 3.2.1.3) or Active / Hot Stand-By approaches (see section 3.2.1.5).

For each candidate architecture, relevant references and the considered variants are stated. If applicable, employed generic principles (see section 3.2) and design principles are listed. The structure of each conceptual system architecture is described via static modeling (see Figure 11), while its behavior is described via dynamic modeling (see Figure 12). The level of detail provided is intended to give an understanding of the architecture, while very specific details can be found in the respective source material.



a. Colored boxes indicate complex subsystems. Gray boxes indicate simple subsystems.
b. White circles indicate interactions between subsystems, either uni-directional or bi-directional. These are explained in more detail in the respective sequence diagram.
Shown example: a uni-directional interface (#4) from subsystem A to subsystem C.
c. Black boxes on the edge indicate interfaces of the ADI towards surrounding systems.
d. Black circles indicate interactions of a subsystem with a surrounding system. To reduce clutter, some of these are not shown in full and only point at the respective interface.
Shown example: a bi-directional interface (#2) of subsystem B with the Sensor System.

Figure 11: Explanation of the block diagrams used in the architecture description.



a. Each numbered interaction states the data type, e.g., sensor data, trajectory, or validation result. The particular value for this scenario is stated in parentheses. The sequence diagrams in the architecture description show the nominal case (without faults or functional insufficiencies).
b. Sequence diagrams show the same subsystems and interfaces as the block diagrams.

Figure 12: Explanation of the sequence diagrams used in the architecture description.

3.1 COLLECTION PROCESS

In the context of AD, a variety of architectural concepts have been proposed by both commercial and academic players. Many of these are meant to address a specific topic, but do not present a complete architecture covering all abstraction levels. Proposals regarding conceptual system architectures can be somewhat tricky:

- Proposals from commercial players are sometimes incomplete, i.e., they only describe the concepts and components on a high level, but not how they work and interact in detail.
- Proposals from academic players are sometimes challenging from a commercial perspective, i.e., they neglect the high cost of implementing textbook redundancy and diversity.

As part of the activities of the Safety & Architecture Working Group, we have screened proposed architectural concepts for their applicability to the conceptual abstraction level.

- When possible, we tried to extract generic underlying principles and cluster similar architectures.
- When necessary, we filled in missing details (from partial or very generic proposals) based on reasonable assumptions to be able to evaluate an architecture's behavior in certain scenarios and ultimately whether system requirements can be met.

3.2 OVERVIEW OF ARCHITECTURAL DESIGN PATTERNS

The conceptual system architectures we have identified share a set of underlying architectural design patterns. In general, such patterns describe well-known ("textbook") solutions to problems commonly faced by architects. These can cover SW functionality, but also system safety as described in [29] or in appendix B of IEC 61508-6:2010 [30]. Within the scope of the Safety & Architecture Working Group, we focus on the latter, i.e., architectural design patterns that help ensure the correctness and availability of a system comprised of Fault Containment Units, i.e., subsystems that can be assumed to fail independently of each other when additional care is taken to avoid common cause failures.

Well-known approaches for fail-operational systems, e.g., from safety-critical fields such as aviation, are not always directly applicable to safe autonomous driving. In this context, we have found that the safety concepts for AD Intelligence have evolved in recent years. While ensuring reliability and availability through redundancy remains the most important strategy, SAE Level 4 AD systems require different structural elements (i.e., channels and subsystems[30][27]), organized in a hierarchical or distributed way for distinct safety responsibilities. We also recognize that the focus of approaches from other industries is on functional safety, while the area of automated driving also needs a strong focus on SOTIF (see e.g., [31]).

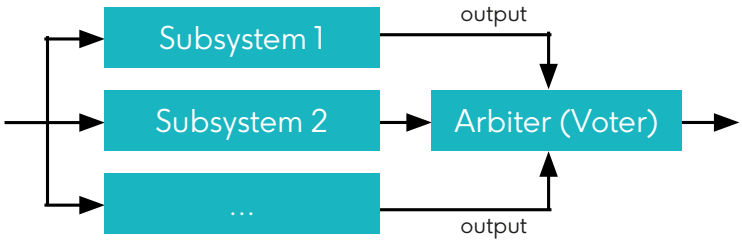
This section provides an overview of the most common redundancy-based architecture patterns. A wider discussion of safety architecture design patterns can be found in literature [29]. Although we focus on high-level safety concepts, it is noteworthy to consider that the architectural patterns are also applicable at lower levels of abstraction, depending on specific use cases. Furthermore, additional safety mechanisms, such as sensor fusion for the perception subsystem, are a well-established approach of AD systems to avoid the single failure and weaknesses of any individual sensor. The main challenge is to trade off complexity and performance while ensuring that the implemented safety mechanism covers relevant faults and functional insufficiencies. Other design patterns, such as watchdogs and sanity checks, are not explicitly mentioned as they are considered detailed implementations.

[27] A channel according to IEC 61508:2010 [30] refers to a system or subsystem, i.e., a separable building block of a system. The channel we are considering in the context of this report refers to a subsystem composed of a perception element, and a planning element, i.e., the plan stage of the so-called "sense, plan, act" model of automated driving, in other words, the end-to-end functionality from sensor input to trajectory output.

3.2.1 INTER-SUBSYSTEM PATTERNS

Inter-subsystem patterns describe how to combine multiple subsystems into an architecture. The subsystems are assumed to fail independently.

3.2.1.1 ARBITRATION AND VOTING



With two or more inputs coming from homogeneous (symmetric) or heterogeneous (asymmetric) subsystems, an element named "arbiter" acts as the decision maker that defines the output. The design of such an arbiter requires high safety integrity and low complexity.

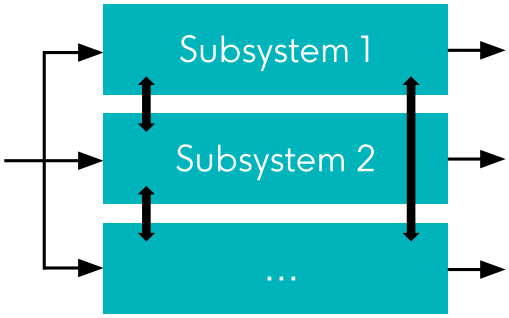
There are different implementations of arbitration depending on the type of input data, the number of available input interfaces and the voting criteria. These aspects depend on the responsibility of the arbiter and are decisive for the performance of the safety measures. Subsystem independence (e.g., diversity in generating inputs) is fundamental to managing common cause failures.

Majority voting can be considered a special case of arbitration.

Applicability:

- The original approach, i.e., binary inputs, odd number of subsystems, and majority voting, can be considered the simplest case. Such a simple arbiter could be used, for example, to determine whether to enable a separate safety subsystem.
- For more complex cases, such as continuous-valued signals (e.g., acceleration) or heterogeneous components, a more sophisticated implementation is required. This problem is comparable to inexact agreement.
- Some examples of arbitration criteria are plausibility checks, risk estimation, or scenario-based prioritization.
- An aspect to consider is the potentially high development costs for the independent subsystems.
- To ensure fail-operational arbitration, multiple arbiters may be considered.

3.2.1.2 AGREEMENT



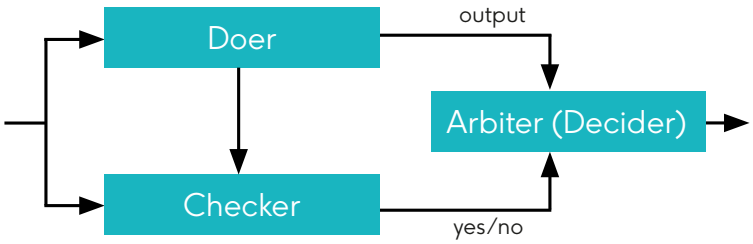
Redundant subsystems called "participants" interact to reach a decision, without an arbitration component. The agreement pattern is based on a closed loop approach and may consist of

multiple rounds of information exchange between all (available) participants.

Applicability:

- Like voting, agreement is applicable for redundant subsystems.
- Agreement mechanisms are also used for the detection and isolation of asymmetric faults.
- Like voting, there are challenges related to the implementation of agreement, especially those related to the type of input data (e.g., inexact agreement). Solutions for this can be the use of convergence algorithms, confidence rating, approximate outputs considering a given precision and allowed system accuracy.
- Agreement algorithms might not be viable when there are numerous acceptable decision candidates that might differ significantly due to the use of nondeterministic algorithms.

3.2.1.3 DOER/CHECKER (OR CONTROL/MONITOR)



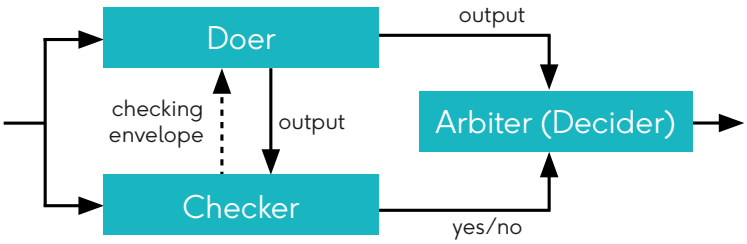
One subsystem, the "Doer", performs a function while another, the "Checker", monitors it. The checker requires higher safety integrity and lower complexity than the doer. The doer implements the nominal capabilities of the system.

There are different implementations of Doer/Checker depending on the comparison strategy and how monitoring is performed. Additional self-checks and cross-checks may be required to prevent single-point failure in the system. Subsystem independence (e.g., separate hardware) is required for high reliability and availability.

Applicability:

- This approach is useful if either a complex Doer can be monitored by a comparatively simpler Checker, or as a diversity measure (in the case of equal complexity).
- Some factors such as time lags and computational accuracy might affect the performance of the monitoring function.
- The original Doer/Checker pattern without further redundancy, as shown in the figure, is only applicable to fail-silent systems.

3.2.1.4 ENVELOPE FEEDBACK PATTERN



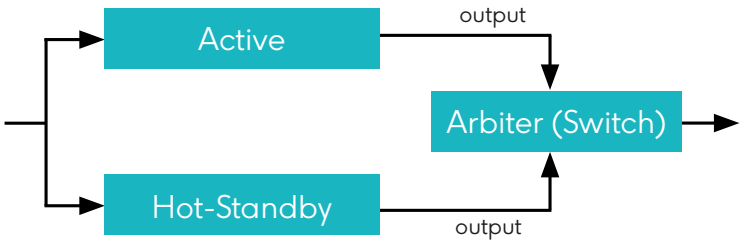
This is an extension of the Doer/Checker basic pattern. Instead of just retroactively checking

the Doer's output, the Checker provides its checking envelope to the Doer. This can allow the Doer to adjust its output such that it minimizes false positives in the Checker[28].
As an example, the Checker may detect a ghost obstacle in a neighboring lane. While the Doer may originally have created a lane-change trajectory, it may decide to instead stay in the current lane for the time being if this is considered safe by both Doer and Checker.

Applicability:

- This approach is useful to decrease the rate of false positives caused by a less capable Checker.
- A "lazy" Doer would break the independence between Doer and Checker. The implementation must prevent that the Doer creates output that relies too heavily on the checking envelope and neglects the Doer's own environment model.

3.2.1.5 ACTIVE AND HOT STAND-BY (OR DUPLEX PATTERN)



Two homogeneous or heterogeneous subsystems operate continuously in parallel while only one of them is active at any given time. A fault detection mechanism acts as a switch between the subsystems. The fault detection mechanism requires redundancy (e.g., cross-checking) to avoid single-point failures or more complicated solutions involving "default bypasses" (e.g., default to Active) in case of loss of the Arbiter.

The component acting as comparator and fault detector shall be designed carefully to ensure high fault coverage. Identifying faulty subsystems at runtime requires high fault coverage for the diagnostic mechanisms [32].

Related alternatives are:

- Warm Stand-By: the reserve subsystem runs in idle state, and
- Cold Stand-By: the reserve subsystem is normally off.

Applicability:

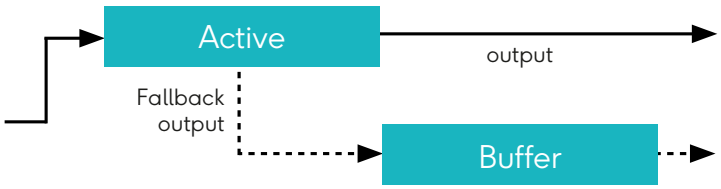
- This approach is suitable for functionalities with strict time constraints.
- The decision between hot, warm, or cold redundancy depends on the required safety level, response time, and power consumption.

3.2.2 INTRA-SUBSYSTEM PATTERNS

Intra-subsystem patterns describe how to improve the behavior of a single subsystem within an architecture. Such patterns can be considered supplemental and do not have a direct impact on the overall system architecture.

[28] This has some similarities with feed-forward mechanisms. The 'disturbance' in this case comes from the Decider silencing the Doer's output per request of the Checker.

3.2.2.1 BUFFERING PATTERN



In this pattern, a single fail-silent subsystem either produces safe output or none. The nominal output or a dedicated fallback output is buffered in a different element and is used for a limited amount of time if the subsystem produces no output.

As an example, the subsystem's output can be a trajectory to control the vehicle. The nominal output will typically be adapted multiple times per second to the environment. A fallback output could be a fully pre-planned MRM trajectory, which does not rely on any further sensor input.

Applicability:

- This approach is useful as a last resort to ensure the system's availability in case of multiple simultaneous subsystem failures.
- For low-speed AD use cases, it may be sufficient to rely on the buffered output without dynamic adaptations. For high-speed AD uses cases, where the time to reach a minimal risk condition can be much longer, this should only be considered a mitigation measure.
- The buffered output must be validated so that it can be considered safe with respect to the current environment (road, traffic participants, ego vehicle state, etc.).
- The buffering element should be connected to the Actuator System with as few intermediary elements as possible.

3.3 MONOLITHIC ARCHITECTURES

3.3.1 SINGLE-CHANNEL ARCHITECTURE

This section describes the simplest possible conceptual system architecture, consisting of a single, monolithic subsystem. Such architectures are typical for ADAS.

3.3.1.1 UNDERLYING CONCEPTS AND DESIGN PRINCIPLES

Centralizing all ADAS functionality into a single domain controller was a significant simplification compared to earlier architectures where different functions were provided by different controllers.

Due to its monolithic nature, this architecture does not employ any inter-subsystem architectural design patterns.

3.3.1.2 STRUCTURAL DESCRIPTION

The proposed architecture is monolithic, i.e., it only consists of a single FCU with interfaces identical to the external interfaces of the AD Intelligence (see Figure 13).

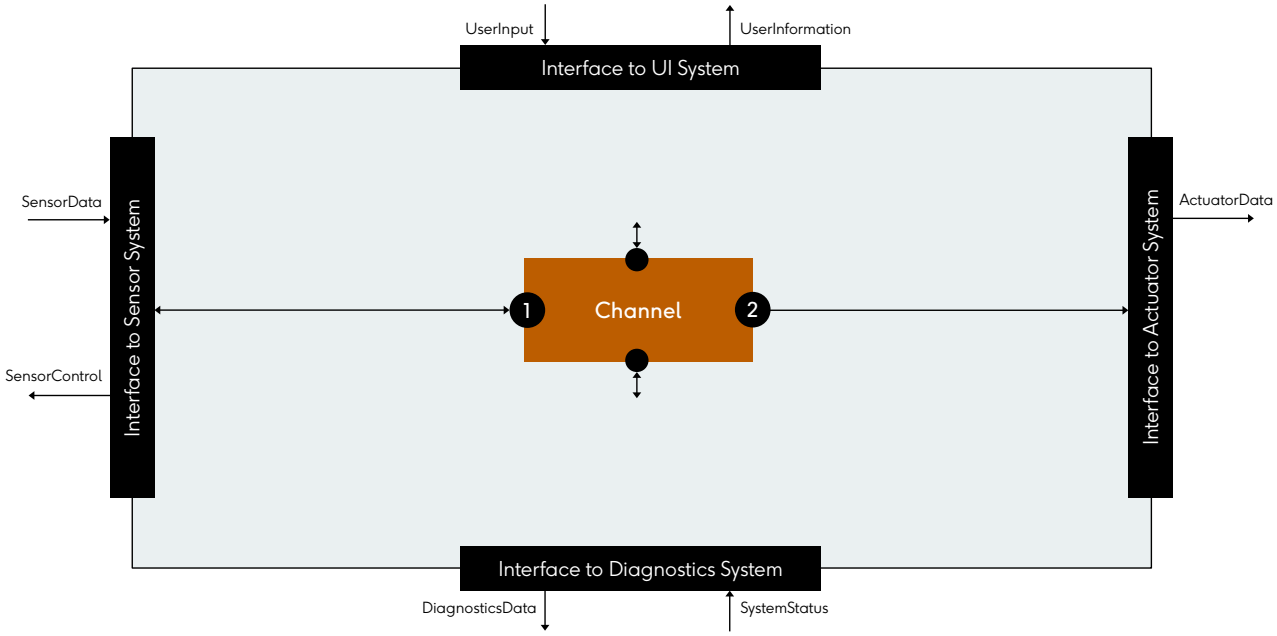


Figure 13: Block diagram of the Single-Channel architecture.

3.3.1.3 BEHAVIORAL DESCRIPTION

Due to its monolithic nature, the behavior of the system is straightforward (see Table 3, Figure 14, and Figure 15).

Table 3: Behavioral description of the subsystems in the Single-Channel architecture.

Subsystem	Behavior
Channel	• Receive SensorData from Sensor System (interface #1). • Generate ActuatorData (nominal trajectory) and send it to Actuator System (interface #2). • If an internal fault is detected (or the SystemStatus is not OK), remain silent.

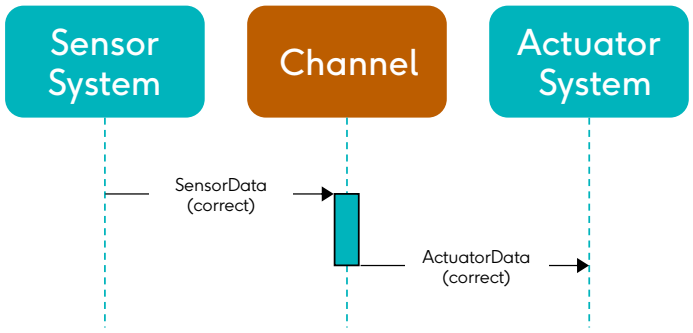


Figure 14: Sequence diagram of the Single-Channel architecture. The nominal case without faults or functional insufficiencies is shown.

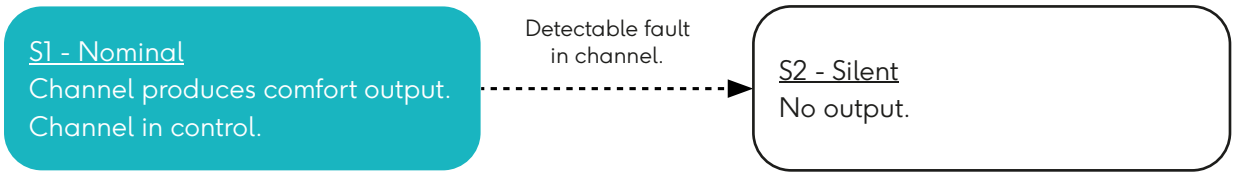


Figure 15: State diagram of the Single-Channel architecture.

3.4 SYMMETRIC ARCHITECTURES

3.4.1 MAJORITY VOTING ARCHITECTURE

This section follows the specific variant of majority voting called "triple modular redundancy (TMR)" with a particular focus on the redundancy aspects of the architecture. As described in [33], this type of redundancy is commonly used in very high reliability systems such as those used in aerospace. To the authors' knowledge, a strict application of this architecture in the AD domain has not yet been officially published. Still, we include it in the report, as the voting paradigm is an obvious and tempting approach for AD systems, and its properties therefore deserve a closer look.

3.4.1.1 UNDERLYING CONCEPTS AND DESIGN PRINCIPLES

Triple modular redundancy is a specific implementation of "N-Modular Redundancy" where three identical channels produce results that are sent to a "Voter". The Voter is responsible for looking at the results from the three channels and deciding which result is likely to be correct.

- The Voter operates based on the assumption that common mode failures are much less likely than single-event errors. This implies that the majority is correct. Therefore, if the voter observes two identical results and one dissimilar result, it will assume that the dissimilar result came about through a failure and the two identical results are in fact correct.
- Different numbers of Voters are possible. The original version of TMR [34] only employs a single Voter, which may pose a single point of failure from the availability perspective; other versions of TMR [35] propose three redundant voters. Other combinations are also possible if some arbitration is possible on the receiver side, e.g., an order of preference if the (single) Voter is silent or if the (multiple) Voters disagree. The variant described in this section employs two Voters, with the Actuator System preferring one over the other.
- In the strictest sense, only identical results can form such a majority. This can be relaxed to some extent to "sufficiently similar" results via inexact voting approaches.

3.4.1.2 STRUCTURAL DESCRIPTION

The proposed conceptual architecture consists of three complex and two simple subsystems:

- Channels 1-3 have similar roles: they compute results. For exact voting, these channels would most likely need to be implemented in an identical way unless the provided functionality is very simple and straightforward. However, even then replica indeterminism may cause issues due to minute differences in timing. For inexact voting, some degree of diversity may be allowed. We assume that this option is chosen to prevent common cause failures.
- Voters 1-2 have similar roles: they decide the correct result by comparing the outputs of the channels. While simple versions of TMR only have a single Voter, redundancy is necessary to ensure availability. If no majority can be found, the Voter can either remain silent, prefer one of the channels, or resort to a pre-planned (buffered) MRM trajectory.

The interfaces to the three channels are identical, and the output of each channel is fed to the two Voters. The results of the Voters are then fed to the Actuator System of the vehicle.

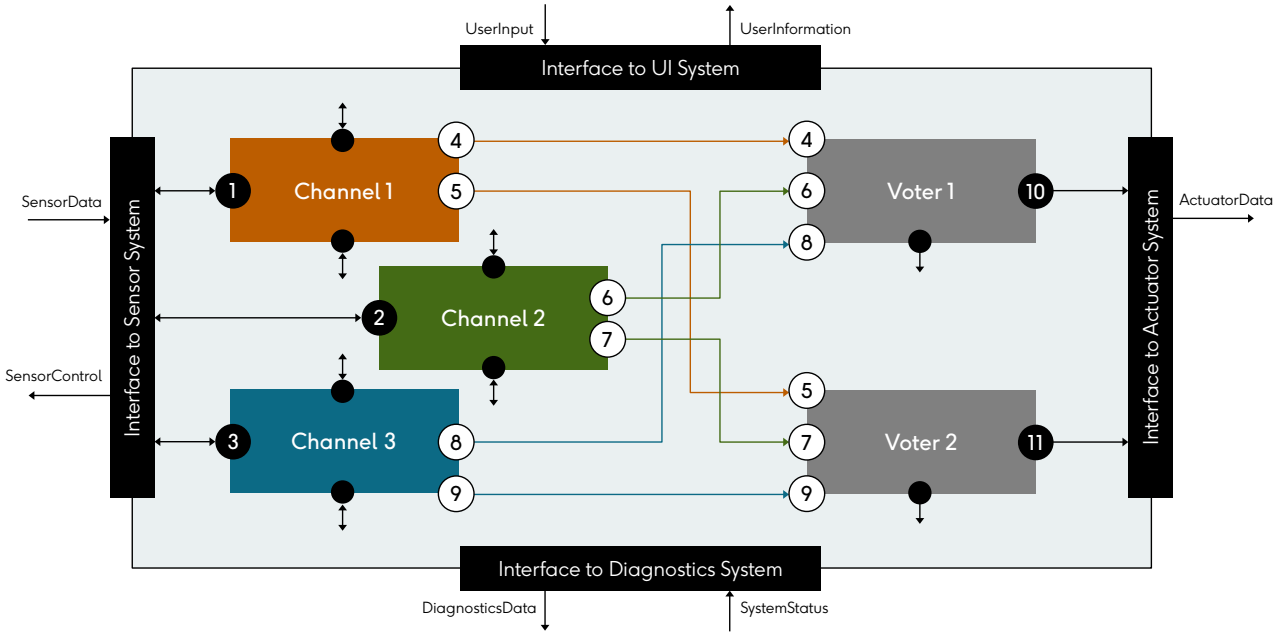


Figure 16: Block diagram of the Majority Voting architecture.

3.4.1.3 BEHAVIORAL DESCRIPTION

Table 4 describes the behavior of each of the subsystems in more detail. The interaction between subsystems is described in the sequence diagram in Figure 17.

Table 4: Behavioral description of the subsystems in the Majority Voting architecture.

Subsystem	Behavior
Channel [1-3]	• Receive SensorData from Sensor System (interface #1/#2/#3). • Generate ActuatorData (nominal trajectory and optionally pre-computed MRM) and send it to Voters (interfaces #4&5/#6&7/#8&9).
Voter [1-2]	• Receive ActuatorData from channels (interfaces #4&6&8/#5&7&9). • Find majority in ActuatorData via inexact voting and send majority output to Actuator System (interface #10/#11).

In the proposed architecture, each channel contains all functions required to implement the entire AD functionality. This includes taking the sensors of the vehicle as input, generating an environment model, trajectories, and actuator setpoints, checking whether the vehicle is operating inside the ODD, etc.

Like the Single-Channel architecture, each channel would:

- Process received sensor data into a consistent environment model.
- Periodically generate trajectories and corresponding actuator setpoints.
- Send these setpoints to the Voter (or ultimately to the Actuator System).
- Remain silent if an internal fault within the channel is detected.

The Voters base their decision on the following scheme:

- Only uncorrupted received results are considered.
- Each received result is compared to every other received result. To implement inexact vo-

ting, a degree of similarity between results needs to be defined and calculated, which may not be straight-forward. If such a majority is found, it is forwarded to the Actuator System. Otherwise, the Voter can (depending on implementation):
    - Remain silent.
    - Prefer one of the remaining channels.
    - Resort to a pre-planned (buffered) MRM trajectory or blind braking.
- If one of the channels is found to be permanently faulty, it can also make sense to send the remaining channels into a degraded mode.
- The full behavior of the Voter is summarized in Table 5, which lists all relevant permutations.

Table 5: Voter behavior in the Majority Voting architecture. Only the relevant permutations are listed, i.e., there is no preference of A over B or C.

Channel 1 output	Channel 2 output	Channel 3 output	Voter decision
Result A	Result B~A	Result C~A	Result A
Result A	Result B~A	Result C	Result A
Result A	Result B~A	None	Result A
Result A	Result B	Result C	Silent (fault) / prefer result A / pre-computed MRM
Result A	Result B	None	Silent (fault) / prefer result A / pre-computed MRM
Result A	None	None	Result A
None	None	None	Silent (fault) / pre-computed MRM

From this table we can observe the simplicity in the Voter's design, i.e., the number of possible combinations the Voter must consider is small. We also see some of the Majority Voting architecture's major flaws:

- Simple voting patterns rely on exact voting. This works well if failure modes are unique and common mode failures are unlikely.
- More advanced voting patterns rely on inexact voting. In our case, diversity is necessary to ensure sufficient independence and prevent common mode failures. Inexact voting relies on the assumption that diverse implementations will produce relatively similar results. While simple problems may have a single "best / correct" solution, more complicated or even complex problems may have multiple "good" solutions, which can differ fundamentally (e.g., evading left, evading right, or coming to a stop in front of an obstacle)[29].
- It is therefore possible for the Voter to reach a state where no decision can be made if each of the channels produces a different result[25][89][30]. This would need to be treated as a fault scenario and a predetermined recovery action would take place.

[29] It should be noted that this may be more pronounced in the automotive domain (busy road) than in the aerospace domain (empty sky). However, voting patterns can still be useful on a lower level if decisions are binary, e.g., voting on the existence of an object based on sensor modalities.
[30] Quote from : Provably correct Decision System: Whenever two independent redundant subsystems are involved in a decision in a complex environment there is the possibility of two different correct outcomes. The introduction of a third subsystem will only mask a single fault if the involved systems are replica determinate .

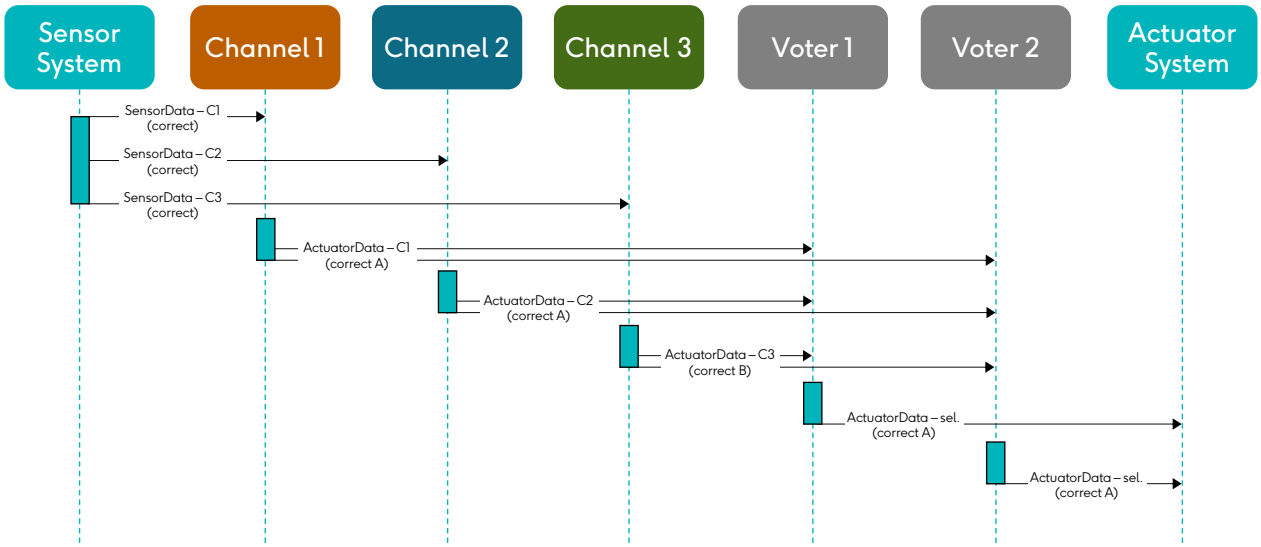


Figure 17: Sequence diagram of the Majority Voting architecture. The nominal case without faults or functional insufficiencies is shown.

3.4.2 CROSS-CHECKING PAIR ARCHITECTURE

3.4.2.1 UNDERLYING CONCEPTS AND DESIGN PRINCIPLES

The Cross-Checking Pair (CCP) architecture is based on a combination of the Doer / Checker basic pattern and the Buffering supplementary pattern. It is related to the 1oo2D architecture described in appendix B of IEC 61508-6:2010 [30] and also loosely resembles a reduced variant of the Daruma architecture (see section 3.4.3) where disagreement between the two channels is resolved via buffering. This architecture is inspired by the straightforward approach of constructing a fail-operational system by "making two channels that check each other" and related, often more HW-centered, approaches [36].

3.4.2.2 STRUCTURAL DESCRIPTION

The Cross-Checking Pair architecture (see Figure 18) consists of two complex and two simple subsystems:

- **Channels 1-2 have similar roles:** they produce actuator commands and also validate the actuator commands produced by the other channel against their own environment model. To ensure sufficient independence, the channels must be diverse. The outputs they produce can vary in their focus, i.e., at least one must produce nominal (comfort) actuator commands, but the other may produce degraded or MRM actuator commands. At least one channel must also produce a pre-planned MRM that can be buffered.
- **Selectors 1-2 have the same role:** they collect actuator commands and validation results from the channels and select the best-ranked actuator commands that are considered OK by both channels. If the two channels cannot find a trajectory they both consider OK, the Selectors resort to the buffered, pre-planned MRM.

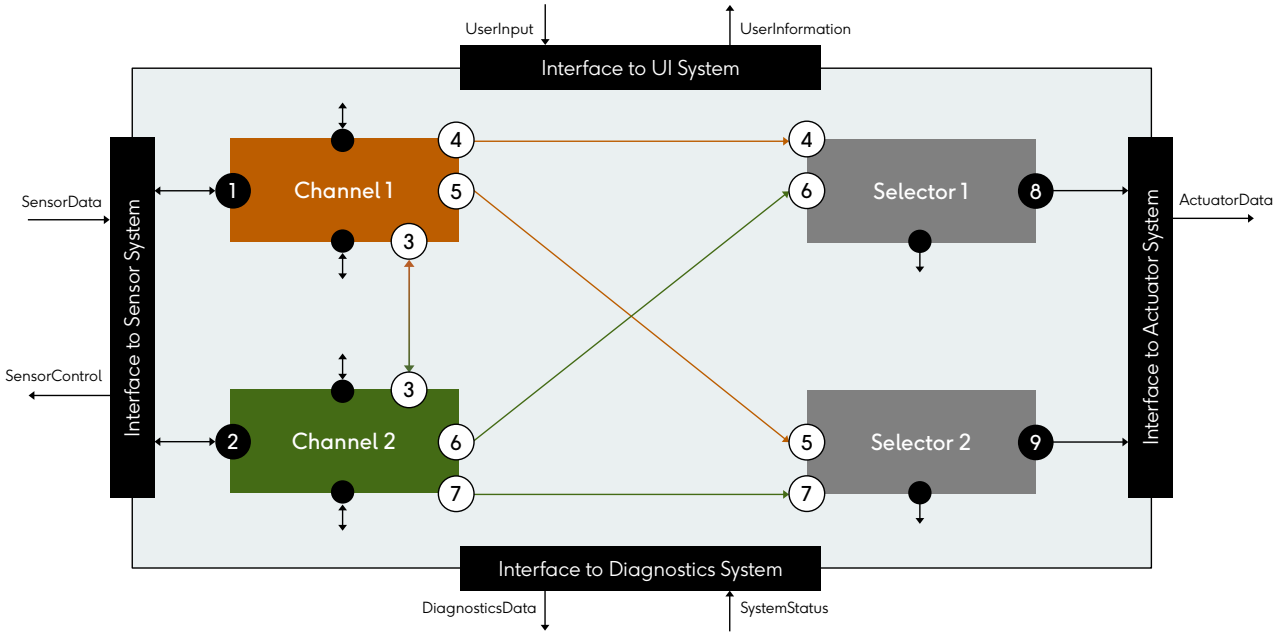


Figure 18: Block diagram of the Cross-Checking Pair architecture.

3.4.2.3 BEHAVIORAL DESCRIPTION

Table 6 describes the behavior of each of the subsystems in more detail. The interaction between subsystems is described in the sequence diagram in Figure 19.

Table 6: Behavioral description of the subsystems in the Cross-Checking Pair architecture.

Subsystem	Behavior
Channel [1-2]	• Receive SensorData from Sensor System and generate environment model (interface #1/#2). • Generate ActuatorData (for nominal trajectory and pre-planned MRM) and send it to other channel (interface #3) and Selectors (interfaces #4&5/#6&7). • Receive ActuatorData from the other channel (interface #3) and validate it against own environment model. • Send ValidationResults (including hash of validated ActuatorData) to Selectors (interfaces #4&5/#6&7).
Selector [1-2]	• Receive ActuatorData from channels (interfaces #4&6/#5&7). • Receive ValidationResults from channels (interfaces #4&6/#5&7). • Rank ActuatorData based on ValidationResults and send highest-ranked (separately for nominal trajectory and pre-planned MRM) to Actuator System (interface #8/#9).

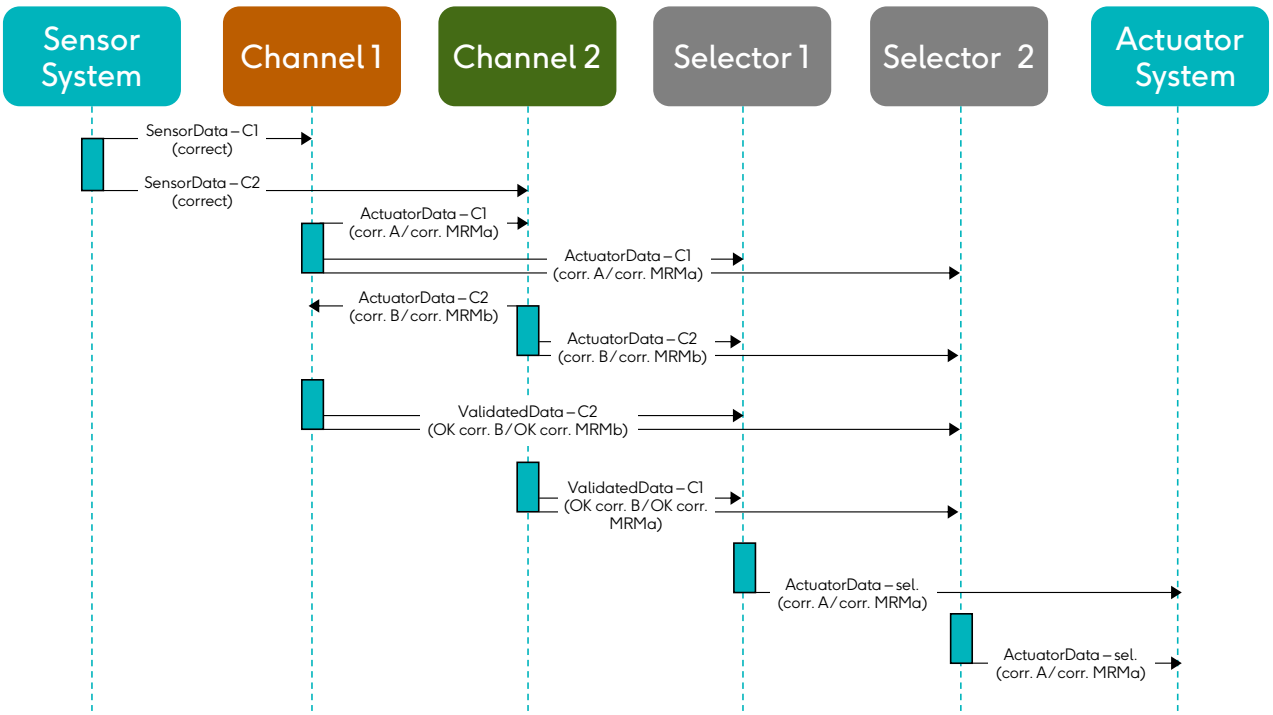


Figure 19: Sequence diagram of the Cross-Checking Pair architecture. The nominal case without faults or functional insufficiencies is shown.

The channels both generate two separate sets of actuator data:

- The nominal trajectory is necessary for performing the use case. It needs to provide comfort and react dynamically to other traffic participants. Therefore, the related actuator data is only valid for a limited amount of time (at most a few seconds). If the other channel is faulty (according to the SystemStatus), the nominal trajectory becomes degraded and is restricted to pulling over and coming to a controlled stop. This is preferable over a pre-planned MRM.
- The pre-planned MRM trajectory is only used if the Selectors cannot find a valid nominal trajectory and therefore silence the ADI. It only needs to come to a controlled stop, preferably in the right-most or emergency lane. The related actuator data needs to cover the entire maneuver (potentially dozens of seconds). A validated pre-planned MRM is buffered in the Actuator System[31] and executed when the ADI no longer provides output.

The validation is based on the environment model of the respective channel, providing independence from the environment model of the other channel that generated the actuator data. Several properties can be considered separately in the ranking:

- The correctness of the actuator data is a crucial property that must be validated, i.e., that the underlying trajectory is free from collisions, can be physically executed, etc. A binary value can be used to express this, e.g., OK or NOK.
- Further relevant properties are the comfort, speed, and efficiency of the actuator data. These can be used for ranking trajectories and can be expressed as a continuous score.

The Selectors base their decision on the following scheme (see Figure 20; this is done separately for nominal and pre-planned MRM trajectories):

- If both nominal trajectories are considered OK by both channels, the one with the better score is forwarded to the Actuator System.

[31] The buffering can occur in different locations. Ideally, this should be independent of the ADI to prevent common causes such as loss of power or communication. The Actuator System may buffer pre-planned MRM trajectories or simply revert to blind braking along the last known curvature.

- If only one nominal trajectory is considered OK by both channels, it is automatically forwarded, and the Selector informs the Diagnostics System of a potential fault or output insufficiency. The Diagnostics System may then request both channels to switch to degraded mode (nominal trajectory plans to pull over and come to a controlled stop).
- If none of the nominal trajectories are considered OK by both channels, no nominal trajectory is forwarded. In this case the Actuator System will automatically use an MRM trajectory, either one that the ADI continues to update or the latest pre-planned (buffered) one.
- If both pre-planned MRM trajectories are considered OK by both channels, the one with the better score is forwarded to the Actuator System and buffered there.
- If only one pre-planned MRM trajectory is considered OK by both channels, it is automatically forwarded.
- If neither pre-planned MRM trajectory is considered OK by both channels, but at least one nominal still is, no pre-planned MRM trajectory is forwarded. In this case the Selector informs the Diagnostics System of a potential fault or output insufficiency and the Diagnostics System may then request both channels to switch to degraded mode (nominal trajectory plans to pull over and come to a controlled stop).
- Switching back and forth between channels is allowed, e.g., the better-scoring channel can be in control as long as the steadiness of the trajectory is ensured, e.g., by constraining switching frequency or the replanning of the next few setpoints.

This is summarized in Table 7, where it is assumed that a channel will only output data that passes its own validation (e.g., Channel 1 validation result of Channel 1 nominal is "OK", otherwise there is no output and Channel 2 validation result of Channel 1 nominal is "NOK").

Table 7: Decision logic of the Selector subsystem in the Cross-Checking Pair architecture.

Channel 1 nominal	Channel 1 MRM	Channel 2 nominal	Channel 2 MRM	Selector subsystem action
"OK" (according to both C1 and C2)	"OK"	"OK"	"OK"	Forward higher-ranked nominal and higher-ranked pre-planned MRM.
"NOK " ("NOK " according to C1 and/or C2)	"OK"	"OK"	"OK"	Forward C2 nominal and higher-ranked pre-planned MRM. Report to Diagnostics System (send channels to degraded mode).
"OK"	"NOK "	"OK"	"OK"	Forward higher-ranked nominal and C2 pre-planned MRM.
"OK"	"OK"	"NOK "	"OK"	Forward C1 nominal and higher-ranked pre-planned MRM. Report to Diagnostics System (send channels to degraded mode).
"OK"	"OK"	"OK"	"NOK "	Forward higher-ranked nominal and C1 pre-planned MRM.
"NOK "	"NOK "	"OK"	"OK"	Forward C2 nominal and C2 pre-planned MRM. Report to Diagnostics System (send channels to degraded mode).
"NOK "	"OK"	"NOK "	"OK"	No output (ADI remains silent and Actuator System resorts to buffered pre-planned MRM).

Channel 1 nominal	Channel 1 MRM	Channel 2 nominal	Channel 2 MRM	Selector subsystem action
“NOK ”	“OK”	“OK”	“NOK ”	Forward C2 nominal and C1 pre-planned MRM. Report to Diagnostics System (send channels to degraded mode).
“OK”	“NOK ”	“NOK ”	“OK”	Forward C1 nominal and C2 pre-planned MRM. Report to Diagnostics System (send channels to degraded mode).
“OK”	“NOK ”	“OK”	“NOK ”	Forward higher-ranked nominal. Report to Diagnostics System (send channels to degra-ded mode).
“OK”	“OK”	“NOK ”	“NOK ”	Forward C1 nominal and C1 pre-planned MRM. Report to Diagnostics System (send channels to degraded mode).
“NOK ”	“NOK ”	“NOK ”	“OK”	No output.
“NOK ”	“NOK ”	“OK”	“NOK ”	Forward C2 nominal. Report to Diagnostics System (send channels to degraded mode).
“NOK ”	“OK”	“NOK ”	“NOK ”	No output.
“OK”	“NOK ”	“NOK ”	“NOK ”	Forward C1 nominal. Report to Diagnostics System (send channels to degraded mode).
“NOK ”	“NOK ”	“NOK ”	“NOK ”	No output.

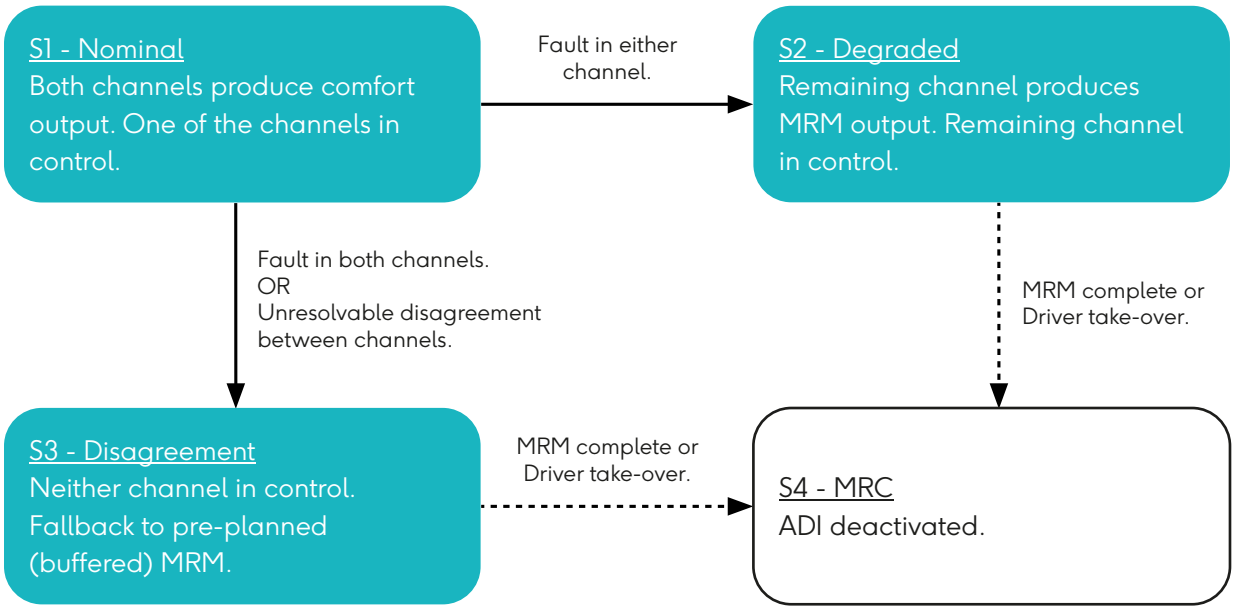


Figure 20: State diagram of the Cross-Checking Pair architecture.

3.4.3 DARUMA ARCHITECTURE

3.4.3.1 UNDERLYING CONCEPTS AND DESIGN PRINCIPLES

The Daruma[32] architecture [21] [37] is based on the Doer/Checker basic pattern and has been evaluated with the public Safety Shell implementation [31] [38]. It uses a scalable approach, i.e., the number of subsystems can be adapted depending on the use case and the desired integrity and availability. To facilitate the integration of existing ADAS without major modifications to functionality[33], such subsystems need not be "aware" of the rest of the system.

In the following, we describe a variant of the architecture based on the design pattern described in [39, 37] with the following modifications:

- Three complex subsystems are used as a basis. This helps resolve potential disagreements between subsystems. At a minimum, two complex subsystems would be necessary [40], in which case the architecture loosely resembles the Cross-Checking Pair architecture (see section 3.4.2).
- The Selector subsystem is duplicated to ensure fault tolerance.
- The Buffering pattern is used to cover the case where the Daruma subsystem is unavailable.

3.4.3.2 STRUCTURAL DESCRIPTION

The Daruma architecture relies on cross-validation of the channels, which is performed by a separate subsystem (see Figure 21). It consists of three complex and three simple subsystems:

- Channels 1-3 all have similar roles: they produce actuator commands and, in addition, output their internal environment model and a pre-planned MRM. To ensure sufficient independence, the channels must be diverse. The outputs they produce can vary in their focus, i.e., at least one must produce nominal (comfort) actuator commands, but the others may produce degraded or MRM actuator commands.
- The Daruma subsystem performs cross-channel validation by comparing the trajectory (and possible associated actuator commands) of each channel against all environment models. It can also consider other diagnostic data. The result of this validation can be scores expressing the safety, comfort, efficiency, driving continuity, etc. of the respective trajectory. Based on these scores, the Daruma subsystem creates a ranking of the channels. In contrast to the Majority Voting architecture, where trajectories that are more similar to others are preferred, Daruma algorithms prefer the trajectory with the best evaluations (from all channels). This is intended to leverage channels' diversity. Two channels might produce similar trajectories, but still consider the trajectory generated by a third channel safer. Dynamic factors can also be taken into account.
- Selectors 1-2 have the same role: they collect actuator commands and validation results and select the actuator commands that have been ranked highest by the Daruma subsystem for execution. If Daruma is silent or found to be faulty by the Diagnostics System, the Selectors instead send a pre-planned (buffered) MRM to the Actuator System.

[32] A Daruma is a traditional Japanese doll that stands for good luck and perseverance, see https://en.wikipedia.org/wiki/Daruma_doll. They are sometimes manufactured as roly-poly toys that right themselves when pushed over.
[33] This is visible in comparison to the Cross-Checking Pair, where the validation functionality is part of the channels themselves. This additional functionality can necessitate significant modifications of an existing ADAS, e.g., higher HW performance requirements.

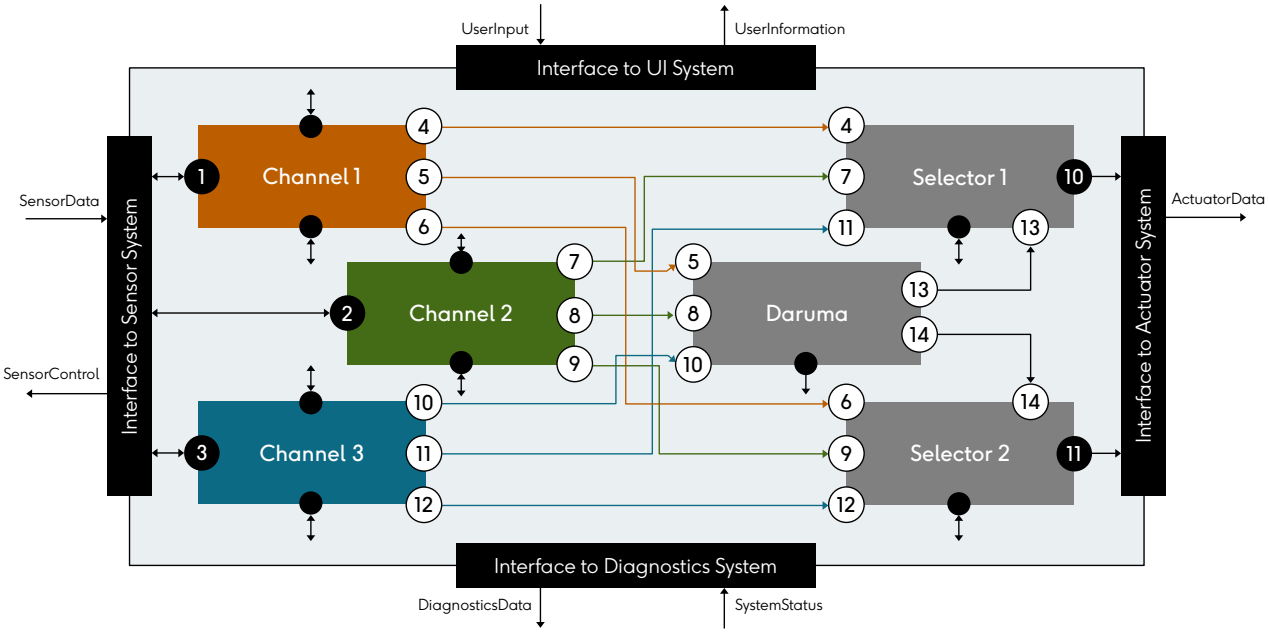


Figure 21: Block diagram of the Daruma architecture.

3.4.3.3 BEHAVIORAL DESCRIPTION

Table 9 describes the behavior of each of the subsystems in more detail. The interaction between subsystems is described in the sequence diagram in Figure 22.

Table 8: Behavioral description of the subsystems in the Daruma architecture.

Subsystem	Behavior
Channel [1-3]	• Receive SensorData from Sensor System (interface #1/#2/#3). • Generate EnvironmentModel and send it to Daruma subsystem (interface #5/#8/#10). • Generate ActuatorData (for nominal trajectory and for pre-planned MRM) and send it to Daruma subsystem (interface #5/#8/#10) and Selectors (interfaces #4&6/#7&9/#11&12).
Daruma	• Receive EnvironmentModels and ActuatorData[34] from channels (interfaces #5&8&10). • Validate each ActuatorData against each EnvironmentModel (3x3) to create ranking. • Send ValidationResults (including hash of validated ActuatorData) to Selectors (interfaces #13&14).
Selector [1-2]	• Receive ActuatorData from channels (interfaces #4&7&11/#6&9&12). • Receive ValidationResults from Daruma subsystem (interface #13/#14). • Based on ValidationResults, select highest-ranked ActuatorData and send them to Actuator System (interface #15/#16). • Buffer highest-ranked pre-planned MRM ActuatorData.

[34] The cross-check focuses on the trajectory, not the actuator setpoints.

The validation in the Daruma subsystem compares each trajectory against each environment model, resulting in 3x3 risk scores, and potentially also environment models against each other. Several properties can be considered separately in the ranking:

- The correctness of the actuator data is a crucial property that must be validated. This property includes the absence of collisions, the physical feasibility of the trajectory, etc. and has priority over all other properties.
- Further relevant properties are the comfort, speed, and efficiency of the actuator data. Degraded trajectories or MRMs will likely score worse in these properties.
- A comparison of the environmental models, e.g., whether objects are correctly recognized.
- A general preference for one of the channels, e.g., the most advanced and performant.

The Selectors base their decision on the following scheme:

- Validation data is only considered if the hash of the checked actuator data matches the corresponding (directly) received actuator data[35]. If the validation data is not received or the hash doesn't match, the related validation results are assumed to be "NOK".
- Actuator data with a higher ranking as output by the Daruma subsystem (according to safety, comfort, speed, efficiency, etc.) is preferred. If Daruma considers a channel insufficiently safe, it marks them (indicated by an asterisk in Figure 44 and Figure 45). If none of the channels are considered sufficiently safe, the Selectors resort to the buffered pre-planned MRM. Switching back and forth between channels is allowed but limited in frequency.

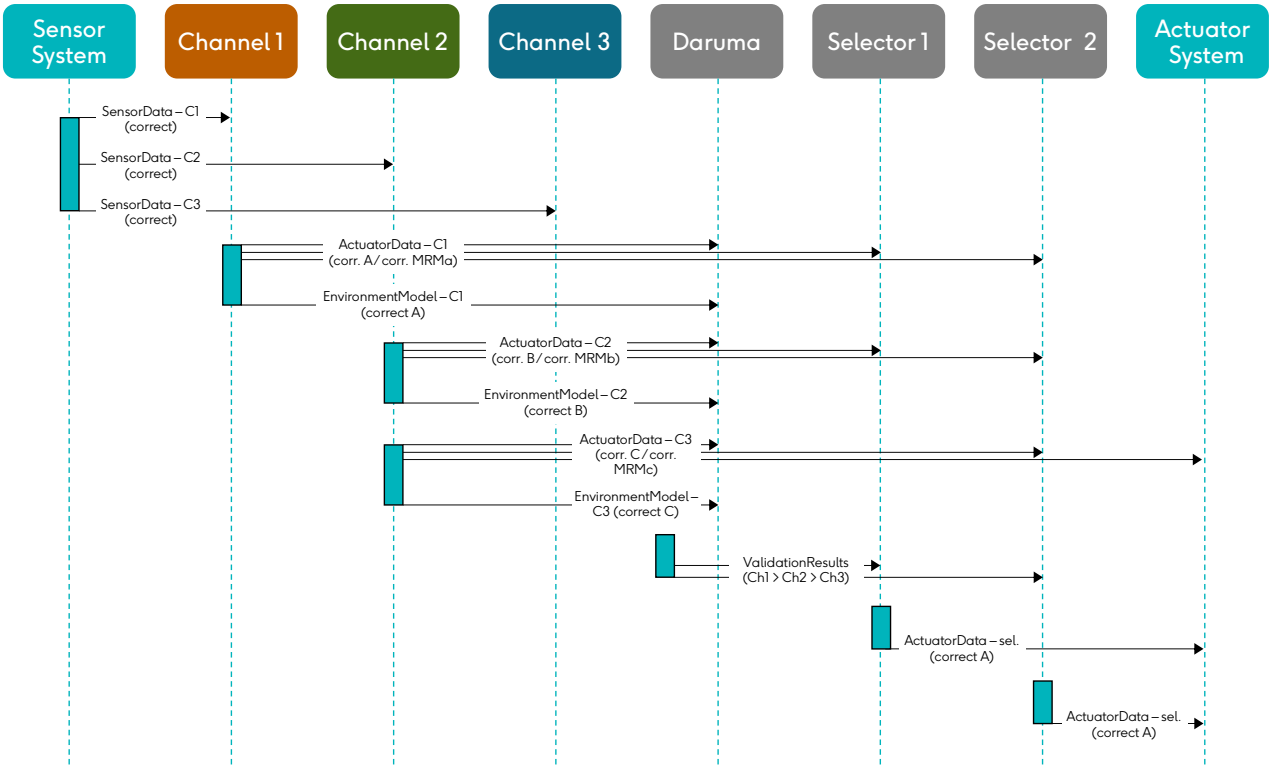


Figure 22: Sequence diagram of the Daruma architecture. The nominal case without faults or functional insufficiencies is shown.

[35] Although this is not described in the source material, it is a reasonable modification (or implementation consideration) to detect Byzantine faults.

3.5 ASYMMETRIC ARCHITECTURES

3.5.1 CHANNEL-WISE DOER / CHECKER / FALLBACK ARCHITECTURE

This section discusses the architecture proposed by Kopetz [25] [41] [42], which can be considered a specific combination of the Doer / Checker and Active / Hot Stand-By (see section 3.2) approaches for decomposition with respect to integrity and availability, respectively. We refer to this architecture as Channel-Wise Doer / Checker / Fallback (C-DCF)[36]. In this case, the decomposition is done for entire processing channels. Involved design principles and their respective intentions are:

- Minimizing interactions between the different subsystems (in this case entire processing channels) is intended to reduce complexity and to prevent emergent behavior (see also D5: *Avoidance of emergent behavior*).
- The source material suggests employing a Time-Triggered Architecture (TTA) to reduce ambiguity between late vs. missing messages and to prevent the formation of mutually inconsistent time domains. This should be considered an implementation consideration as timing is generally a safety-related property in systems with safety-related availability.

3.5.1.1 UNDERLYING CONCEPTS AND DESIGN PRINCIPLES

The C-DCF architecture is based on a combination of the Doer / Checker approach (for decomposition with respect to integrity) and the Active / Hot Stand-By approach (for decomposition with respect to availability). These are outlined in sections 3.2.1.3 and 3.2.1.5, respectively.

The conceptual architecture of the variant proposed by Kopetz is based on a Time-Triggered Architecture (TTA), i.e., scheduled task execution and communication across all subsystems. This simplifies the Doer / Checker and Active / Hot Stand-By decompositions:

- Missing and delayed messages between channels are treated the same way.
- Latencies due to communication between channels and the redundancy management can be bounded and reduced.

It further requires sufficient independence between channels to prevent common cause faults[37]:

- Diversity in the HW and SW implementations of the different channels is required (see section 5).
- The source material also proposes to use three disjoint sensor sets as input for each channel but acknowledges that this can likely not be fully achieved due to cost reasons.

3.5.1.2 STRUCTURAL DESCRIPTION

The proposed conceptual system architecture consists of three complex and two simple subsystems (see Figure 23), namely:

- The Primary Driving System (P) controls the vehicle under nominal conditions, i.e., it is similar to some SAE L2 systems[25][38]. It periodically produces nominal trajectories (e.g., timed waypoints) and actuator setpoints (e.g., desired acceleration / deceleration and curvature values for steering, powertrain, and brakes) and transmits these to the Monitoring System and the Decision System.
- The Monitoring System (M) detects unsafe trajectories produced by the P-System, whether nominal conditions prevail, and whether the Fallback System is still alive.

[36] The source material does not state an explicit name for this architecture. Within this report, it is referred to as Doer / Checker / Fallback due to its combination of said architectural design patterns. The suffix Channel-Wise is to differentiate it from another DCF architecture.
[37] For complex subsystems (i.e., the Primary Driving, Monitoring, and Fallback System described in the subsequent section) this may necessitate diverse SW implementations. For sufficiently simple subsystems (i.e., the Decision System) with fully verifiable SW, no diversity is necessary.
[38] The other subsystems effectively take over the tasks performed by a human driver in an SAE L2 system and are collectively called Safety Assurance Subsystem in .

- The Fallback System (F) controls the vehicle under off-nominal conditions. It only aims to bring the vehicle to a minimal risk condition, i.e., to execute an MRM, but must be able to do that even after an ODD exit. It periodically produces trajectories and actuator setpoints and transmits these to the Decision System.
- The Decision System (D) decides which setpoints are forwarded to the Actuator System. It consists of two identical instances (D1 and D2) to achieve fault tolerance.

A more detailed structural description of the subsystems is given in the source material.

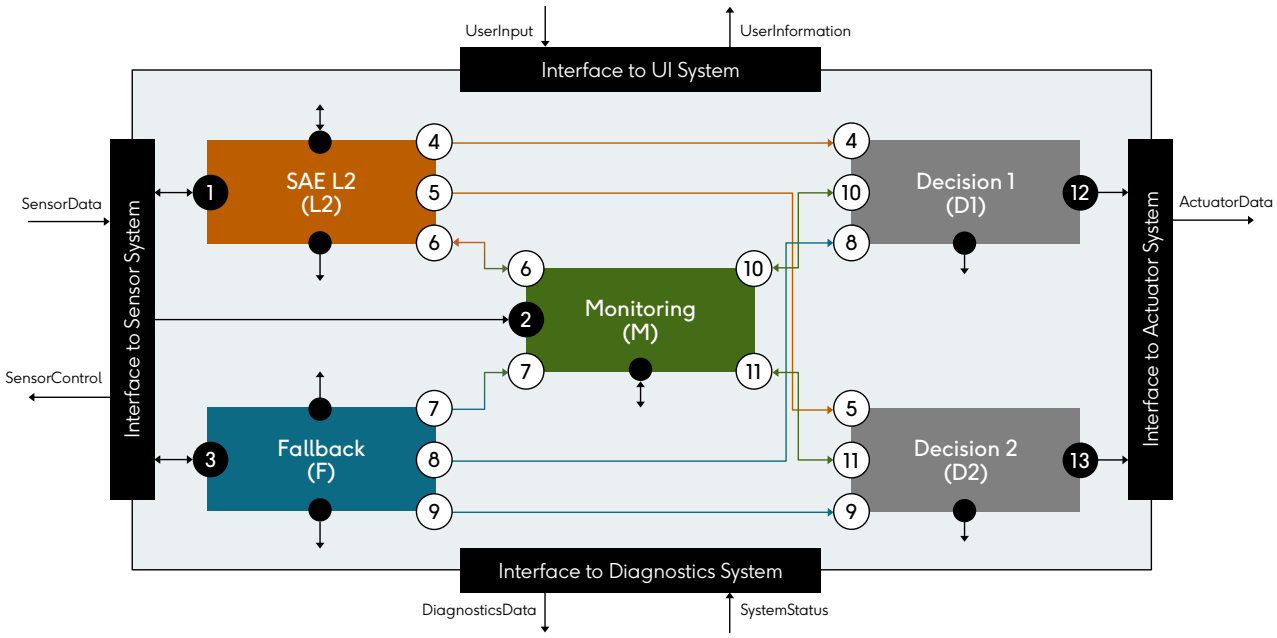


Figure 23: Block diagram of the Channel-Wise Doer/Checker/Fallback architecture.

3.5.1.3 BEHAVIORAL DESCRIPTION

Table 9 describes the behavior of each of the subsystems in more detail. The interaction between subsystems is described in the sequence diagram in Figure 24 (showing the time-driven tasks with the common main cycle time of the AD Intelligence in the fault-free case). Consensus in the Actuator System (compare S4: *AD Intelligence output consistency*) is straightforward: actuators follow the received setpoints with the higher priority, i.e., they prefer L2 setpoints over F setpoints if they receive both.

Table 9: Behavior of the subsystems of the Channel-Wise Doer / Checker / Fallback architecture.

Subsystem	Behavior
SAE L2	• Receive SensorData from Sensor System (interface #1). • Generate ActuatorData (for nominal trajectory) and send it to M-System (interface #6) and D-Systems (interfaces #4&5). • Go to degraded mode (MRM only) if system state demands it (interface #6; e.g., if F-System is faulty).
Fallback	• Receive SensorData from Sensor System (interface #3). • Generate ActuatorData (for MRM trajectory) and send it to M-System (interface #7) and D-Systems (interfaces #8&9).
Monitoring	• Receive SensorData from Sensor System (interface #2). • Receive ActuatorData from L2-System (interface #6) and F-System (interface #7). • Receive ActuatorData from D-Systems (relay on interfaces #10&11). • Validate ActuatorData from L2-System. This involves checking the received trajectory / actuator setpoints against the M-System's environment model. • Check that ActuatorData from L2-System is same as received from the D-Systems. This is intended to catch Byzantine failures where the L2-System sends different outputs to the M-System and the D-Systems. • Validate ActuatorData from F-System. • Check that ActuatorData from F-System is same as received from the D-Systems. • Send ValidationResult to D-Systems (interfaces #10&11). This determines which subsystem will be in control of the actuators. • Send ValidationResults to L2-System (interface #6). If the F-System is faulty, this will request the L2-System to restrict itself to degraded or MRM trajectories.
Decision [1-2]	• Forward ActuatorData from L2-System (interface #4 / #5) and F-System (interface #8 / #9) back to M-System (interface #10 / #11). This is intended to catch Byzantine failures. • Receive ValidationResults from M-System (interface #10 / #11). • Select ActuatorData from L2-System or F-System depending on ValidationResults and send it to Actuator System (interface #12 / #13).

For all of the AD Intelligence, task execution and communication are based on a time-triggered schedule. If a message is not received in the planned time slot (or does not have the correct iteration counter), it counts the same as if it hadn't been received at all or if it had been received in a corrupted state (e.g., with an invalid checksum).

Transient faults in one of the complex subsystems can occur quite frequently, so the D-Systems allow recovery of the L2-System if necessary[39]. However, unduly frequent transient faults are considered indicative of an underlying problem and cause the L2-System to go into a degraded mode (see Figure 25 and G5: Frequent switching).

[39] How feasible a switch back to the L2-System is depends on the involved time scales. If the L2-System can be recovered quickly and the F-System generates degraded trajectories (not MRMs), it may not be very noticeable to the passengers. The source material assumes that many P-System failures will be transient, e.g., due to a temporary SOTIF violation. In any case, continuous back-and-forth switching must be prevented.

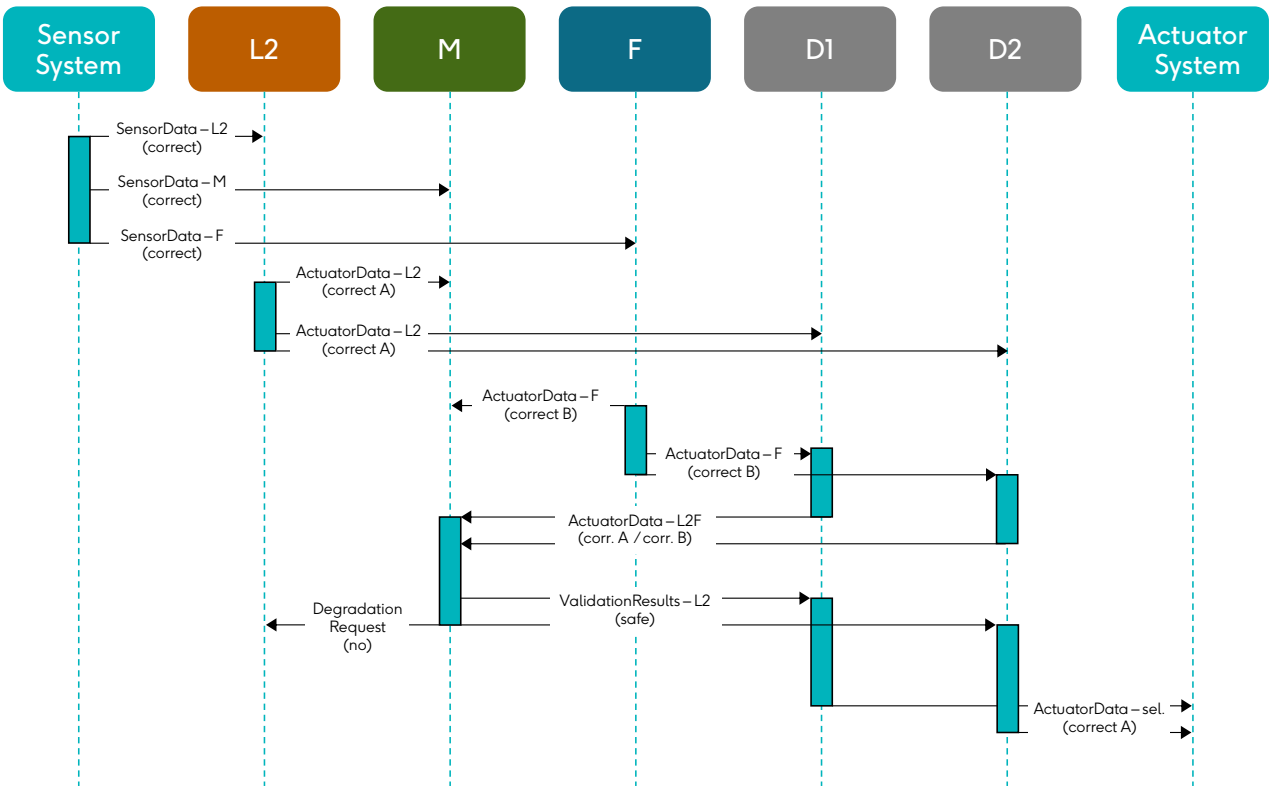


Figure 24: Sequence diagram of the C-DCF architecture. The nominal case without faults or functional insufficiencies is shown.

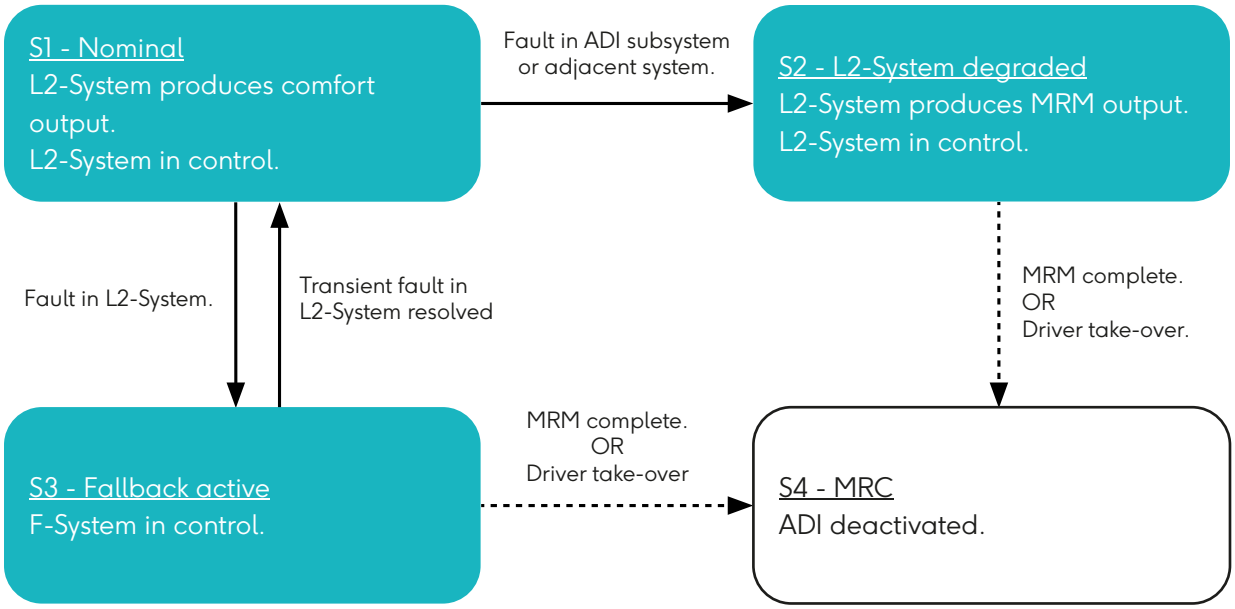


Figure 25: State diagram of the C-DCF architecture.

3.5.2 LAYER-WISE DOER / CHECKER / FALLBACK ARCHITECTURE

In [43], a multi-channel approach combined with the Doer / Checker pattern is presented in a patent as a safety architecture for AD. This section summarizes the most relevant aspects of this invention, which we refer to as Layer-Wise Doer / Checker / Fallback (L-DCF)[40].

[40] The source material does not state an explicit name for this architecture. Within this report, it is referred to as Doer / Checker / Fallback due to its combination of said architectural design patterns. The suffix Layer-Wise is to differentiate it from another DCF architecture.

3.5.2.1 UNDERLYING CONCEPTS AND DESIGN PRINCIPLES

Doer / Checker pairs are used as the main architectural pattern. The primary Doer / Checker pair acts during normal mode, and a secondary ("safing") Doer / Checker pair provides a degraded mode of operation in case the primary pair fails. Additionally, the arbiter "Priority Selector" determines the output to be sent to the actuators (see Figure 26). The source material proposes to use two disjoint sensor sets as input for the primary and safing channels.

3.5.2.2 STRUCTURAL DESCRIPTION

The pattern shown in Figure 26 can be repeated for different layers or stages (e.g., perception, planning, and trajectory execution. It consists of four complex and two simple subsystems, namely:

- The Primary unit (P) controls the vehicle under nominal conditions. Depending on the layer, this may involve perception, planning, and/or trajectory execution. It may have a low integrity level and fail arbitrarily.
- The Primary Safety Gate (PSG) complements the Primary to form a Doer / Checker pair. It checks the outputs of the Primary and silences it if necessary. It is a high-integrity component and is fail-silent.
- The Safing unit (S) controls the vehicle if the Primary unit is faulty or not available. Depending on the layer, this may involve perception, planning, and/or trajectory execution. It also generates a "Permissive Envelope" signal, which indicates a reference used to validate the Primary output. The Permissive Envelope may, for example, specify a maximum acceleration rate. It needs to be carefully implemented to prevent a single point of failure where the Primary channel tries to generate a trajectory that lies inside a faulty envelope while at the same time the Safing channel also generates a faulty trajectory.
- The Safing Safety Gate (SSG) checks the outputs of the Safing and silences it if necessary. It also evaluates whether the Permissive Envelope is appropriate and whether the Primary lies within it, and buffers the MRM planned by the Safing unit.
- The Priority Selector (PS) makes the final selection of outputs to pass on to the Actuator System. It consists of two identical instances (PS1 and PS2) to achieve fault tolerance. The instances may fail silently but must then trigger an emergency stop using either a buffered pre-planned MRM trajectory or a "constant braking" (blind braking) maneuver. As it is simpler than the Safety Gates, a great deal of effort can be spent on its verification to achieve the required high level of integrity.

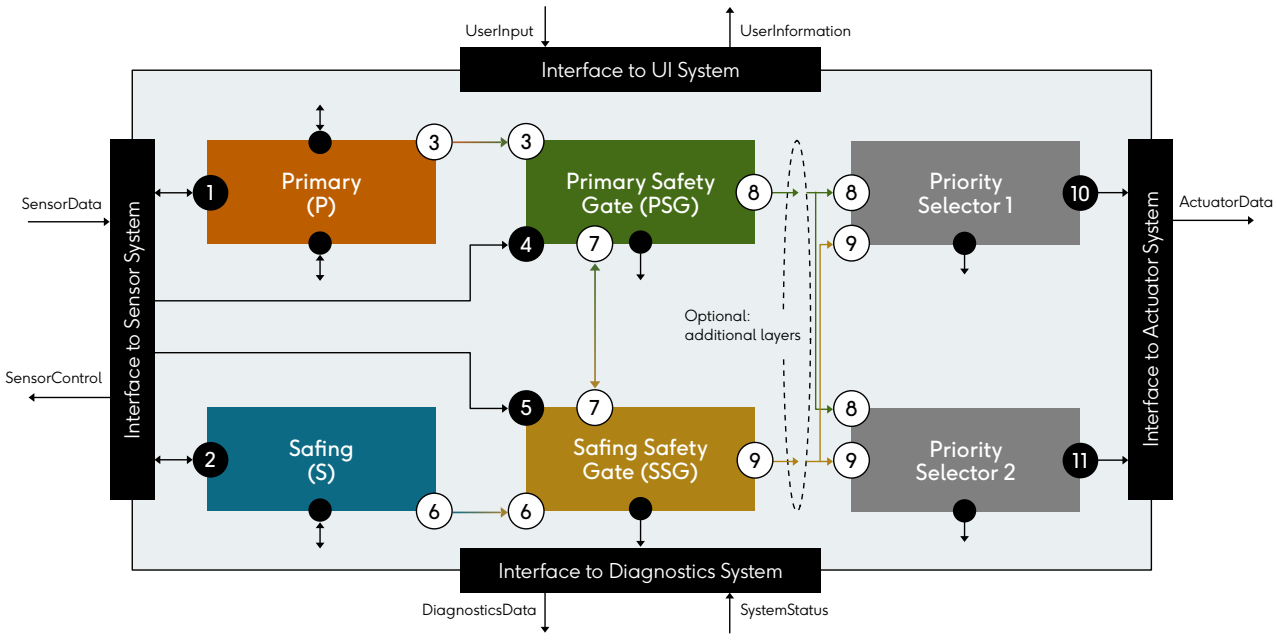


Figure 26: Block diagram of the Layer-Wise Doer / Checker / Fallback architecture.

3.5.2.3 BEHAVIORAL DESCRIPTION

Various implementation alternatives are mentioned in the patent (e.g., time-triggered vs. event-triggered architecture). Depending on the choice, the system will behave differently. Table 10 describes the behavior of each of the subsystems in more detail. The interaction between subsystems is described in the sequence diagram in Figure 27.

Table 10: Behavioral description of the subsystems in the Layer-Wise Doer / Checker / Fallback architecture.

Subsystem	Behavior
Primary	• Receive primary SensorData from Sensor System (interface #1). • Depending on the layer, generate perception output, planning output (nominal trajectory), or execution output (nominal ActuatorData) and send it to Primary Safety Gate (interface #3).
Primary Safety Gate	• Receive safing SensorData from Sensor System (interface #4). • Receive output from Primary (interface #3). • Receive Permissive Envelope from Safing Safety Gate (interface #7). • Validate output of Primary (including against Permissive Envelope) and only send valid output to Safing Safety Gate (interface #7) and Priority Selectors (interface #8).
Safing	• Receive safing SensorData from Sensor System (interface #2). • Depending on the layer, generate perception output, planning output (pre-planned MRM), or execution output (pre-planned MRM ActuatorData) and send it to Safing Safety Gate (interface #6). • Generate Permissive Envelope and send it to Safing Safety Gate (interface #6).
Safing Safety Gate	• Receive primary SensorData from Sensor System (interface #5). • Receive output from Safing (including Permissive Envelope; interface #7). • Validate output of Safing and only send valid output to Priority Selectors (interface #9). • Buffer valid Safing output (pre-planned MRM). • Validate Permissive Envelope and forward it to Primary Safing Gate (interface #7). • Validate whether Primary lies within Permissive Envelope.
Priority Selector [1-2]	• Receive ActuatorData from Primary Safety Gate (interface #8) and Safing Safety Gate (interface #9). • Forward highest-ranked ActuatorData to Actuator System (interface #10/#11). • Resort to “constant braking” (blind MRM) command if no valid ActuatorData available (interface #10/#11).

The two channels each consist of a fail-silent Doer/Checker pair.

- These channels can be stacked in multiple layers. The Primary output of the first layer serves as input to the Primary of the second layer, whereas the Safing output serves as input to the Primary Safety Gate, Safing, and Safing Safety Gate of the second layer.
- If both the Primary unit and the Safing unit fail, the system remains safe by recovering actions performed by the downstream stages (e.g., executing an emergency stop).

The Priority Selectors are fault-tolerant to ensure availability of the system.

- If the Safing Safety Gate for the planning stage fails, both Primary and Safing outputs are inhibited. The downstream stage gets no inputs, and thus sends no outputs, which causes the execution of the last safe trajectory.
- If the Safing Safety Gate for the trajectory execution stage fails, both Primary and Safing output are inhibited, which causes the Priority Selector to execute an emergency stop.

The Safing unit generates pre-planned MRM trajectories, which are buffered in the Safing Safety Gate and checked for validity in a cyclic (i.e., periodic and deterministic) way. These can be used as a last resort if neither channel is available.

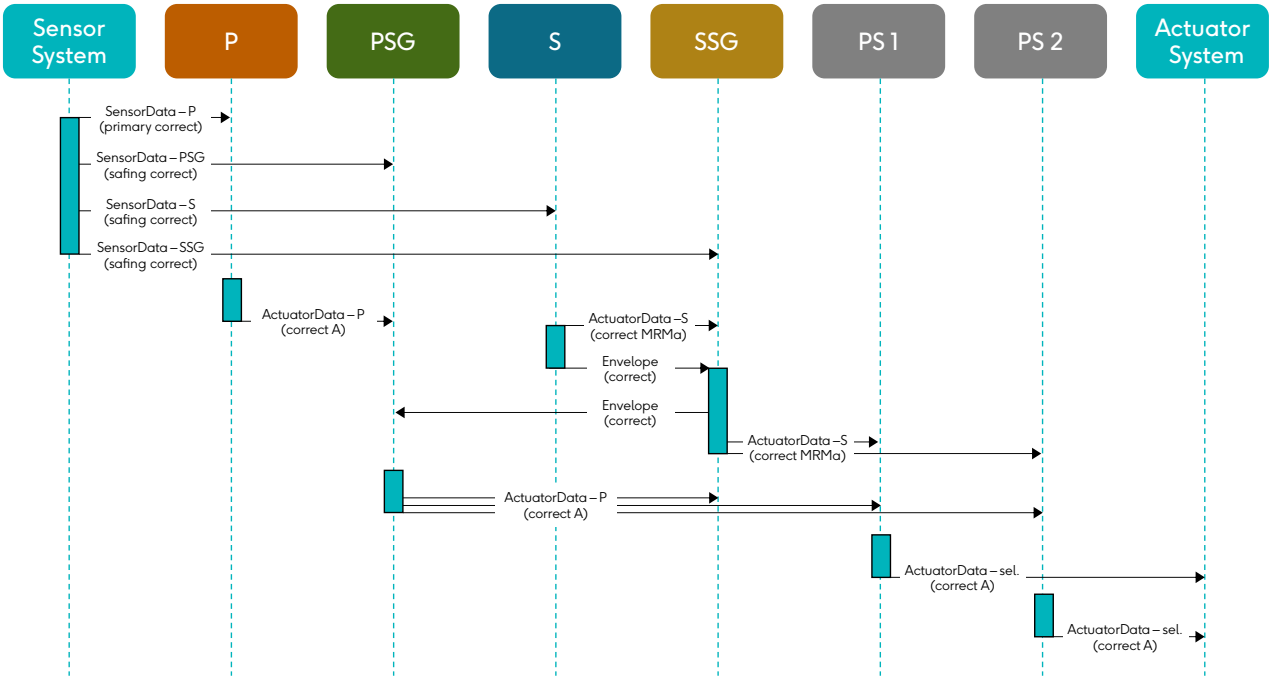


Figure 27: Sequence diagram of the L-DCF architecture. The nominal case without faults or functional insufficiencies is shown.

3.5.3 DISTRIBUTED SAFETY MECHANISM ARCHITECTURE

This section discusses the Distributed Safety Mechanism (DSM) architecture proposed in [44] [45][41], which can be considered a distributed variant of the Doer/Checker/Fallback approach. It has been formally verified in the source material.

3.5.3.1 UNDERLYING CONCEPTS AND DESIGN PRINCIPLES

The DSM architecture is based on separation of concerns and a distributed monitoring approach. These are intended to enable a highly scalable architecture by allowing the nominal channel to be extended for more functionality without affecting the rest of the system. Inspired by the E-GAS layered monitoring concept [46], it includes safety monitoring components at function, controller, and vehicle levels.

This architecture is based on exactly one subsystem being in control of the Actuator System at all times:

- All subsystems apart from the controlling one are silenced, i.e., they do not send control commands to the Actuator System. This is different compared to the C-DCF architecture,

[41] An open-access version of this paper is available at https://arxiv.org/ftp/arxiv/papers/2011/2011.00892.pdf

where a “smart switch” decides which control commands to forward to the Actuator System.

- High reliability and determinism are required by the system-level distributed communication protocol to support the safety mechanisms.

The source material provides additional details on suggested implementation:
Redundant communication networks (primary and secondary) should be used.
Two disjoint sensor sets should be used as input for the nominal and safety channels (see source material for more details).

In the following, we describe a variant of the proposed architecture with the modification that the SFM subsystem receives sensor data that it can process, not just sensor diagnostics data.

3.5.3.2 STRUCTURAL DESCRIPTION

The DSM architecture consists of four subsystems (see Figure 28):

- The Function channel (FUN) performs the nominal function. It periodically produces trajectories and actuator setpoints and transmits these to the Sensor and Function Safety Monitor and Actuator System. It also sends diagnostic data to SFM. The DSM architecture foresees the possibility to add additional FUN subsystems in a modular way.
- The Sensor and Function Safety Monitor (SFM) detects faults and functional insufficiencies in the FUN and in itself and sends diagnostic data to the Controller Safety Mechanism. It can also suppress the output of the FUN[42]. As a monitor, its primary concern is the safety of the AD functions, e.g., from a SOTIF perspective.
- The Controller Safety Mechanism (CSM) can trigger a shutdown of the FUN and/or SFM. It monitors the Vehicle Safety Mechanism through a watchdog and also compares the FUN trajectory against an envelope produced by the VSM. As a monitor, its primary concern is the safety of the function controller (FUN + SFM), e.g., from a HW and platform SW perspective. As a “last resort” if the VSM is not available, it can generate MRM actuator setpoints and send these to the Actuator System[43].
- The Vehicle Safety Mechanism (VSM) monitors the status of the other subsystems and sends MRM actuator setpoints to the Actuator System if any of them are not available. It also generates a checking envelope that the CSM can use to check the FUN trajectory. As a monitor, its primary concern is the safety of the vehicle, e.g., from a data and power networks perspective. It is assumed to fail silently.

[42] The source material considers a service-oriented architecture, which is difficult to represent as a conceptual system architecture.
[43] Such a last resort could be constant “blind” braking on the last known curvature. It could also be implemented on the Actuator System side.

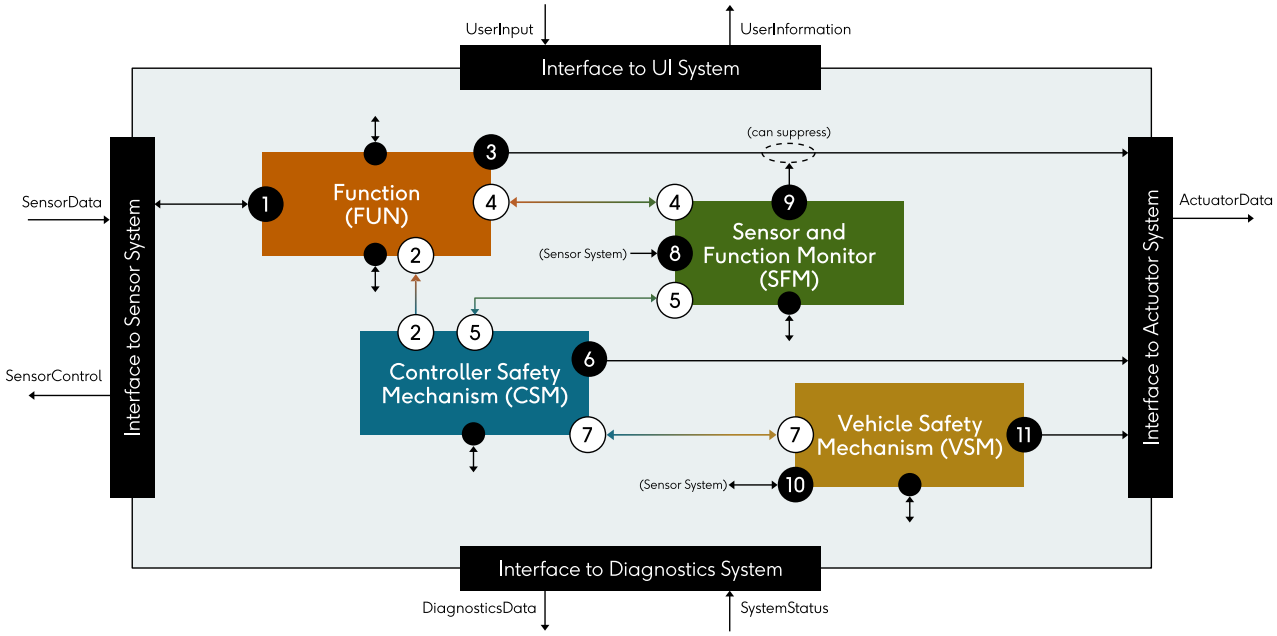


Figure 28: Block diagram of the DSM architecture

3.5.3.3 BEHAVIORAL DESCRIPTION

Table 11 describes the behavior of each of the subsystems in more detail. The interaction between subsystems is described in the sequence diagram in Figure 29. The DSM architecture considers a system-wide degradation concept with five system states, i.e., nominal, detour, comfort stop, safe stop, or emergency stop. As shown in more detail in Figure 30, the appropriate degradation depends on which subsystem is faulty.

Table 11: Behavior of the subsystems of the DSM architecture.

Subsystem	Behavior
Function	• Receive nominal SensorData from Sensor System (interface #1). • Generate ActuatorData (usually "nominal" mode; "detour" degraded mode only when requested by CSM on interface #2) and send it to SFM (interface #4) and Actuator System (interface #3).
Sensor and Function Monitor	• Receive nominal SensorData from Sensor System (interface #8). • Diagnose FUN (interface #4) and send DiagnosticsData and FUN trajectory to CSM (interface #5). • If necessary, suppress FUN output (interface #9).

Subsystem	Behavior
Controller Safety Mecha-nism	• Receive DiagnosticsData and FUN trajectory from SFM (interface #5). • Receive VSM envelope (interface #7) and compare to FUN trajectory. • If FUN is found to be faulty, silence FUN (interface #2). • If SFM is found to be faulty, silence SFM (interface #5) and request FUN to go to “detour” degraded mode (interface #2). • Send DiagnosticsData to VSM (interface #7). • Initiate watchdog challenge to VSM (interface #7). • If watchdog challenge to VSM fails, generate ActuatorData (“emergency stop” blind MRM) and send it to Actuator System (interface #6). • Reply to watchdog challenge from VSM (interface #7).
Vehicle Safety Mecha-nism	• Receive safety SensorData from Sensor System (interface #10). • Receive DiagnosticsData from CSM (interface #7). • Generate checking envelope and send it to CSM (interface #7). • If FUN or CSM is found to be faulty, generate ActuatorData (“comfort stop” or “safe stop” MRM) and send it to Actuator System (interface #11). • Reply to watchdog challenge from CSM (interface #7). • Initiate watchdog challenge to CSM (interface #7).

As shown in the sequence diagram, several communication paths are not used in the nominal case, i.e., some subsystem outputs are silenced. Only in certain cases with faults or functional insufficiencies, i.e., when the FUN subsystem is silenced by the CSM, are these degraded or emergency options used.

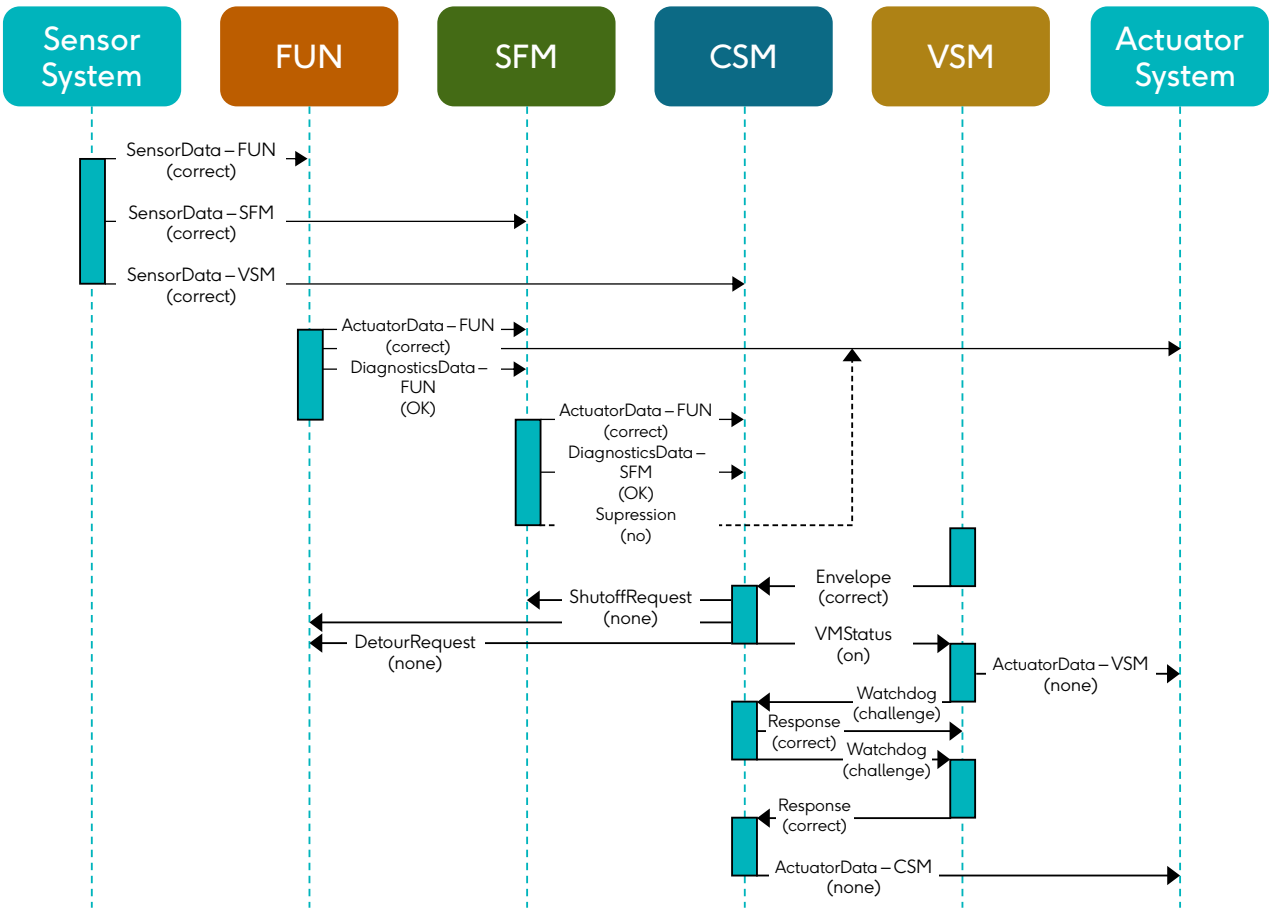


Figure 29: Sequence diagram of the DSM architecture. The nominal case without faults or functional insufficiencies is shown.

Depending on the nature of the fault or output insufficiency, the CSM and VSM can use different levels of degradation (similar to the degradation cascades described in [47]) as shown in Figure 30. If a fault or output insufficiency is detected in FUN or SFM, the corresponding subsystem is turned off and thus silenced. The CSM can use one of two options:

- If VSM is not available, it uses “detour”, which describes a degraded functionality. The vehicle can reduce risk by continuing at a lower speed but aims to complete the mission (“limp-home mode”).
- If VSM and FUN or SFM are not available, it uses “emergency stop”, which describes an MRM. The vehicle brakes blindly, following the last known curvature.

The VSM can use one of the following two options:

- If FUN is not available, it uses “comfort stop”, which describes a degraded trajectory. The vehicle gracefully pulls over to the right side and comes to a controlled stop.
- If CSM is not available, it uses “safe stop”, which describes an MRM. The vehicle pulls over to the right side or comes to a stop in-lane.

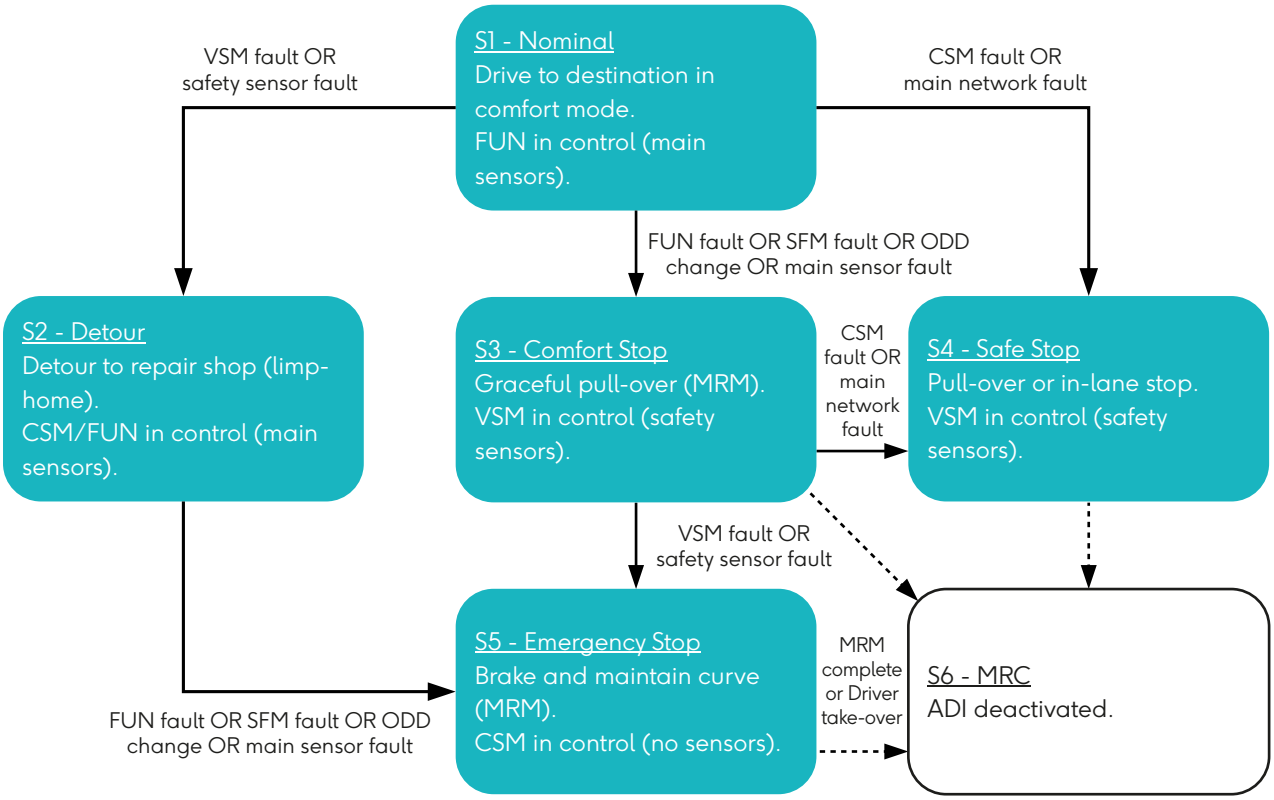


Figure 30: State diagram of the DSM architecture.

3.5.4 AD-EYE ARCHITECTURE

3.5.4.1 UNDERLYING CONCEPTS AND DESIGN PRINCIPLES

This architecture was first proposed and described in the following patents [48] [49], and has since been described in [50], [51] and realization aspects have been described in [52]. The design has the following underlying principles.

1. Avoiding unnecessary complexity. Introducing multiple complex channels significantly increases system cost, creates additional failure modes, and makes agreement between channels harder, particularly for SOTIF-related issues. While some complexity is essential due to the nature of the functional needs, minimizing the accidental[107][44] complexity of the system is to be strived for.
2. Separation of concerns and structured diversity. The AD-EYE architecture enforces separation of concerns by assigning distinct objectives to each channel: performance for Channel 1 and robustness for Channel 2. This structured approach results in naturally diverse designs without unnecessary redundancy. The channels apply a Doer-Checker-Fallback pattern, maintaining structural independence between primary operation, monitoring, and fallback. This separation limits the spread of high ASIL requirements, keeping critical functions verifiable and the Selector subsystem deliberately simple to ease formal validation.
3. Robustness towards transient errors and graceful degradation. A core design principle to be robust against transient errors, supporting controlled degradation during faults, and allowing adaptation to different operational contexts without structural redesign.
4. Reuse and Scalability. Variability in feature maturity and performance requirements across diverse offerings—spanning ODDs, vehicle classes, and feature configurations—is inherent to any OEM portfolio. To consistently address this variability is essential to reduce engi-

[44] The terms Accidental and Essential complexity are terms first defined in [107].

neering overhead and minimize verification and validation effort, requiring an architecture that is flexible enough to scale to the different offering levels while keeping the essence of the structure unchanged.

3.5.4.2 STRUCTURAL DESCRIPTION

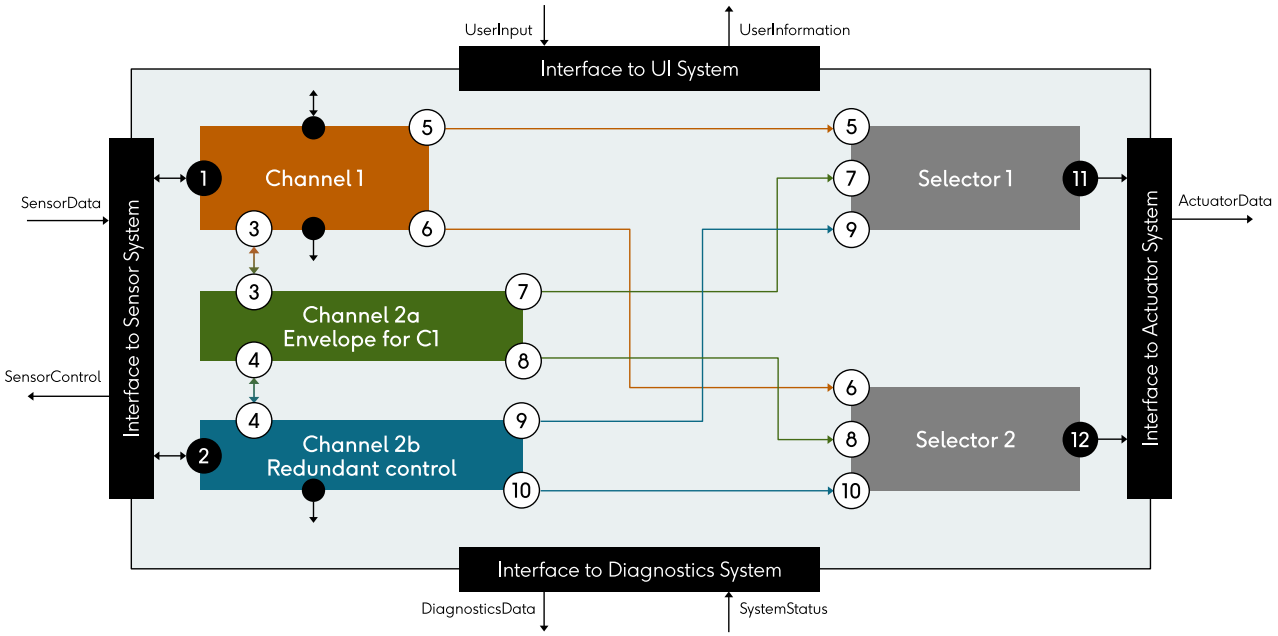


Figure 31: Block diagram of the AD-EYE architecture.

The AD-EYE architecture proposed by KTH comprises two asymmetric channels, Channel 1 and Channel 2, and a discrete Selector. Both channels produce equivalent outputs with respect to the Actuator System, but only one channel's output is sent to the actuators at a time. The channels degrade independently and can independently execute an MRM to reach a Minimal Risk Condition (MRC). In contrast to some other architectures, such as the C-DCF architecture, AD-EYE allows for graceful degradation of Channel 1 in case of transient errors and functional insufficiencies, since Channel 2a determines the operational envelope for Channel 1. The internal redundancy within each subsystem can be adjusted to meet specific operational needs, but each subsystem fulfills distinct and fixed roles.

- Channel 1 is in control of the AD function by default and sends commands via the Selector component. It is inherently complex, optimized for performance, and supports graceful degradation and recovery from transient errors, temporary limitations, and other manageable faults, restoring full functionality when conditions permit.
- Channel 2 is minimalistic to simplify validation, reduce computational complexity, and minimize systematic errors. It maintains an independent simplified perception stack across its two components and uses deterministic algorithms where possible.
  - Channel 2a monitors Channel 1 using a simplified environmental model, platform status, and road network information. It defines and continuously adjusts an Operational Envelope that constrains Channel 1's operation based on current system health. Envelope limits are tightened under degraded conditions, such as perception faults in Channel 1, to prevent operation in uncertain environments. For most non-severe faults, Channel 2a does not invalidate Channel 1's data but constrains its behavior through envelope adjustments. For other non-severe detected faults reported from Channel 1, Channel 2a is permitted to change the destination given to Channel 2, e.g., to a workshop or a stop outside of the highway in addition to the

Operational Envelope constraints to allow for a limp home mode. In the most restrictive case, the envelope may force Channel 1 to perform an MRM in the current lane. Channel 2a may select Channel 2b to execute an MRM only when a severe internal fault is detected within Channel 1, e.g., something that may compromise the safety of the trajectory output of Channel 1.

  - Channel 2b provides a robust fallback, activating if Channel 1 is severely degraded or is unavailable. It executes MRMs based on precomputed trajectories, which are continuously revalidated using sensor data from Channel 2's perception stack over a limited time horizon.

- The Selector design is deliberately simple, minimizing internal logic to meet high safety requirements cost-effectively. It acts as a multiplexer, and only forwards actuator commands from the selected channel to the actuator subsystem without additional processing.

3.5.4.3 BEHAVIORAL DESCRIPTION

Table 12: Behavior of Subsystems within the AD-EYE architecture

Subsystem	Behavior
Channel 1	• Receive nominal SensorData from Sensor System (interface #1). • Generate nominal ActuatorData and send it to Actuator System via the Selectors (interface #5, #6). • Expose internal states and ActuatorData from Channel 1 to Channel 2a and receive constraints to the Operational Envelope of Channel 1 from Channel 2a. Receive heartbeat from Channel 2a. (interface #3)
Selector [1-2]	• Gateway nominal ActuatorData to Actuator System (interface #11, #12), unless safety ActuatorData is required, based on decision from Channel 2a (interface #7, #8).
Channel 2a	• Monitor the internal states and ActuatorData of Channel 1 and generate constraints on the Operational Envelope of Channel 1 based on stored data, safety SensorData. Send heartbeat for the status of Channel 2. (interface #3) • Receive perception data from Channel 2b. (interface #4) • Inform the Selectors which ActuatorData, either safety or nominal, is to be sent on to the Actuator System. (interface #7, #8)
Channel 2b	• Receive safety SensorData from Sensor System (interface #2). • Send perception data to Channel 2a. (interface #4) • Generate safety ActuatorData (MRM and pre-planned MRM) and send it to Actuator System via the Selectors (interface #9,10)

Figure 33 shows a state machine representation, which is explained in more detail in Table 13. The sequence of events under nominal operation is shown in Figure 32.

Table 13: Transitions in the state diagram for the AD-EYE architecture.

Transition	Causes
1	"SOTIF" errors, errors in vehicle platform, other errors in Channel 1 etc.
2	No errors detected or transient errors are healed
3	Detection of an irrecoverable error in Channel 1, or when progress to goal cannot be achieved by Channel 1 due to limitations in environment or limited operational envelope caused by degradations.
4	Detection of an irrecoverable error in Channel 2
5	Reaching the desired goal, the fallback goal due to degraded conditions, or the emergency MRC

Degraded modes would be implemented by Channel 1, and their triggering could come from either Channel 1 or from Channel 2.

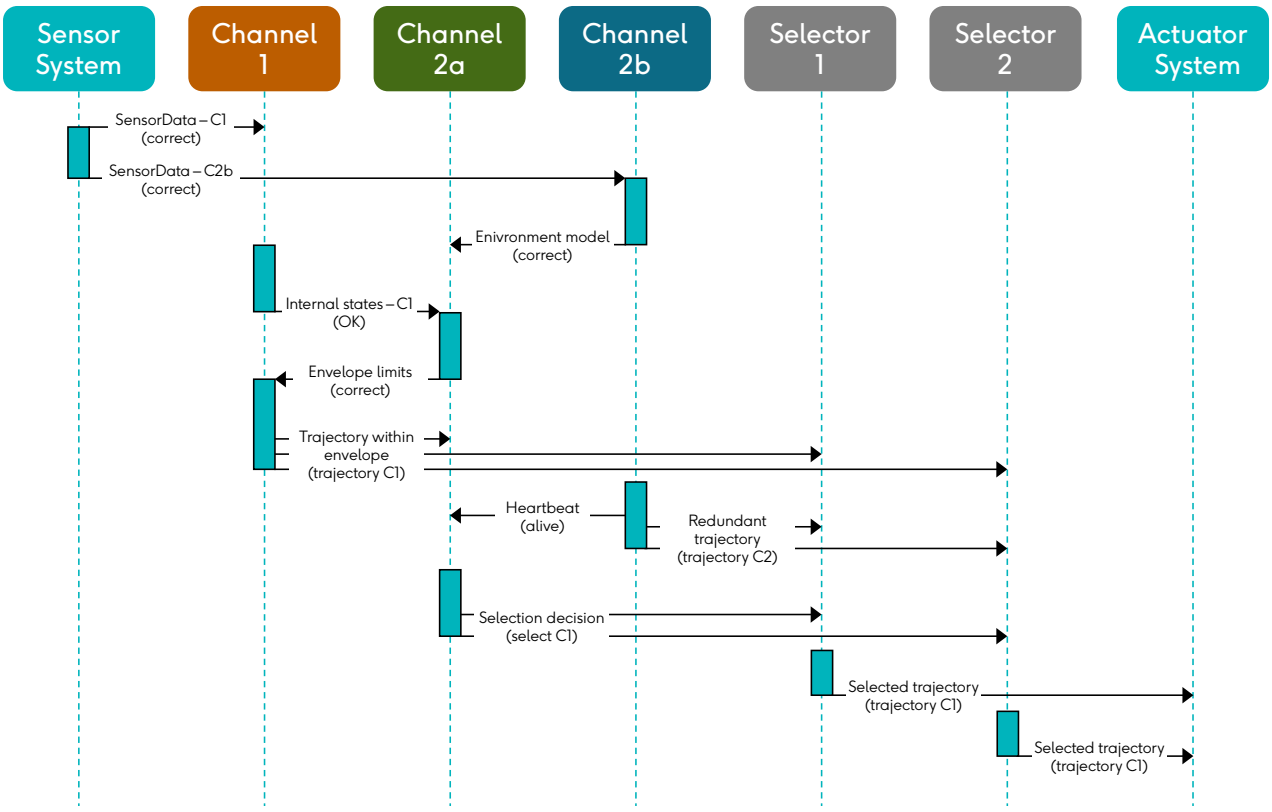


Figure 32: Sequence diagram of the AD-EYE architecture. The nominal case without faults or functional insufficiencies is shown.

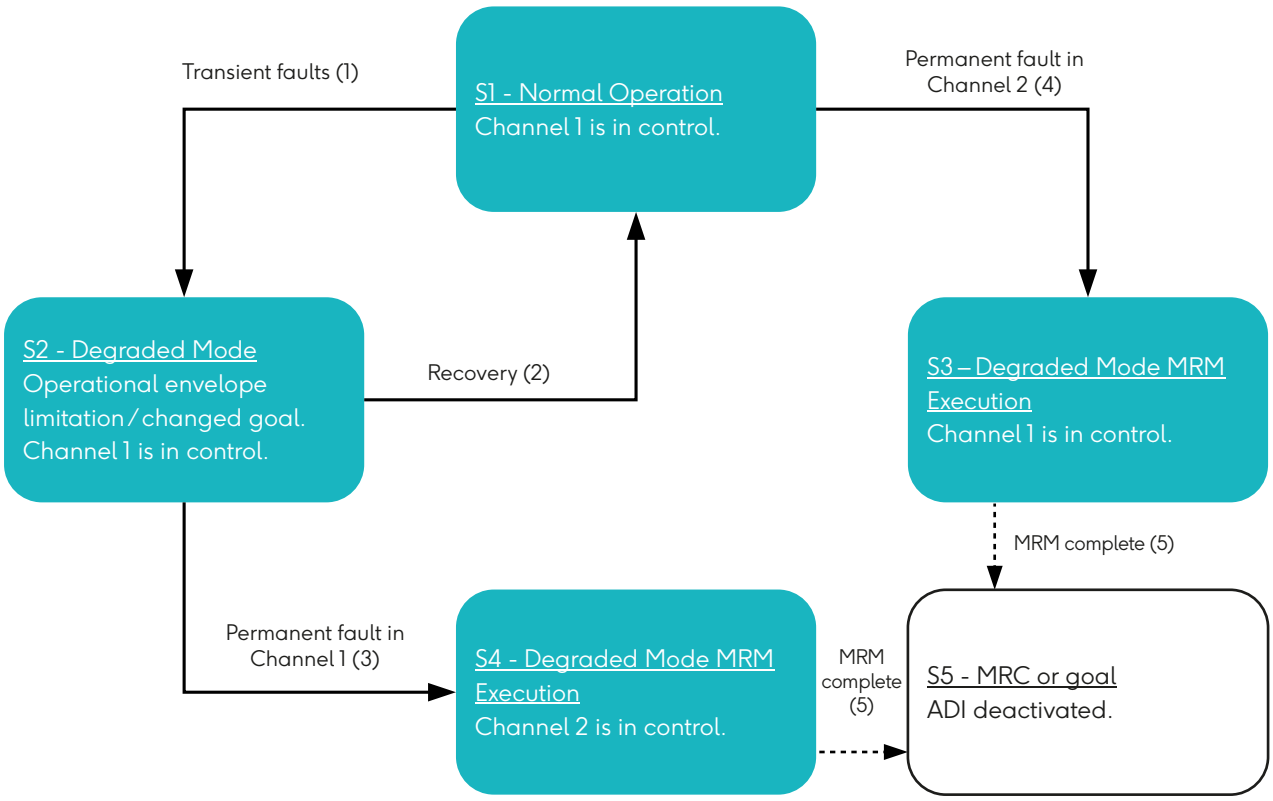


Figure 33: State diagram of the AD-EYE architecture.

3.6 RELATED EXAMPLES FROM THE INDUSTRY

3.6.1 AUDI ZFAS

In 2015, Audi gave some insights into their then next-generation HW and SW platform "zentrales Fahrerassistenzsteuergerät[45] (zFAS)" [53], intended to cover more complex ADAS use cases and a novel SAE L3 Traffic Jam Pilot AD use case[46].

Historically, most ADAS were a collection of dedicated ECUs and sensors for every individual function or use case, e.g., Adaptive Cruise Control (ACC), Light Assistant, Parking Assistant, Top View, etc. As more such systems were added to vehicles, cost and complexity scaled poorly and the performance of the functions provided remained limited, since fusing the information from the distributed sensors proved to be difficult.

Starting around 2015, the leading players in the automotive industry started working on more centralized platforms that decoupled specific sensors from specific ADAS functions by introducing a centralized sensor data fusion layer in-between, with the intention to gain several benefits:

- Multiple control units could be integrated into one unit and their HW resources shared.
- A more modular architecture (due to decoupling SW from HW) could allow updating functions or deploying additional ones over the lifetime of the vehicle.
- The central environment model could reduce redundancies and make consistent information available to many applications.

[45] German for "Central Driver Assistance Controller".
[46] This use case was intended to support hands-off / feet-off / eyes-off driving in traffic jam scenarios (up to 60 km/h) on highways. In case of a failure of the system, the driver was supposed to take over within ~10 seconds.

- Improved recognition of the vehicle's surroundings and a more detailed environment model gained through multi-sensor data fusion could support more complex SAE L2 (ADAS) use cases and even novel SAE L3 use cases.

At the time, developing an integrated HW and SW platform (see Figure 34) capable of hosting a large number of applications with widely varying computational needs (e.g., FPGA or GPU) was challenging. Table 14 shows an overview of the different HW components in the Audi zFAS system and the hosted functions.

Figure 34: ADAS domain controller ECU in the Audi zFAS platform [54].

Table 14: HW components in the Audi zFAS platform [55].

Automotive-qualified embedded microcontroller	• Various functions (up to ASIL D) • Interface to the rest of the vehicle
FPGA	• Sensor fusion • Sensor processing
Image processing SoC	• Image processing • Computer vision • Driver monitoring
Front camera image processing SoC	• Computer vision • Emergency braking

From a conceptual architecture perspective, the zFAS architecture can be considered a Single-Channel architecture (see section 3.3.1). For the initial underlying use case of a Traffic Jam Pilot (low speed and a restrained environment), the safety requirements are assumedly different from most AD use cases with respect to integrity (i.e., complex functionality likely does not need to reach the highest ASIL) and availability (i.e., the system likely does not need to provide complex fallback functionality in case of a fault).

3.6.2 TESLA FULL SELF-DRIVING SYSTEM

Tesla's "Full Self Driving" (FSD) seems to be a more recent implementation of a Single-Channel architecture. Despite the suggestive label and marketing as a highly autonomous system, FSD and its precursors "Autopilot" and "Enhanced Autopilot" are formally sold as SAE L2 systems, where the driver needs to supervise and be ready to take control of the vehicle at any time. While the Autopilot function is meant as a (semi-)autonomous highway driving system, the FSD system aims to include urban roads. Technical information is made available by Tesla in the course of yearly "AI Days", e.g., [56] and shows in considerable detail that the system is (at the time of writing) purely camera-based and composed of multiple complex machine learning modules that are specialized in various elements of the world model (objects, lanes, ...). A single, common planning module on top is responsible for computing the actual vehicle trajectory and controlling the vehicle motion. There is no mention of any functionally redundant blocks like supervision, or of fault-tolerance mechanisms like comparisons or voting on the SW architecture level.

With the introduction of the so-called "HW3" generation of the central driving computer, Tesla deployed the core SOC twice, in a parallel redundant fashion, stating that if either one were to fail, the redundant component would take over. It remains unclear, however, if that redundancy is exploited on a functional and logical architecture level – quite likely the same Single-Channel FSD stack is essentially intended to be deployed twice, and the redundant SOC is meant to just address random hardware faults in the underlying electronic components, like core SOC failures and power supply outages, but not functional deficiencies or systematic implementation faults. Therefore, the architecture may still be considered as monolithic single channel logically. Unconfirmed information [57] states that redundancy in HW3 has ultimately been dismissed in favor of using the second SOC to instead increase computational performance.

3.6.3 BMW SCALABLE AV PLATFORM ARCHITECTURE

In 2020, BMW unveiled some details on their (at the time) planned scalable AV platform architecture [58] [59], intended for SAE L3 AD features such as a Highway Pilot system similar to the one outlined in section 1.1. The published materials include an overview of the (at the time) planned HW architectures for different offering levels for the then-planned SOP 2021 (see Figure 35), as well as a conceptual system architecture for the SAE L3 system, dubbed hPAD (see Figure 36).

Based on the structural description in these materials, the conceptual architecture proposed by BMW shares similarities with the Channel-Wise Doer/Checker/Fallback architecture. Going by the depicted subsystems and high-level functional blocks, the "MAIN" channel appears similar to the "Primary Driving System", the "SAFE" channel similar to the "Monitoring System", and the "SAFE fail-degraded" channel similar to the "Fallback System". However, the "SAFE" channel also seems to produce trajectories and some cross-checking between "MAIN" and "SAFE" appears to occur. This is similar to the Cross-Checking Pair architecture.

As the published behavioral description is incomplete, we decided not to include this candidate conceptual system architecture in our evaluation. This avoids proceeding based on speculation and assumption.

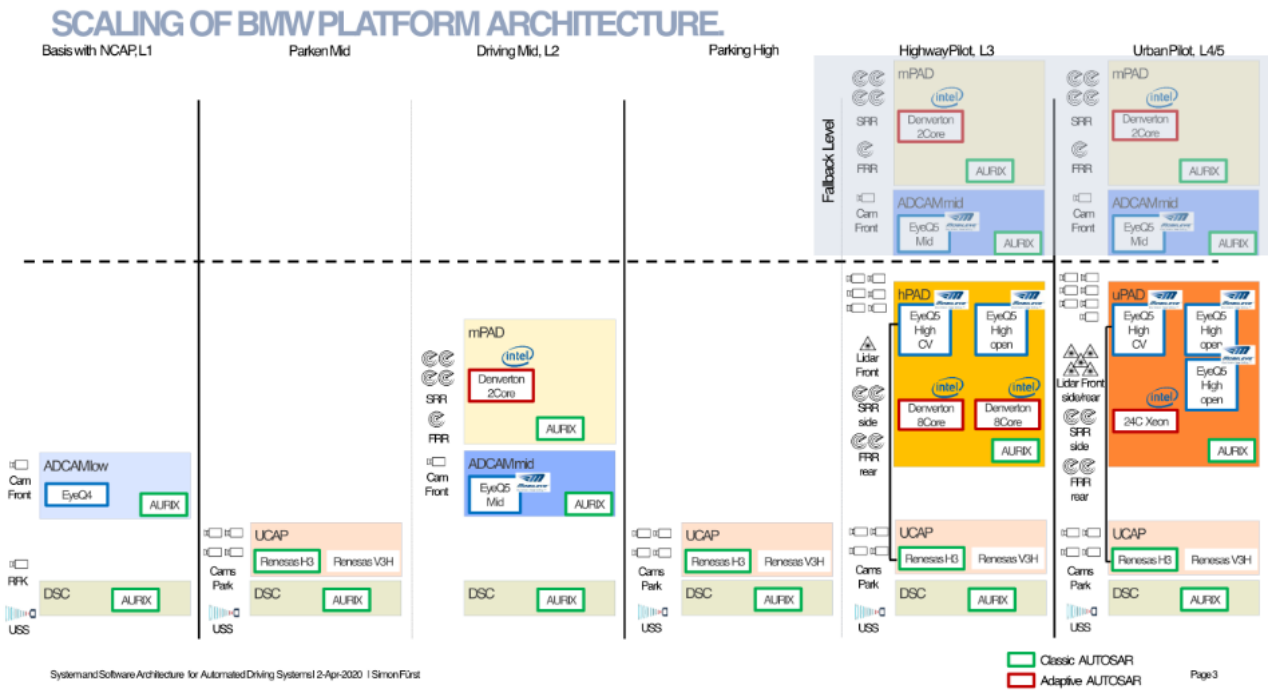


Figure 35: (At the time) planned HW architectures for different offering levels as proposed by BMW [59].

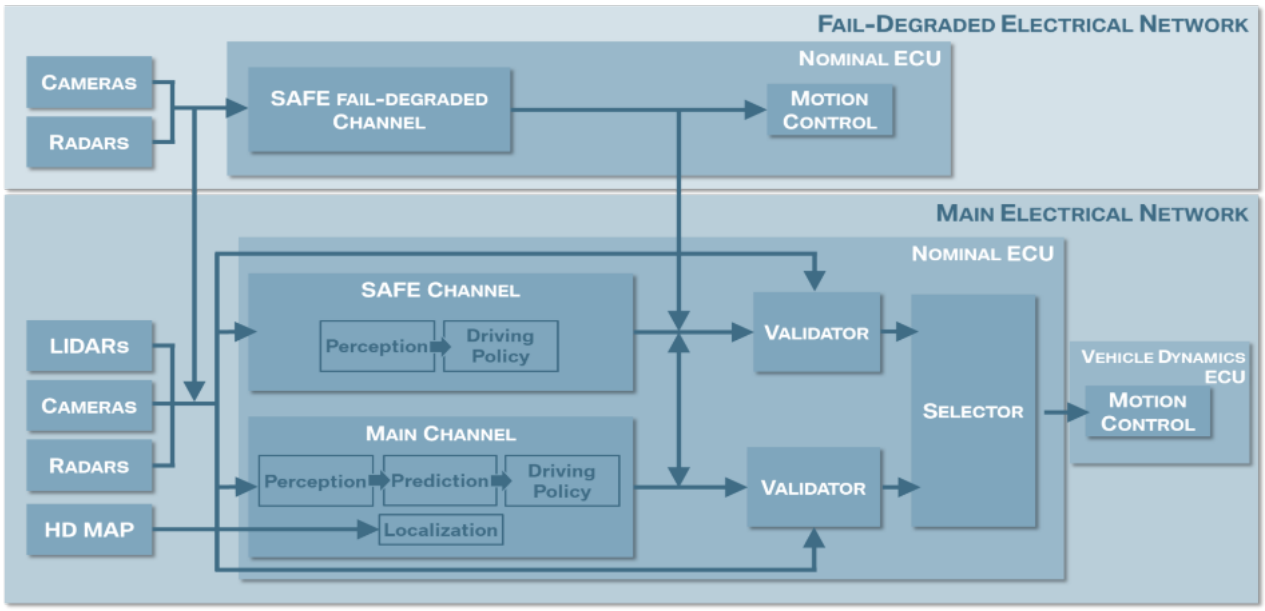


Figure 36: Conceptual architecture proposed by BMW for an SAE L3 system [58].

3.6.4 MOBILEYE SAFETY ARCHITECTURE AND TRUE REDUNDANCY

In 2024, Mobileye revealed more details about their architecture for self-driving cars [60]. Previously, Mobileye had proposed a high-level design [61] wherein each channel was based on a different sensor modality. This was intended to facilitate sufficient independence between channels (dubbed "True Redundancy"). In addition, Mobileye proposed Responsibility Sensitive Safety (RSS), a concept for checking the output of a (potentially AI-based) channel based on kinematic rules [62].

This architecture is based on the Primary/Guardian/Fallback pattern, equivalent to Doer/Checker/Fallback. Similar to the C-DCF architecture, it consists of three main channels:

- The Primary channel performs the nominal functionality. It consists of a compound AI system with perception blocks (for each sensor modality) and a policy block.
- The Guardian channel checks the Primary's outputs. It consists of independent RSS blocks for each sensor modality and Checker blocks for other functionalities, e.g., lane semantics or traffic lights.
- The Fallback channel performs the degraded functionality. It consists of an end-to-end AI system.

There are a few differences to the C-DCF architecture:

- The Primary and Guardian channels appear to share the same sensor-specific perception blocks, whereas in other architectures subsystems may share sensors, but not parts of the perception layer. The True Redundancy concept could be considered a potential practical solution to ensuring sufficient independence between the subsystems from a sensor perspective. At least for "physical obstacles" (to distinguish from lanes and traffic lights), the Guardian uses an independent RSS block per sensor modality and votes on the final "safe/unsafe" outcome.
- If the Guardian considers the Primary unsafe, it switches to the Fallback. If it also considers the Fallback unsafe, it additionally enforces MRM braking.

3.6.5 MERCEDES DRIVE PILOT

The Mercedes DRIVE PILOT system represents the first commercially available SAE Level 3 Traffic Jam Pilot (TJP) and is certified in Germany and some US states. It comes with strict operational limitations: activation is only permitted on highways when a lead vehicle is present and the speed is below 60 km/h. Mercedes is gradually expanding the operational design domain (ODD), e.g., the speed limit in Germany has recently been extended to 90 km/h, but the system remains focused on controlled highway environments with clear restrictions on speed and lane usage.

From an architectural perspective, publicly available information on DRIVE PILOT's internal redundancy management is limited. However, conference presentations [63] and technical disclosures indicate that the system employs a combination of redundant sensors and actuators, as well as a central AD ECU responsible for trajectory planning, including MRMs in case of faults or ODD violations. This is likely intended to ensure that the system can bring the vehicle to a safe stop even when a sensor, processing element, or actuator fails. Given that only a single central ECU is mentioned, this might involve executing a pre-planned MRM or similar.

4 ARCHITECTURE EVALUATION

4.1 EVALUATION PROCESS

Section 3 presented architectures that strive to be practical solutions to the question of how, conceptually, an automated driving architecture should be designed. The architectures are not limited to a specific use case of automated driving, and for the most part they do not explicitly target a specific design criterion, like those described in section 2, although without doubt the safety and availability of the ADI was a key consideration in the design of most candidate architectures.

In this section we seek to describe the merits or potential drawbacks each architecture might show with respect to the evaluation criteria. To form an unbiased basis for the evaluation, we first start with a generic evaluation of each architecture in section 4.2, by listing a number of observations related to each criterion, i.e., properties of each architecture perceived by the Safety and Architecture Working Group team and (if not obvious) their significance for the specific criterion.

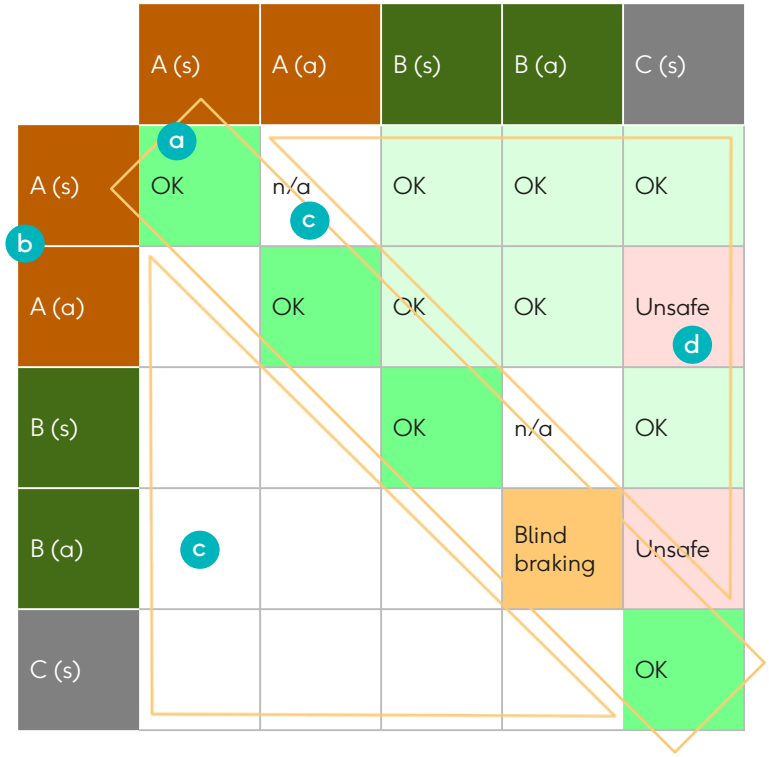
As the second step, we give the concrete evaluation of the architectures under the defined reference use case of an SAE L4 Highway Pilot function in section 1.1. To this end, we evaluate the significance of each criterion for that use case – as some will be must-haves for the conceptual architecture, while others might be of lesser significance or merely nice to have. Next, we directly compare the architectures, considering the observed properties from the generic evaluation and inferring merits or weaknesses with respect to each evaluation criterion, and finally ranking them under the criterion.

While some findings may be of principle nature and not easy to overcome, for others the conceptual nature of the architectures and the high level of their descriptions may leave room to define countermeasures against weaknesses in a further, more detailed design step. Also, depending on the particular use case and environment, the relative significance of the evaluation criteria may change, and criteria might be modified and/or added. It is therefore important to emphasize that the evaluation that we provide is not intended as an absolute and final judgement. Rather, it may be understood as a blueprint for the readers of this report for how to analyze an architecture, identify deficiencies, and derive improvement measures in a systematic way.

4.2 GENERIC EVALUATION

The generic evaluation of each proposed candidate conceptual system architecture with respect to the defined architecture evaluation criteria relies on tables and diagrams explained below, to allow for better comparison later on:

- Figure 37 explains how to read the tables discussing all cases where one or two subsystems encounter a fault or output insufficiency.
- Figure 38 explains how to read the sequence diagrams illustrating different failure scenarios.



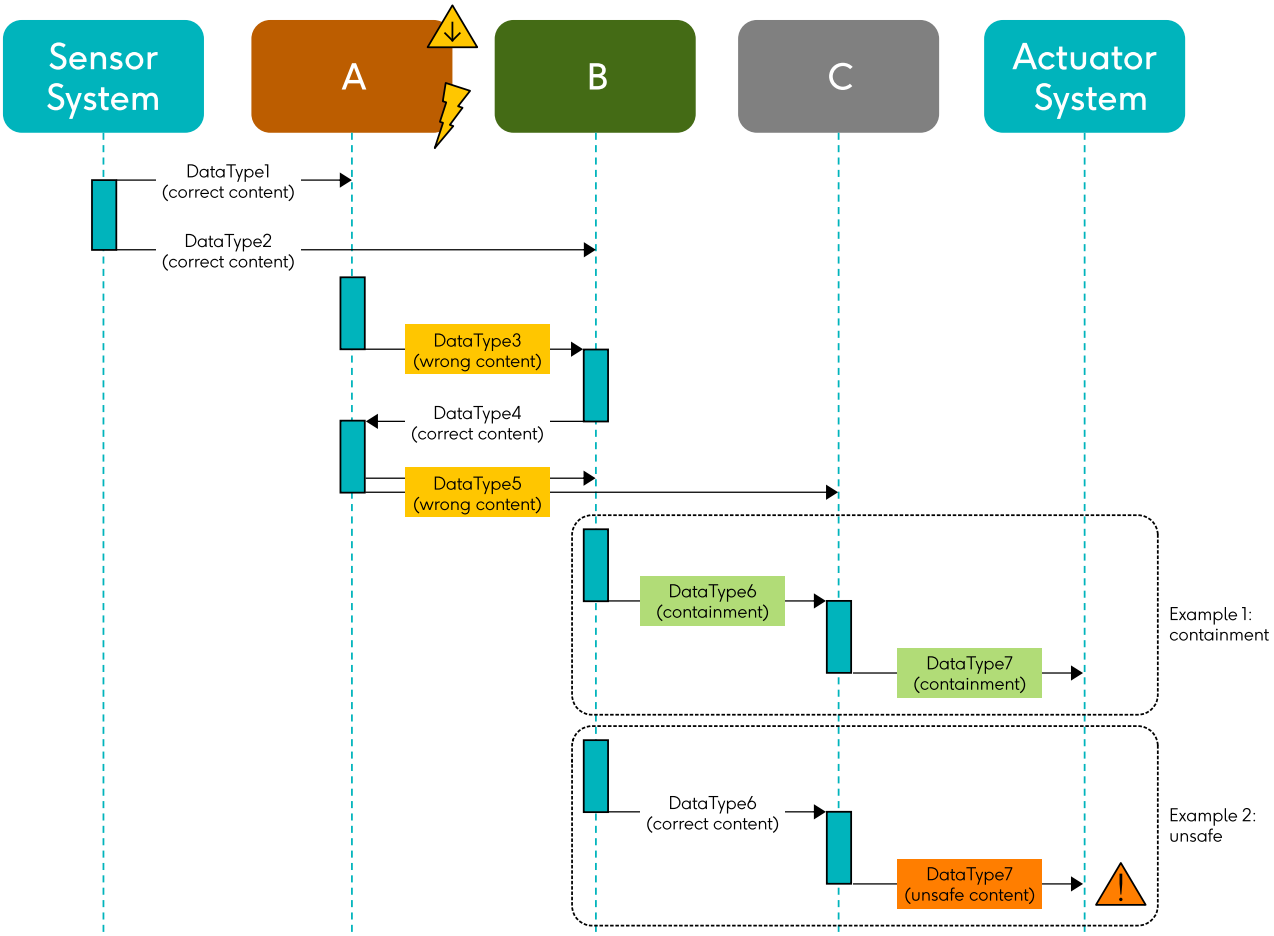
a. Diagonal elements indicate scenarios where only a single subsystem encounters a fault or functional insufficiency. The field is marked green if at least a dynamic/reactive MRM can be provided, orange if at least blind braking, and red if unsafe. Shown example: scenario where subsystem A is fail-silent.

b. The failure mode of each subsystem is indicated with either fail-silent (s) or fail-arbitrary (a). Some simple subsystems can be constrained to fail-silent failure modes.

c. Some combinations are not applicable (colored white). Scenarios below the diagonal are mirrored and not filled out (colored white).

d. Off-diagonal elements indicate scenarios where two subsystem encounter faults or functional insufficiencies. Covering such dual-point faults is "nice to have": the field is marked light green if at least a blind MRM can be provided and light red if unsafe. Shown example: scenario where subsystem A is fail-arbitrary and subsystem C is fail-silent.

Figure 37: Explanation of the failure scenario tables used in the generic evaluation.



a. Sequence diagrams in the architecture evaluation mostly show scenarios with faults (indicated by a lightning bolt) or functional insufficiencies (indicated by a triangle with arrow pointing down).

b. The transmission of faulty outputs between subsystems is highlighted in yellow.

c. If faulty outputs can be contained by other subsystems, e.g., via checking, this is highlighted in green.

d. Unsafe outputs that reach the ADI boundary are highlighted in orange and indicated by a triangle with an exclamation mark.

Figure 38: Explanation of the failure scenario sequence diagrams used in the generic evaluation.

4.2.1 EVALUATION OF THE SINGLE-CHANNEL ARCHITECTURE

4.2.1.1 AVAILABILITY

Availability of the system

This is a monolithic architecture. Its ability to provide the DDT under nominal or failure conditions depends on the availability of a single FCU with interfaces identical to those of the ADI.

- As shown in Figure 39 and Table 15, a single fault or output insufficiency leads to immediate unavailability of the ADI (if it is detected and the ADI goes fail-silent) or unsafe output (if it is not detected). To mitigate (but not eliminate) the vulnerability to single-point faults, redundancy measures within the single subsystem could be attempted.
- Dual- or multi-point faults are not addressed by this architecture.

Table 15: Possible fault / output insufficiency scenarios for the Single-Channel architecture, considering silent (s) and arbitrary (a) failure modes. Diagonal elements: single-point faults ("OK" marks an MRM or better). Off-diagonal elements: dual-point faults ("OK" marks blind braking or better).

	C (s)	C (a)
C (s)	Unavailable (unsafe)	n/a
C (a)		Incorrect (unsafe)

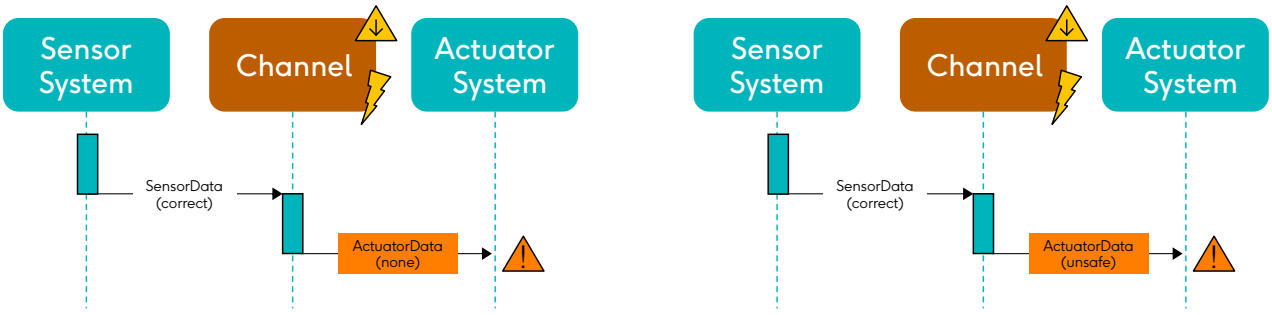


Figure 39: Sequence diagram of the Single-Channel architecture. Two cases with a fault or output insufficiency (detected by the single channel in the left panel; undetected in the right panel) are shown. Unsafe messages are highlighted in yellow.

Diagnostics scheme

This architecture does not use a diagnostics scheme. If a fault is detected in the single subsystem, it remains fail-silent.

- There are no other subsystems to inform.
- There are no degradation measures triggered upon the detection of faults.

Degradation scheme

This architecture does not use a degradation scheme. If a fault is detected in the single subsystem, it remains fail-silent.

- Any error in the ADI will immediately negatively impact the system functionality.
- No degradation is possible, i.e., no complex fallback functionality (MRM) is provided. No standby fallback system exists.

4.2.1.2 NOMINAL FUNCTIONALITY

Availability of nominal function

A single trajectory is generated from a single world model.

- This architecture only relies on self-diagnostics, i.e., there are no other mechanisms that could lead to "false positives". Considering the smaller HW and SW footprint, its base failure rate (without compensation due to redundancy) may also be lower. The absence of a redundant subsystem cross-checking the outputs makes this architecture prone to false negatives; functional insufficiencies also cannot be compensated for. The lack of fallback puts pressure on the complex functionality to reach the highest possible ASIL, which is very hard to achieve for a monolithic system (see also G1: Design faults in large and complex monolithic systems).
- Without redundancy or fallback options, any fault or output insufficiency will be immediately noticeable and/or potentially unsafe.
- There is no arbitration in this architecture.
- The decision logic of the single channel is very simple.

4.2.1.3 CYBERSECURITY

Interactions between subsystems

This is a monolithic architecture, i.e., the correctness and availability of the entire DDT depends on a single channel. If this channel is compromised, the entire DDT is jeopardized.

- There are no interfaces between subsystems.
- There are no subsystems to exchange data with.
- There are no interactions between subsystems.

Interactions with external systems

Communication with external systems can be expected.

- The single channel needs to communicate with external systems.
- The single channel will likely need constant connectivity to access HD maps or similar. It will also require regular updates, which implies OTA. This makes the usage of additional, potentially more secure, update mechanisms (e.g., in the workshop vs. OTA) less feasible.
- Compromising the single channel allows for full control over the entire ADI.

4.2.1.4 SCALABILITY

Scalability towards higher availability

This architecture does not consider scalability.

- The only way to improve the correctness and/or availability of the ADI is to improve the single channel. This is limited by G1: Design faults in large and complex monolithic systems and G2: Single-event upsets in non-redundant HW.
- No subsystems can be added to this architecture.

Scalability towards differing offering levels

This architecture does not support re-using existing subsystems.

- This single channel could be partially reused for more complex AD architectures, e.g., as a fallback channel. Existing Single-Channel ADAS could, without architectural changes, not be turned into an ADI.
- This architecture is not partitioned into subsystems that might be used as scalable options.

4.2.1.5 SIMPLICITY

Number, complexity, and performance of subsystems

Superficially, this architecture appears simple, but the implementation complexity could be significant.

- There is only a single channel in this architecture.
- The single channel can be very complex. If it also needs to reach the highest possible ASIL to ensure correctness, its development costs could be very high.
- The single channel has high performance requirements.

Required level of diversity

This is a monolithic architecture without clearly separated subsystems.

- There are no other channels. However, the internal structure of the single channel may need to be more complex and may involve internal redundancy and diversity, which would actually be one of the other architectures "hidden" inside the single channel.
- There are no other subsystems to integrate or develop independently.

Complexity of validation

This is a monolithic architecture without clearly separated subsystems to validate and verify independently.

- The single channel can only be validated as a whole. As this is a very complex subsystem, efforts for review, testing, and analysis could be higher than for a partitioned system.
- No independent validation of different subsystems is necessary. It is also not necessary to perform integration verification.
- There are no other subsystems to investigate for correlated or common cause failures.

4.2.1.6 SAFETY OF THE INTENDED FUNCTIONALITY

Support to accommodate functional insufficiencies

In a monolithic architecture, there are no other subsystems to compensate for functional insufficiencies of the single channel.

- A single channel that shall address all SOTIF requirements appears to be impractical for SAE L4 systems unless the ODD is very restricted. To reach the very high integrity requirements and ensure correctness, it may require more analysis and testing efforts, independently of the use case because of its internal complexity compared to other architectures.
- There are no other subsystems that could employ diverse algorithms or functionality, particularly to compensate for AI/ML weaknesses.
- The single channel may use several sensor modalities. Depending on the type of failure and the FCU design, a fallback trajectory may or may not be available. Such a fallback trajectory could be derived from the last nominal one or be predefined.

Support to manage operational conditions

In a monolithic architecture, all responsibilities regarding operational conditions lie with the single channel.

- The single channel providing the functionality must also monitor the ODD.
- It is assumed that the ODD monitoring is part of the single channel. If it fails, a pre-defined MRM must be used, which may be inadequate for the operational conditions. As this MRM would be generated by the same single channel, no independence would be gained.
- No architectural support for monitoring safety performance is provided by this architecture.

4.2.2 EVALUATION OF THE MAJORITY VOTING ARCHITECTURE

4.2.2.1 AVAILABILITY

Availability of the system

The Majority Voting architecture might struggle to provide the high availability necessary for AD. While homogeneous redundancy (i.e., identical channels) is susceptible to common cause faults and functional insufficiencies, heterogeneous redundancy (i.e., diverse channels) will have to implement inexact voting. If the outputs of the channels differ too much, which can happen even in the nominal case, the lack of agreement forces the ADI to select one (potentially deficient) channel or to go silent.

- This architecture can maintain integrity and availability in the presence of a single fault or output insufficiency, but only if at least two channels work properly and agree on the course of action (see Figure 41). Figure 41 and Table 16 show that for heterogeneous channels a single fault or output insufficiency can lead to disagreement. Depending on the implementation, this leads to either no output (no availability), preferring one of the remaining channels (potentially unsafe), or resorting to a buffered MRM.
- Common cause faults in the channels need to be avoided. This makes diversity between the channels crucial, complicating the voting by necessitating the implementation of inexact voting. Developing three diverse channels providing similar (nominal) functionality may be very challenging. Systematic faults in the Voters need to be avoided. This should be accomplishable as these are relatively simple. For homogeneous channels, systematic faults pose a major problem, i.e., all channels may continue operating but produce incorrect output.
- If the inexact voting does not lead to a majority, e.g., because the channels propose fundamentally different courses of action like evading to the left vs. to the right, the ADI remains silent (see Figure 41). This is not a safe state. A failure of one of the Voters is safe, though.

Table 16: Possible fault / output insufficiency scenarios for the Majority Voting architecture, considering silent (s) and arbitrary (a) failure modes. Diagonal elements: single-point faults ("OK" marks an MRM or better). Off-diagonal elements: dual-point faults ("OK" marks blind braking or better). For space reasons, (*) indicates "remaining channels may disagree, leading to either silence (unsafe), the preference of a channel (potentially unsafe), or a pre-planned MRM", and (**) indicates "no majority, leading to either silence (unsafe), the preference of a channel (potentially unsafe), or a pre-planned MRM".

	C1 (s)	C1 (a)	C2 (s)	C2 (a)	C3 (s)	C3 (a)	V1 (s)	V2 (s)
C1 (s)	Remaining channels may disagree (MRM or unsafe) (*)	n/a	OK	No majority (MRM or unsafe) (**)	OK	(**)	(*)	(*)
C1 (a)		(*)	(**)	Incorrect (unsafe)	(**)	Incorrect (unsafe)	(*)	(*)
C2 (s)			(*)	n/a	OK	(**)	(*)	(*)
C2 (a)				(*)	(**)	Incorrect (unsafe)	(*)	(*)
C3 (s)					(*)	n/a	(*)	(*)
C3 (a)						(*)	(*)	(*)
V1 (s)							(*)	Unavailable (unsafe)
V2 (s)								(*)

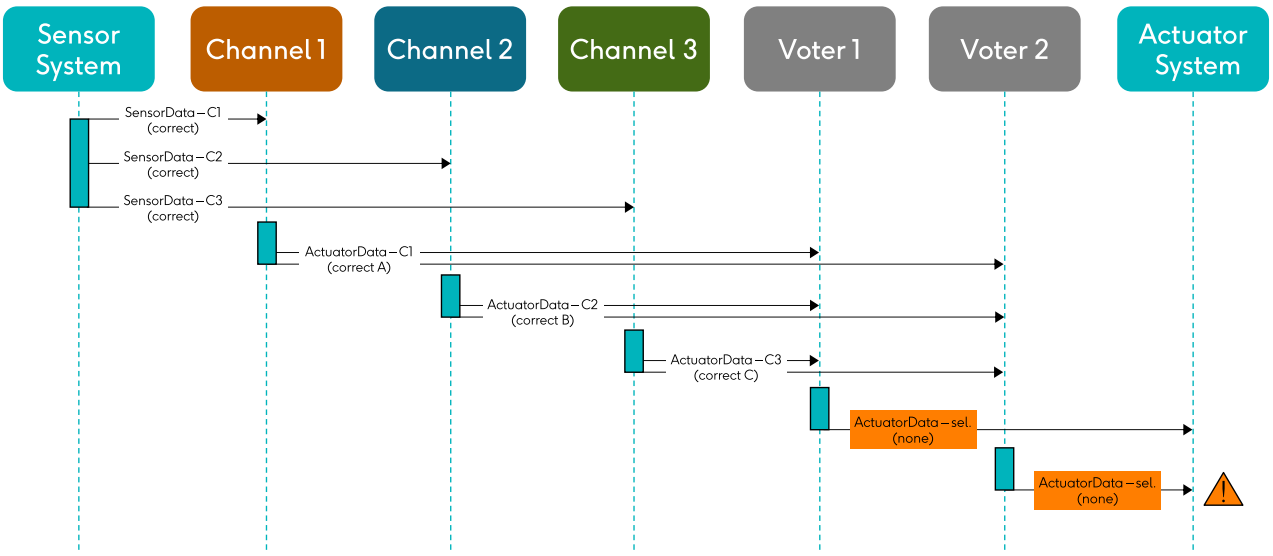


Figure 40: Sequence diagram of the Majority Voting architecture. The case without faults or functional insufficiencies, but disagreement between the channels is shown. Unsafe messages are highlighted in orange.

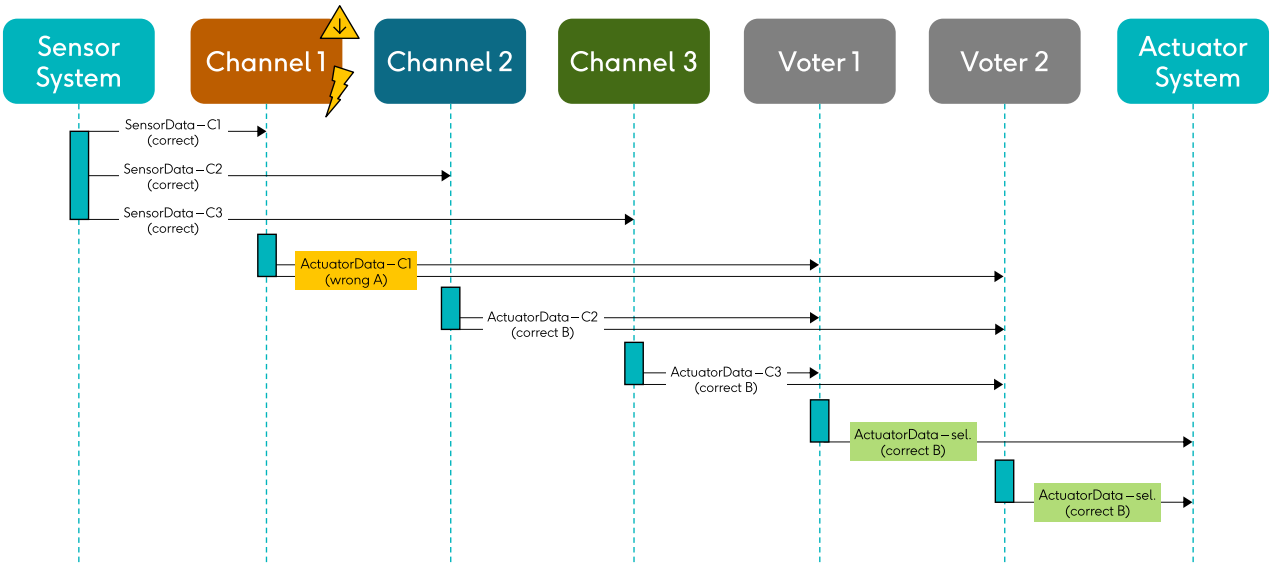


Figure 41: Sequence diagram of the Majority Voting architecture. The case with a fault or output insufficiency in Channel 1 is shown. Faulty messages are highlighted in yellow, fault containment in green.

Diagnostics scheme

This architecture does not employ a diagnostics scheme.

- The different subsystems are not aware of each other's condition.
- The different subsystems do not adapt their behavior based on faults elsewhere in the system.

Degradation scheme

This architecture does not employ a degradation scheme. Instead, all three channels provide the nominal functionality.

- A single error in the system is not immediately noticeable to the end user. For heterogeneous channels, disagreements between the channels, e.g., if they propose fundamentally different courses of action (e.g., brake, steer left, steer right), need to be carefully considered. The Voter may then either switch to a fail-silent state, raise a flag to the Diagnostics System, or choose an output based on a pre-defined policy (e.g., prefer one of the channels or resort to a buffered or fixed MRM trajectory). Similar strategies can also be employed for homogeneous channels if no majority can be found.
- This architecture does not foresee any levels of degradation. The Voter may, depending on the implementation, consider different options if no majority can be found, though this is not degradation of the channels.

4.2.2.2 NOMINAL FUNCTIONALITY

Availability of nominal functionality

Majority Voting generally works best if there is a straightforward definition of "same or sufficiently similar output". This is simple for binary output but gets complicated for trajectories, which can vary both fundamentally (e.g., turn left, turn right, or continue straight ahead) and gradually (e.g., 100 km/h vs. 101 km/h)[47].

- Even with inexact voting, disagreements between three diverse channels can be common[48] and the architecture is sensitive to these. Even in structured traffic, e.g., highway driving, many different maneuvers are possible in any given situation, i.e., there is no obvious best or correct solution.
- Frequent disengagements due to disagreement between the three diverse channels could lead to nuisances and even unsafe situations.
- While majority voting on simple outputs is straight-forward, inexact voting on more complex outputs like trajectories or actuator control commands is not. These data structures have at least 2 spatial and a temporal dimension, making the definition of similarity between them non-trivial.
- The main complication lies in defining the similarity between outputs and the involved thresholds, i.e., "how similar is similar enough?". Once such a decision has been made (binary "similar" vs. "non-similar"), the arbitration algorithm is simple: an output that is similar to both other outputs is preferred over an output that is only similar to one other output. If no two outputs are similar, the ADI remains silent.

[47] As mentioned earlier, voting patterns can still be useful on a lower level if decisions are binary, e.g., voting on the existence of an object based on sensor modalities.
[48] Even identical replicas can produce different outputs due to minute timing differences. This is known as replica indeterminism. Periodically produced outputs can best be compared if all channels use the same cycle time, but even in this case small drifts can easily lead to inconsistencies.

4.2.2.3 CYBERSECURITY

Interactions between subsystems

The complex subsystems do not interact directly. This can make it harder to corrupt multiple channels after an attack on one of them, i.e., a security incident can only occur if there are multiple attacks on different channels. For homogeneous channels, the same vulnerability could be exploited for such an attack.

- The three complex subsystems do not directly interact with each other. Communication with the two simple Voter subsystems is unidirectional. The Voters are simple and reliable, thus assumed to be harder to attack.
- Only small data structures, i.e., trajectories and/or actuator control commands, are exchanged.
- The interfaces between subsystems are well defined and can be restricted.

Interactions with external systems

While all subsystems in this architecture will need updates, the subsystems are relatively loosely coupled. This could allow the use of different and potentially more secure update mechanisms, making it harder to compromise the ADI with a single attack.

- As all three complex subsystems provide the nominal functionality, they will require communication with off-board systems.
- At least one of the three channels will likely need constant connectivity to access HD maps or similar.
- All three channels will likely require regular updates, though not necessarily at the same time due to their loose coupling. This implies OTA and thus a larger attack surface for gaining remote access to the system. The Voters may only need to be updated rarely and could use a more secure update mechanism, though this depends on how robust inexact voting is, i.e., if it needs adjustments over time.

4.2.2.4 SCALABILITY

Scalability towards higher availability

The Majority Voting architecture is scalable in a straightforward way. The Voters can be modified easily to accommodate additional channels in the voting policy.

- More channels could be added to improve availability of the system. However, in case of heterogeneous (diverse) channels this also increases the likelihood of disagreements between channels, which is detrimental to availability.
- If this architecture is based on diverse channels providing the nominal functionality, adding more channels not only increases production cost (another high-performance chip or ECU) but also significantly increases development cost (another diverse channel).
- Adding more channels is most useful. Additional Voters could also be added, but this is of limited use.

Scalability towards different offering levels

This architecture could largely be based on existing systems. However, it may be challenging and expensive to provide several equally capable systems to use as channels and adapt them for inexact voting, especially so as diverse implementations will likely be required.

- An existing ADAS could be re-used as one of the channels.
- The Voters would need to be newly developed for the ADI.

4.2.2.5 SIMPLICITY

Number, complexity, and performance of subsystems

The Majority Voting architecture consists of a small number of subsystems. However, three of these are complex and will have high performance requirements. It may also be difficult to develop a suitable voting policy for heterogeneous channels.

- This architecture consists of five subsystems.
- The three channels will likely involve ML/AI, with associated high performance requirements and HW cost. These channels run in parallel and have no dependency on each other. The functionality of each channel can be modified separately.
- For homogeneous channels, the development cost is comparable to the Single-Channel architecture. For heterogeneous channels, which are likely needed to prevent common cause faults, it scales with the number of channels.
- The two Voters are simple and can be implemented with high integrity. They only add a small delay to the end-to-end response time from reading the input signal to sending the output signal. Developing a sound voting policy to address the problems mentioned earlier can be challenging.

Required level of diversity

This architecture will likely require diversity between very complex subsystems (the channels to be voted on) to avoid common cause faults. For homogeneous channels, no diversity is necessary.

- Due to their similar functionality, it may be challenging to find diverse approaches.
- All channels shall provide the nominal functionality, which might make it hard to find different and equally capable suppliers.

Only the Voters are relatively simple and can likely be implemented with the highest integrity levels to avoid systematic faults.

Complexity of validation

Validation of the Majority Voting architecture is complicated due to its symmetry, i.e., correlated faults or correlated functional insufficiencies in two or more channels are immediately relevant.

- The three channels can be validated independently as they have no direct interactions. However, validation of the voting itself can be non-trivial.
- Under the assumption that channels are independent, the validation target (in terms of residual fault probability) can be lowered significantly for each channel.
- Proving that the channels are independent may require very high efforts as it involves all three channels (with the same role) simultaneously.

4.2.2.6 SAFETY OF THE INTENDED FUNCTIONALITY

Support to accommodate functional insufficiencies

Diverse channels can compensate for each other's functional insufficiencies. However, disagreements between channels remain hard to resolve.

- An output insufficiency in one channel can be compensated for by the other two channels if these are sufficiently independent. For most ODDs, multiple acceptable solutions to trajectory planning and actuator setpoint generation exist. This can lead to disagreements between diverse channels.
- The diversity of the three channels should provide some robustness to functional insufficiencies.
- No specific strategy for defining independent sensor sets composed of different sensor modalities is defined for this architecture. If sensors are shared between the channels, which can be expected due to cost reasons, their independence can be compromised. Additional analysis and rationales will be needed to argue that independence is maintained.

Support to manage operational conditions

This architecture does not define specific ways to handle the ODD in a centralized way. It has very limited capability to react to changing operational conditions and could have difficulties handling operational modes other than the nominal one.

- Each channel has to monitor the ODD separately. No central mechanism for deciding the reaction to an ODD exit or similar is foreseen.
- When an output insufficiency occurs, one of the other channels takes over control of the vehicle. As all channels are capable of providing the nominal functionality, this is applicable for the entire foreseen ODD. However, the channels are not aware of each other's condition and can only adapt to a limited degree to degraded conditions.
- No specific way for monitoring safety performance is defined for this architecture. Monitoring the rate of disagreements could be useful.

4.2.3 EVALUATION OF THE CROSS-CHECKING PAIR ARCHITECTURE

4.2.3.1 AVAILABILITY

Availability of the system

The CCP architecture is only robust with respect to failure modes of the planning layer[49]. In the general case, however, a single fault or output insufficiency can silence the ADI, leading to a pre-planned MRM, which may be inadequate for high-speed AD use cases.

- In the presence of a single fault, this architecture remains available to a limited degree (see Table 17). If a fault or functional insufficiency "only" leads to an unsafe trajectory or actuator setpoint output (see Figure 42) of one channel, the other channel can continue controlling the vehicle.
- However, if the cross-checking functionality is also compromised (see Figure 43), the channels will disagree, and the Selectors must resort to the buffered MRM, e.g., blind braking using the last known curvature. Such a strongly degraded reaction should be considered unsafe in the presence of a single fault for high-speed AD use cases.
- A simultaneous failure of both channels leads to the buffered MRM. This is an acceptable degradation for a dual-point fault.

[49] We can assume that each channel can be internally divided into a perception layer (processing the sensor data into an environment model), a planning layer (processing the environment model into a trajectory and actuator setpoints), and a checking layer (validating the other channel's trajectory against the own channel's internal environment model).

Table 17: Possible fault / output insufficiency scenarios for the Cross-Checking Pair architecture, considering silent (s) and arbitrary (a) failure modes. Diagonal elements: single-point faults ("OK" marks an MRM or better). Off-diagonal elements: dual-point faults ("OK" marks blind braking or better).

	C1 (s)	C1 (a)	C2 (s)	C2 (a)	S1 (s)	S2 (s)
C1 (s)	OK	n/a	OK	Incorrect (unsafe)	OK	OK
C1 (a)		Blind braking	Incorrect (unsafe)	Incorrect (unsafe)	OK	OK
C2 (s)			OK	n/a	OK	OK
C2 (a)				Blind braking	OK	OK
S1 (s)					OK	OK
S2 (s)						OK

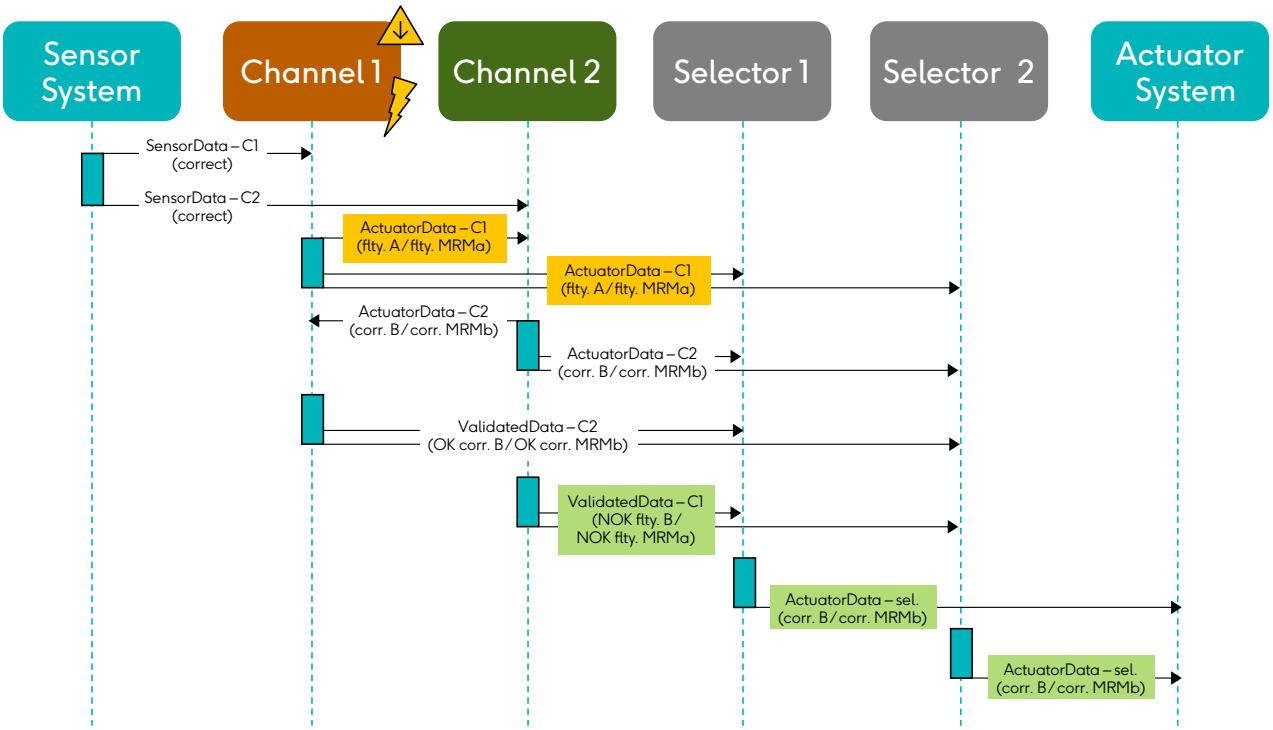


Figure 42: Sequence diagram of the CCP architecture. The case with a fault or output insufficiency in the planning layer of Channel 1 is shown. Faulty messages are highlighted in yellow, fault containment in green.

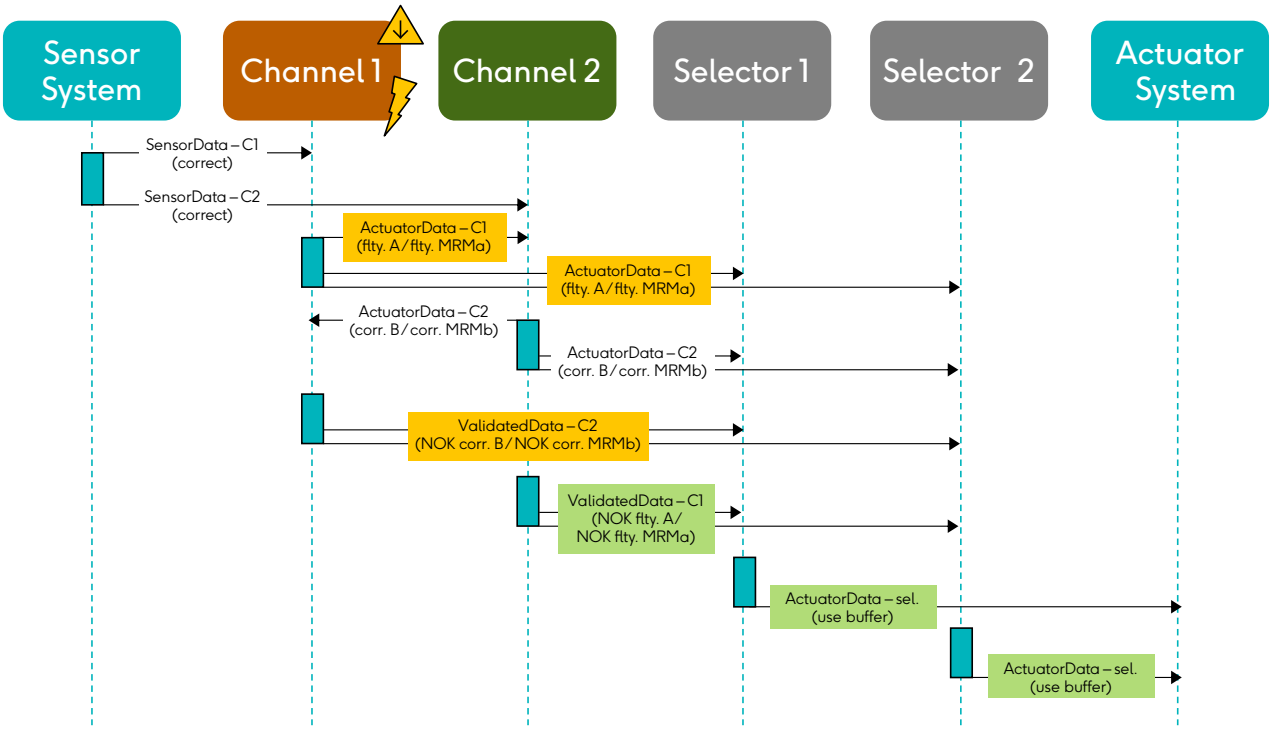


Figure 43: Sequence diagram of the CCP architecture. The case with a fault or output insufficiency in the perception layer of Channel 1 is shown. Faulty messages are highlighted in yellow, fault containment in green.

Diagnostics scheme

The direct cross-check between the two channels in this architecture can be used to adapt the ADI's behavior, e.g., by using degradation when faults or functional insufficiencies are detected in one of the channels. However, this only works for failure modes of the planning layer.

- The two channels are aware of each other's condition as they cross-check each other.
- Each of the channels could restrict itself to a degraded mode, e.g., more conservative driving at lower speed or even an MRM, if it detects that the other channel is faulty. However, this does not address that the other channel might still invalidate it.

Degradation scheme

The CCP architecture does not specify a degradation scheme

- An error such as in Figure 42 would not immediately impact the vehicle behavior. However, if the faulty channel does not recover quickly, the remaining channel should start the hand-over to the driver and/or restrict itself to degraded trajectories (pull over and come to a controlled stop). An error such as in Figure 43 would lead to an immediate MRM.
- The channels can only act more conservatively and initiate the actions mentioned above. Other levels of degradation are not possible. In general, a disagreement between the two channels is strongly degraded and not graceful.

4.2.3.2 NOMINAL FUNCTIONALITY

Availability of nominal functionality

While this architecture has equally capable cross-checking channels, a single fault or output insufficiency causes a noticeable loss of functionality.

- Since both channels provide the nominal functionality (and are thus roughly equally capable), false positives in the cross-check should be less likely. As long as a trajectory acceptable to both channels is generated, the architecture can compensate for a false positive.
- A single fault or output insufficiency can still immediately lead to a strongly degraded MRM and complete stop. This can pose a nuisance (blocked traffic) or even lead to unsafe situations.
- The arbitration in the Selectors is straightforward. The cross-check itself can be more complicated, i.e., similar to a Doer/Checker pair.
- The algorithms for the Selectors are very simple. They prefer actuator setpoints that have been found safe by both channels over those that have only been found safe by a single channel (if the second channel is silent). If neither of these applies, the Selectors remain silent and the Actuator System resorts to the buffered MRM.

4.2.3.3 CYBERSECURITY

Interactions between subsystems

The different subsystems in the CPP architecture almost all communicate with each other.

- There are interactions between all subsystems, except between the two Selectors.
- All interactions run once per execution cycle. The exchanged data structures are relatively small, i.e., trajectories and actuator setpoints.
- There are well-defined interfaces between subsystems.

Interactions with external systems

This architecture requires interactions with the back end for both channels.

- The two channels likely require continuous communication with the back end.
- Both channels provide the nominal functionality. As such, they will likely require constant connectivity to access HD maps or similar.
- Both channels will require regular updates (probably via OTA), though not necessarily at the same time. The two Selectors should not change significantly over time and may thus only rarely if ever need updates. Other, more secure update mechanisms could be used for this.

4.2.3.4 SCALABILITY

Scalability towards higher availability

Adding more subsystems to this architecture would make it very loosely resemble the Daruma architecture. For scalability, it is detrimental to include the cross-checking functionality inside the channels.

- This architecture could be extended by adding more channels. However, this also has an impact on existing channels, which need to perform additional cross-checks, which require additional interfaces.
- Adding more channels might make sense, adding more Selectors would not usefully improve the architecture.

Scalability towards different offering levels

This architecture does not consider the reuse of existing ADAS. The subsystems would mostly need to be newly developed.

- Existing ADAS would require significant modifications to be used as one of the channels. The cross-checking functionality and involved interfaces could require significant internal changes.
- Only parts of an ADAS SW stack could be carried over. The cross-checking functionality and the Selectors would have to be newly developed.

4.2.3.5 SIMPLICITY

Number, complexity, and performance of subsystems

The CPP architecture has very few subsystems but requires two channels capable of providing the nominal functionality.

- There are two complex and two simple subsystems in this architecture.
- The two channels both need to be capable of providing the nominal functionality, which likely involves ML/AI.
- While the two channels will have high performance requirements, the two Selectors are simple and have very low performance requirements.

Required level of diversity

While diversity is only required between two subsystems in this architecture, these both provide the same, nominal functionality.

- Diversity between the two channels is required to prevent common cause faults or functional insufficiencies. This is at the very least true for the perception layer[50], but could be easier for the planning and checking layers. The two Selectors are sufficiently simple that they can be developed to the highest integrity levels and use homogeneous redundancy.
- Both channels need to provide the same, nominal functionality. This makes it harder to find diverse approaches and/or different suppliers.

[50] Diversity in the perception layer could be facilitated by using different sensor sets for each channel. This may help prevent some common cause functional insufficiencies. Diversity in the checking layer might not be obligatory if it is implemented with the highest integrity level.

Complexity of validation

This architecture benefits from the fact that there are only two complex subsystems. However, these are more closely coupled, which increases the complexity of the validation.

- The two channels can be validated independently, but the integrated system needs substantial validation.
- The closer coupling implies that the overall validation effort could be higher due to emergent behavior.
- The absence of common cause failures and functional insufficiencies only needs to be shown for two complex subsystems (most other architectures require three).

4.2.3.6 SAFETY OF THE INTENDED FUNCTIONALITY

Support to accommodate functional insufficiencies

The cross-checking channels can compensate for each other's functional insufficiencies.

- An output insufficiency in one channel can be compensated for by the other channel if these are sufficiently independent. If Channel 1 has a false positive (i.e., it sees a non-existing object and plans a trajectory with unnecessary braking or evasion), Channel 2 should still consider Channel 1's output safe. If Channel 1 has a false negative (i.e., it doesn't see an existing object and plans an unsafe trajectory), it should still consider Channel 2's output safe.
- The diversity of the channels should provide some robustness to functional insufficiencies.
- No specific strategy for defining independent sensor sets composed of different sensor modalities is defined for this architecture. If sensors are shared between the channels, which can be expected due to cost reasons, their independence can be compromised. Additional analysis and rationales will be needed to argue that independence is maintained.

Support to manage operational conditions

This architecture does not define specific ways to handle the ODD in a centralized way.

- Each channel has to monitor the ODD separately. No central mechanism for deciding the reaction to an ODD exit or similar is foreseen.
- When an output insufficiency is detected, the other channel takes over control of the vehicle. As all channels are capable of providing the nominal functionality, this is applicable for the entire foreseen ODD. The channels are also aware of each other's condition and can use degradation, e.g., more conservative maneuvers and MRMs.
- No specific way for monitoring safety performance is defined for this architecture. Monitoring the checking results and Selector behavior could be useful.

4.2.4 EVALUATION OF THE DARUMA ARCHITECTURE

4.2.4.1 AVAILABILITY

Availability of the system

The Daruma architecture consists of several simultaneously active, heterogeneous channels. All outputs are cross-checked against the channels' environment models to create a ranking according to safety, comfort, efficiency, etc.

- This architecture maintains safety (both integrity and availability) after the failure of any single subsystem. If one of the channels suffers from a fault or functional insufficiency (see Figure 44 and Table 18), the Daruma subsystem will likely assign it a low safety score (as two environment models will consider it "unsafe") and a low ranking and the Selector will pick one of the higher-ranked, safer outputs. If one of the simple, and thus fail-silent, Selector blocks is faulty, the respective other one takes over. If the fail-silent Daruma block is faulty, the Selectors resort to a buffered MRM (see Figure 45), which may be too strongly degraded for a single-point fault. It may also be challenging to implement the medium-complexity Daruma subsystem as a fail-silent subsystem.
- The architecture can also tolerate some dual-points faults. A simultaneous failure of a (fail-arbitrary) channel and one of the (fail-silent) Daruma or Selector blocks remains safe. As all three channels generate Actuator Data, a simultaneous failure of two of them can also be safe, but only if these failures are detectable (e.g., fail-silent) by the cross-check. In the extreme case, only one of the Selector blocks might remain, outputting the buffered MRM as a last resort. The diversity of the channels can help address common cause faults. The three channels can be designed to all aim to fulfill the nominal function or some might focus on generating MRM trajectories – accordingly, their requirements towards sensors, HW processing elements, and application SW can be similar or differ significantly. The diverse case is more robust to common cause faults.

Table 18: Possible fault / output insufficiency scenarios for the Daruma architecture, considering silent (s) and arbitrary (a) failure modes. Diagonal elements: single-point faults ("OK" marks an MRM or better). Off-diagonal elements: dual-point faults ("OK" marks blind braking or better).

	C1 (s)	C1 (a)	C2 (s)	C2 (a)	C3 (s)	C3 (a)	D (s)	S1 (s)	S2 (s)
C1 (s)	OK	n/a	OK	OK (if re-sorting to MRM)	OK	OK (if re-sorting to MRM)	OK	OK	OK
C1 (a)		OK	OK (if re-sorting to MRM)	Incorrect (unsafe)	OK (if re-sorting to MRM)	Incorrect (unsafe)	OK	OK	OK
C2 (s)			OK	n/a	OK	OK (if re-sorting to MRM)	OK	OK	OK
C2 (a)				OK	OK (if re-sorting to MRM)	Incorrect (unsafe)	OK	OK	OK
C3 (s)					OK	n/a	OK	OK	OK
C3 (a)						OK	OK	OK	OK
D (s)							Blind braking	OK	OK
S1 (s)								OK	OK (if MRM buf-fered at actuators)
S2 (s)									OK

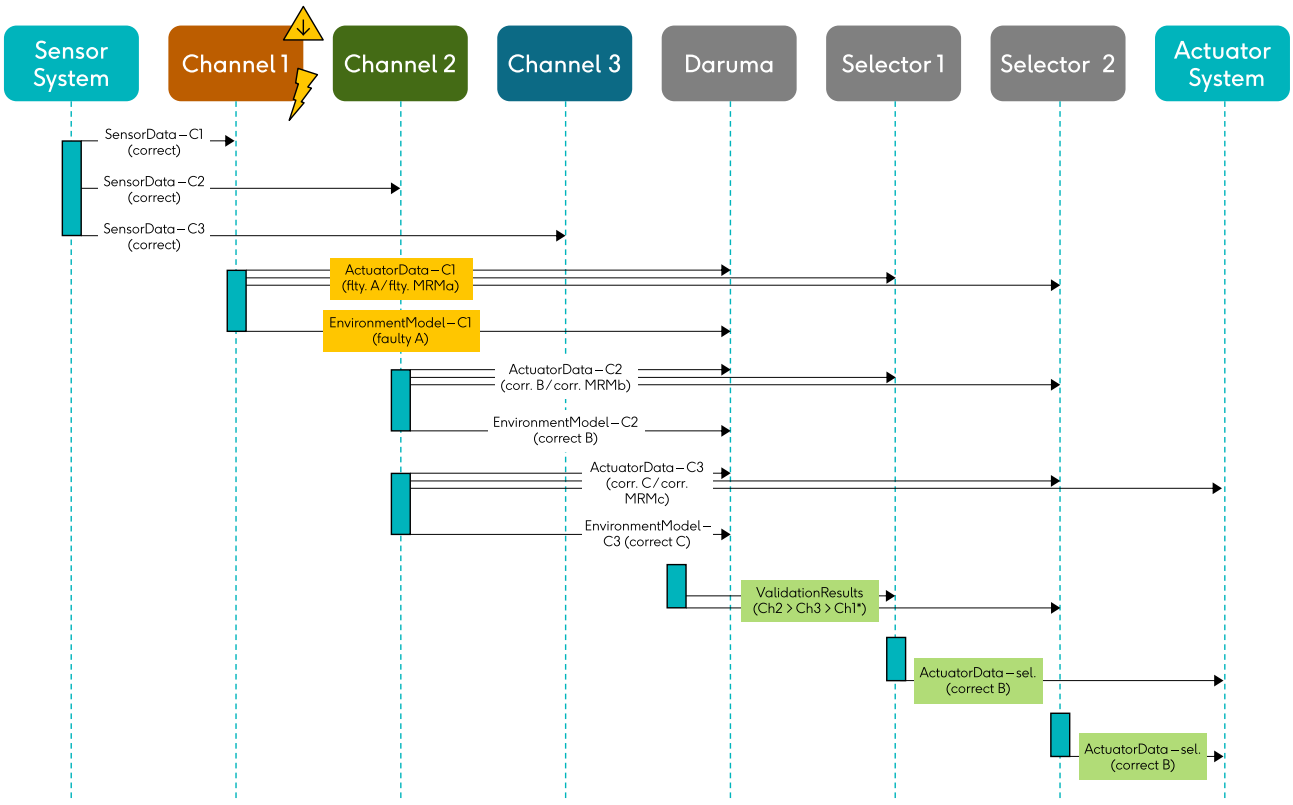


Figure 44: Sequence diagram of the Daruma architecture. The case with a fault or output insufficiency in Channel 1 is shown. Faulty messages are highlighted in yellow, fault containment in green.

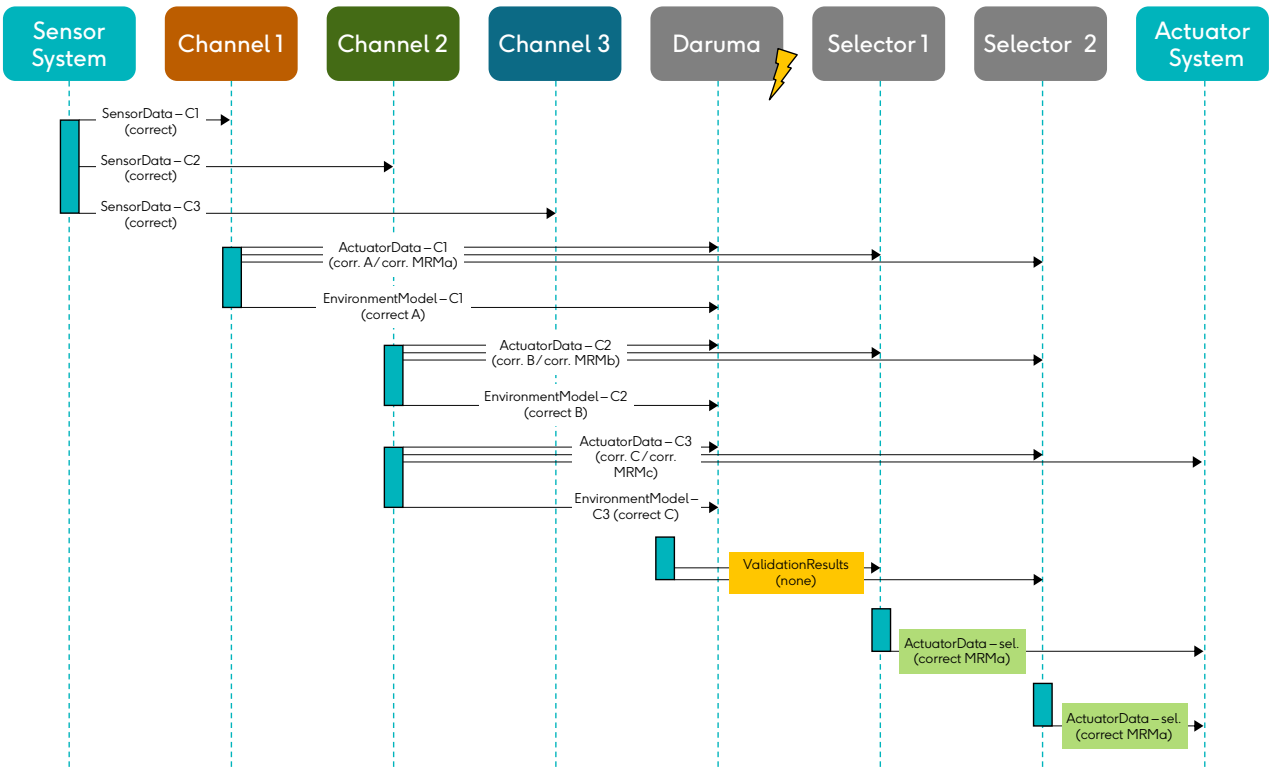


Figure 45: Sequence diagram of the Daruma architecture. The case with a fault in the Daruma subsystem is shown. Faulty messages are highlighted in yellow, fault containment in green.

Diagnostics scheme

The Daruma architecture is based on cross-checking between all channels and dynamically selecting the best safe output for the current driving situation.

- Through cross-channel analysis, the faulty channel can be identified. This can allow additional vehicle-level reactions, e.g., informing the driver and asking the remaining channels to perform an MRM.
- The channels are, by design, not aware of each other's condition. They only provide the trajectory/actuator data and the environment model on standardized interfaces.

Degradation scheme

The Daruma architecture can, by design, be extended to include any number of channels higher than one. While we discuss the one with three channels, which is the smallest one able to satisfy high availability, more could be added to increase degradation options.

- How noticeable the loss of a channel is to the end user depends on their relative level of degradation. If another channel can provide the nominal function, it will be unnoticeable. Losing the only channel that can do so would, however, be immediately noticeable.
- The levels of degradation can be very fine and tunable in this architecture as it is based on dynamically switching between channels to find the one best suited for a particular driving situation.

4.2.4.2 NOMINAL FUNCTIONALITY

Availability of nominal functionality

The Daruma architecture dynamically selects the best safe output for the current driving situation and is therefore able to well keep up nominal functionality even under faults, provided that a symmetric system layout is implemented (i.e., all channels can handle nominal functionality).

- This architecture favors switching to a channel with a safer trajectory as long as the channels are healthy. This may be unnoticeable as long as at least two channels work properly. This makes the architecture less sensitive to false positives (i.e., a channel "sees" an inexistent object) in the cross-validation of the different channels' trajectories and environment models (see also Table 19).
- Selecting the most suitable channel for each driving situation can help reduce nuisances. Among the safe trajectories, Daruma prefers the one that continues driving instead of stopping.
- The evaluation algorithms in the Daruma blocks can be split into two parts: the first is to calculate different scores related to the safety, comfort, efficiency, etc. of a trajectory with respect to each of the available environment models. The second step is to create a ranking according to these scores. This ranking process will likely prominently consider the safety scores. The algorithm for this may also take other diagnostics data or even the traffic situation into account.
- The arbitration algorithms in the Selectors can be minimal, e.g., selecting the highest-ranked output if it received a majority (i.e., at least 2oo3) and the hashes match.

Table 19: Different combinations of failures and/or functional insufficiencies and the resulting Daruma ranking. If a channel experiences a false positive (FP) detection (i.e., a phantom obstacle), it may plan an unnecessarily conservative trajectory. If it experiences a false negative (FN), it may plan an unsafe trajectory leading to a collision.

Channel 1	Channel 2	Channel 3	Possible Daruma ranking	Comment
OK	OK	OK	Ch1 > Ch2 > Ch3	Nominal case.
FP	OK	OK	Ch1 > Ch2 > Ch3	Channel 1 may give the other two channels low safety scores.
FP	FP	OK	Ch1 > Ch2 > Ch3	A cautious reaction to the phantom obstacle may still be safe.
FP	FP	FP	Ch1 > Ch2 > Ch3	May still be safe.
FN	OK	OK	Ch2 > Ch3 > Ch1	The other two channels may give Channel 1 a low safety score. May be indistinguishable from the case with two false positives.
FN	FN	OK	Ch3 > Ch1 > Ch2	Channel 3 may give the other two channels low safety scores. May be indistinguishable from the case with one false positive.
FN	FN	FN	Ch1 > Ch2 > Ch3	Unsafe.

4.2.4.3 CYBERSECURITY

Interactions between subsystems

The Daruma architecture requires only small amounts of data to be exchanged between its subsystems.

- In this architecture, the channels only interact with the Daruma and Selector subsystems. This communication is unidirectional.
- While most interfaces are narrow, an exchange of the environment model in a standardized format is necessary. Depending on the implementation, this may still be a relatively simple and small data structure, e.g., object list and outline of drivable area.
- Defined and restricted interfaces can be used.

Interactions with external systems

The different subsystems of the Daruma architecture require communication with external systems, e.g., a backend infrastructure.

- All three channels require connectivity. As simpler subsystems, the Daruma and Selector blocks are less likely to require this.
- Channels performing the nominal function may need constant connectivity to access HD maps or similar.
- All channels will need regular updates, which implies OTA. The updates will likely be less frequent for more strongly degraded channels or even absent for simple subsystems like the Daruma and Selector blocks. The different subsystems may make use of different update mechanisms, e.g., at the workshop.

4.2.4.4 SCALABILITY

Scalability towards higher availability

The Daruma architecture is intended to be extensible starting with two channels.

- More channels can be added to achieve higher availability goals. These can focus on particular driving situations or levels of degradation.
- However, the cross-validation and subsequent ranking make adding many strongly degraded channels impractical as they could outvote those performing the nominal function and lead to more false positives.

Scalability towards different offering levels

The Daruma architecture can reuse existing ADAS with minimal modifications.

- An existing ADAS can be reused as one of the channels. Only the standardized output interfaces for trajectory/actuator data and environment model need to be added.
- The Daruma and Selector subsystems are specific to the architecture and cannot be carried over from SAE L2 systems. They need to be newly developed for SAE L4 use cases.

4.2.4.5 SIMPLICITY

Number, complexity, and performance of subsystems

The Daruma architecture consists of both simple and complex subsystems.

- The architecture is comprised of three complex subsystems (the channels), two simple subsystems (the Selectors), and a medium complexity subsystem (the Daruma). Optionally, more channels can be added.
- All channels are likely to involve AI-based approaches. Depending on their relative level of degradation, i.e., focus towards providing the nominal function or only MRMs, their performance requirements can differ. At least one channel has high performance requirements.
- The Daruma and Selectors do not require AI-based approaches. The Selectors have low performance requirements, but Daruma may require at least medium performance.

Required level of diversity

The Daruma architecture relies on sufficient independence between the three (or more) channels.

- To achieve this, the architecture will likely have to be based on heterogeneous redundancy, i.e., diversity between the three complex subsystems. Depending on whether the channels have a similar or different focus, this may be easier or harder to achieve. Due to its very different role and lack of a perception component, the Daruma subsystem is inherently diverse compared to the channels.
- The Selectors are simple subsystems and can probably rely on homogeneous redundancy if they are developed to the highest applicable standards to preclude systematic faults.

Complexity of validation

This architecture has several diverse subsystems requiring independent validation.

- The channels are not directly coupled to each other and can be validated separately.
- Due to the cross-check in the Daruma subsystem, an indirect coupling exists, which requires joint validation. Their interplay and joint failure modes, e.g., scenarios where more than one channel suffers from functional insufficiencies could be quite complex.
- Ensuring the absence of common cause failures between the three channels could be challenging as they all have similar roles.

4.2.4.6 SAFETY OF THE INTENDED FUNCTIONALITY

Support to accommodate functional insufficiencies

This architecture considers several SOTIF aspects in the responsibilities of its subsystems.

- The architecture allows that each channel may focus on specific driving situations.
- The different channels are explicitly intended to compensate for each other's functional insufficiencies. They should be implemented in a diverse way on the algorithmic level.
- A diverse set of sensors can be used. No specific strategy for defining independent sensor sets composed of different sensor modalities is defined.

Support to manage operational conditions

The Daruma architecture considers overarching SOTIF aspects, e.g., related to off-board analysis.

- Each channel needs to check the ODD on its own. By design, the channels do not directly interact with each other.
- Other aspects to ensure safe usage are not explicitly mentioned but could be considered in the implementation.
- The cross-check scores related to safety, comfort, efficiency, etc. can be collected for off-board analysis.

4.2.5 EVALUATION OF THE CHANNEL-WISE DCF ARCHITECTURE

4.2.5.1 AVAILABILITY

Availability of the system

The C-DCF architecture consists of a heterogeneous hot standby redundancy (L2-System and F-System). The outputs of the L2-System are checked by the independent M-System. The system is capable of controlling the vehicle under both nominal and failure conditions.

- This architecture maintains safety (both integrity and availability) after the failure of any single subsystem (see Table 20) with its redundant subsystems and communication channels. If the L2-System is faulty (see Figure 46), the M-System will detect this, and the D-Systems will switch to the output of the F-System. A faulty M-System only poses a latent fault, which can be detected by latent fault tests[51]. If the F-System is faulty (see Figure 47), the M-System detects this latent fault and triggers a degraded mode in the L2-System. If one of the simple, and thus fail-silent, D-Systems is faulty, the respective other one takes over.
- The architecture can also tolerate some dual-point faults. Simultaneous failures of one of the complex subsystems (L2, M, or F) and one of the (fail-silent) D-Systems remain safe. A simultaneous failure of two complex subsystems will generally lead to the ADI going fail-silent. This could be addressed by adding a last resort, e.g., with a buffered MRM trajectory.

[51] These regularly test that the fault detection mechanisms work as intended. As an example, at regular intervals a faulty trajectory can be sent to the M-System with the expectation that the M-System finds it "unsafe".

Table 20: Possible fault / output insufficiency scenarios for the C-DCF architecture, considering silent (s) and arbitrary (a) failure modes. Diagonal elements: single-point faults ("OK" marks an MRM or better). Off-diagonal elements: dual-point faults ("OK" marks blind braking or better).

	L2 (s)	L2 (a)	M (s)	M (a)	F (s)	F (a)	D1 (s)	D2 (s)
L2 (s)	OK	n/a	OK	OK	Una-vailable (unsafe)	Incorrect (unsafe)	OK	OK
L2 (a)		OK	OK	Incorrect (unsafe)	Una-vailable (unsafe)	Incorrect (unsafe)	OK	OK
M (s)			OK	n/a	OK (if de-faulting to L2)	Incorrect (unsafe)	OK	OK
M (a)				OK	Una-vailable (unsafe)	Incorrect (unsafe)	OK	OK
F (s)					OK	n/a	OK	OK
F (a)						OK	OK	OK
D1 (s)							OK	Una-vailable (unsafe)
D2 (s)								OK

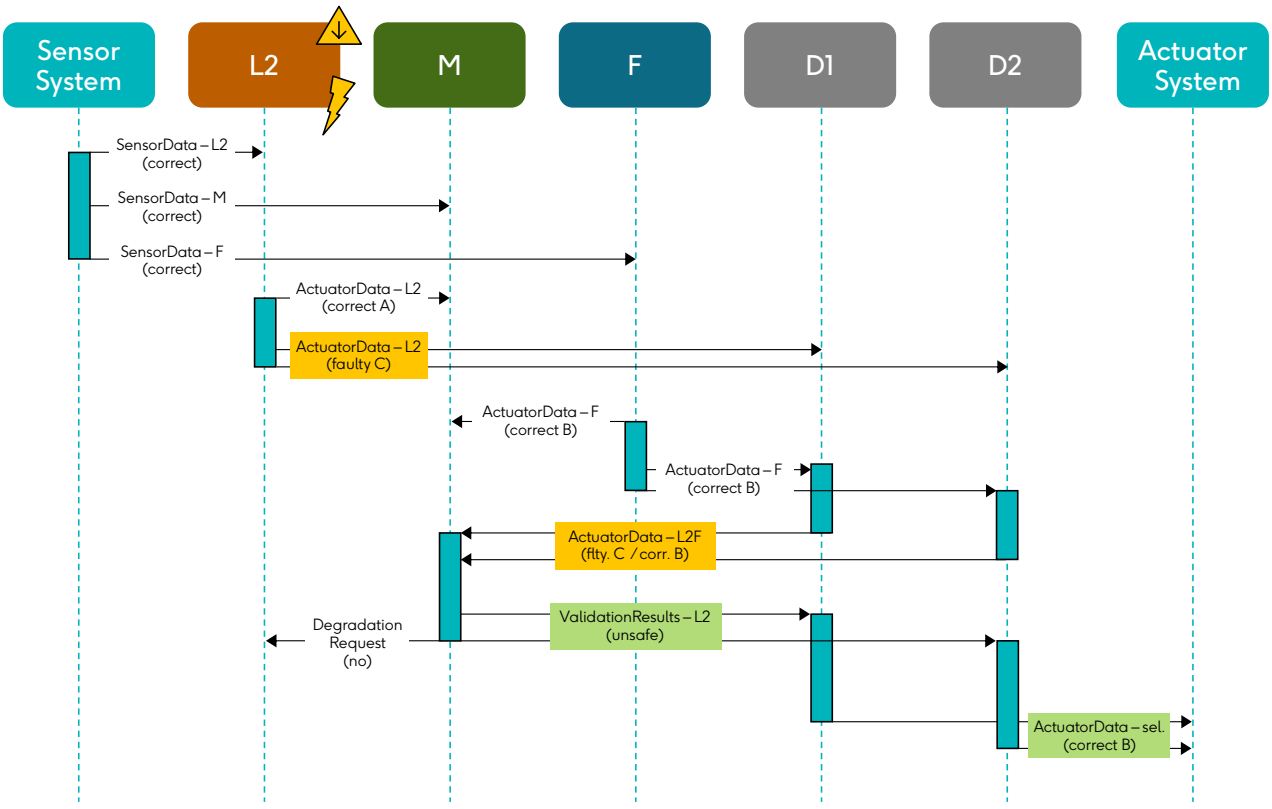


Figure 46: Sequence diagram of the C-DCF architecture. The case with a fault or output insufficiency in the L2-System is shown. Faulty messages are highlighted in yellow, fault containment in green.

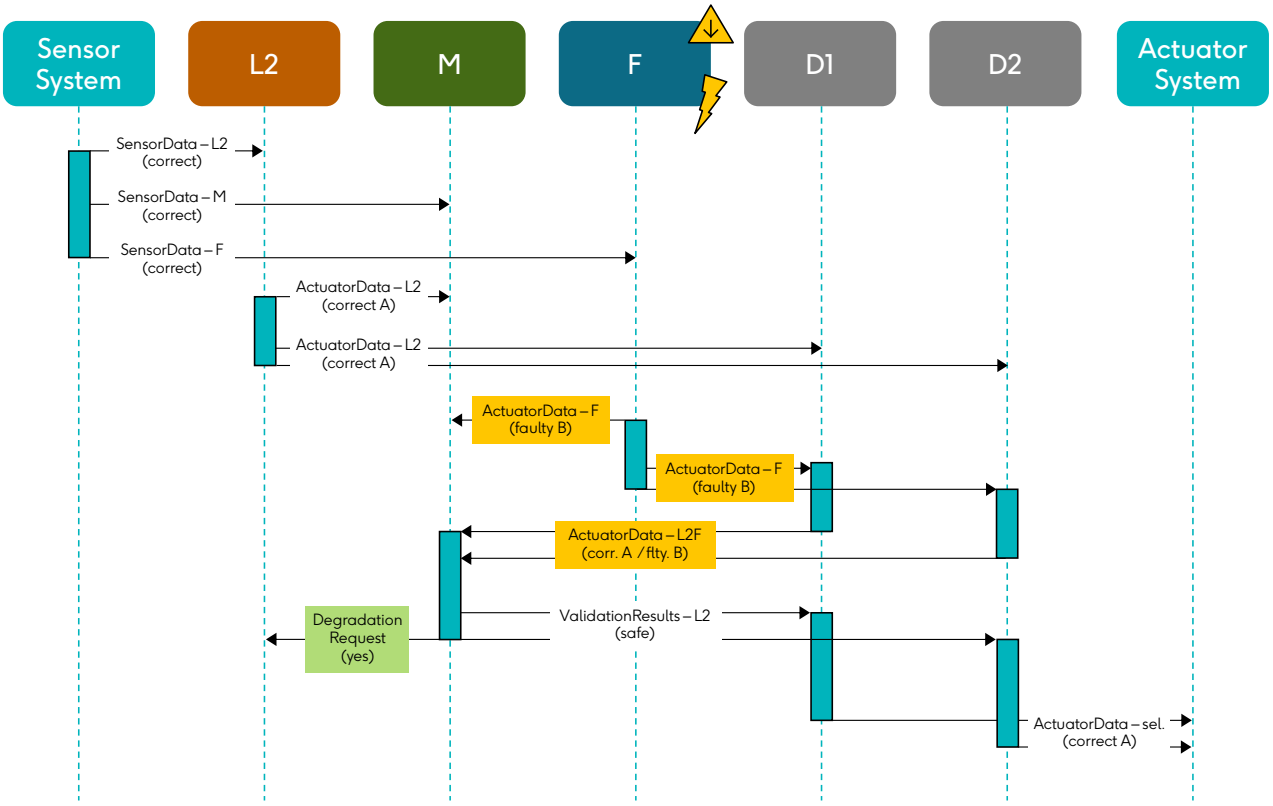


Figure 47: Sequence diagram of the C-DCF architecture. The case with a fault or output insufficiency in the F-System is shown. Faulty messages are highlighted in yellow, fault containment in green.

Diagnostics scheme

The C-DCF architecture is based on checking the outputs of the L2- and F-Systems.

- The M-System checks both other complex subsystems and informs the L2-System if the F-System is found to be faulty.
- This allows it to request the L2-System to go to a degraded mode (execute MRM and inform the driver) if there is a latent fault in the F-System.

Degradation scheme

This architecture provides for a controlled degradation of functionality under fault conditions.

- Recovery of the L2-System is explicitly allowed, but only after a reset. The source material assumes that this would make transient faults or functional insufficiencies less noticeable to the end user. A permanent switch to the F-System, which only generates MRM trajectories, would be noticeable, though. Similarly, a latent fault in the F-System leads to a noticeable reaction in the L2-System.
- Only the L2-System has defined levels of degradation, i.e., normal operation and MRM.

4.2.5.2 NOMINAL FUNCTIONALITY

Availability of nominal functionality

The C-DCF relies on an evaluation of the outputs of L2-System and safety evaluation of the resulting trajectory in M-System. The D-Systems need to be simple and fault tolerant.

- This architecture is sensitive to false positives in the M-System, where even a transient "detection" of an inexistent object can lead to switching to the F-System. This may be compensated for if the L2-System can be recovered quickly enough. However, it is unclear how fast such a recovery could be achieved and consequently how noticeable a false positive would be to the passengers.
- All complex subsystems are necessary for the reliability of the system. Although failures of the L2-, M-, or F-Systems lead to a planned safety reaction, they reduce the system reliability and can be perceived as a nuisance, e.g., unnecessary MRMs.
- The arbitration algorithms are simple and straightforward. The M subsystem evaluates trajectories against its internal environment model and makes a binary decision based on that, e.g., using a threshold based on risk scores.
- The logical decisions are outlined in the source material. The algorithms of the trajectory evaluation itself are specific to the implementation.

4.2.5.3 CYBERSECURITY

Interactions between subsystems

This architecture only requires low amounts of data to be exchanged between its subsystems.

- There is no direct interaction between the L2- and F-Systems. The other subsystems all interact with each other. Some communication is bidirectional.
- All interfaces are narrow, i.e., only trajectories, actuator setpoints, evaluation results, etc.
- Defined and restricted interfaces can be used.

Interactions with external systems

While all subsystems in this architecture will need updates, some subsystems are relatively loosely coupled. This could allow the use of different and potentially more secure update mechanisms, making it harder to compromise the ADI with a single attack.

- All complex subsystems require communication with the back end.
- As the provider of the nominal functionality, the L2-System will likely need constant connectivity to access HD maps or similar.
- All complex subsystems will require regular updates, likely via OTA. Only the D-Systems may – if they are so simple that they can be thoroughly verified – not need updates, or at most very rarely. A different update mechanism could be used for this.

4.2.5.4 SCALABILITY

Scalability towards higher availability

The C-DCF architecture does not foresee extensibility.

- Improving the availability of the system would entail improving the availability of each subsystem, which is not an architectural topic.
- Technically, it would be possible to add more L2-Systems, but this is not explicitly covered in the source material. It would require extensive modifications of the M-System and a more complex arbitration.

Scalability towards different offering levels

It may be possible to carry over components from an existing SAE L2 system or downgrade to such a system.

- The L2-System has similar functionality and requirements to an SAE L2 ADAS and could naturally be reused in a lower-tier vehicle as such, with M-, F- and D-Systems omitted.
- An SAE L2 ADAS system could be upgraded to the L2-System; some modifications, e.g., for degraded mode, would be necessary.
- The M-, F- and D-Systems are specific to the architecture and cannot be carried over from SAE L2 systems. These components would need to be newly developed for SAE L4 use cases.

4.2.5.5 SIMPLICITY

Number, complexity, and performance of subsystems

The C-DCF architecture consists of both simple and complex subsystems.

- The architecture is comprised of three complex subsystems (L2, M, and F) and two simple subsystems (D1 and D2).
- All complex subsystems are likely to involve AI-based approaches. The performance requirements for the M- and F-Systems, which focus on monitoring and emergency reactions only, are likely lower than for the L2-System.
- The D-Systems are much simpler and have low performance requirements. If they can be fully verified, homogeneous redundancy is recommended for them.

Required level of diversity

This architecture relies on sufficient independence between the three complex subsystems.

- The L2-, M-, and F-Systems should be based on heterogeneous redundancy, i.e., diversity. This is facilitated quite naturally by the fact that they have different roles and capability levels.
- The D-Systems can be homogeneously redundant. They are much simpler and could be fully verified.

Complexity of validation

This architecture has several diverse subsystems requiring independent validation.

- The L2- and F-Systems are not directly coupled and could be independently validated. The L2- and M-Systems are more tightly coupled and probably require joint validation. It may be similar for the F- and M-Systems. The D-Systems are so simple that they can probably be fully verified.
- The possibility of independent validation reduces the validation effort drastically (compared to black-box validation of a fully integrated system, e.g., the Single-Channel architecture).
- Ensuring sufficient independence between the L2-, M-, and F-Systems is facilitated by them having different roles. This can allow for using different functional approaches, e.g., for perception, environment modeling, prediction, and/or planning.

4.2.5.6 SAFETY OF THE INTENDED FUNCTIONALITY

Support to accommodate functional insufficiencies

This architecture uses different roles for the complex subsystems.

- The complex subsystems have complementary functions – i.e., nominal function (L2), monitoring (M), and MRM function (F).
- Assuming diverse algorithmic implementations, a high level of mutual coverage of triggering conditions and avoidance of functional insufficiencies can be supported.
- A diverse sensor set can be used. Disjoint (or only partially shared) sensor sets for the three complex subsystems are suggested.

Support to manage operational conditions

The C-DCF architecture supports a differentiated ODD supervision and handling on the architecture level.

- The L2-System contains mechanisms to detect an ODD exit.
- The M-System also acts as an ODD monitor and checks the output of the L2-System for violations of the ODD-related assumptions and the generated trajectory.
- The F-System by definition does not consider a particular ODD but assumes any drivable road condition without weather or geofence limitation for its MRM.
- No specific way for data collection is defined for this architecture. The M-System could naturally be enhanced to track operating conditions and safety margins of the L2-System's operation.

4.2.6 EVALUATION OF THE LAYER-WISE DCF ARCHITECTURE

4.2.6.1 AVAILABILITY

Availability of the system

The L-DCF architecture consists of a heterogeneous hot redundancy (Primary and Safing channels). Each of these consists of a Doer/Checker (Safety Gate) pair, which makes this architecture robust to dual-point faults or functional insufficiencies (see Table 21).

- This architecture maintains safety (both integrity and availability) after the failure of any single subsystem. If the Primary subsystem is faulty (see Figure 48), the Primary Safety Gate detects this and silences the entire Primary unit, allowing the Safing channel (equivalent to a Fallback) to take over. Similarly, a (latent) fault in the Safing unit can be detected and reacted to. It may also be challenging to implement the medium-complexity Safety Gates, which process sensor data, as fail-silent subsystems. If one of the simple, and thus fail-silent, Priority Selectors is faulty, the respective other one takes over. The source material shows a shared sensor set for the PSG, S, and SSG subsystems, which needs to be handled carefully to prevent common cause faults; this pattern is also repeated for other layers, e.g., using the same occupancy grid as inputs to the planners and Safety Gates.
- The architecture can also tolerate dual-point faults. If there are simultaneous faults in both Primary and Safing subsystems (see Figure 49), the Safety Gates resort to buffered MRM trajectories. If there are simultaneous faults in both Safety Gates, the Priority Selectors resort to a blind braking maneuver (see Figure 50).

Table 21: Possible fault/functional insufficiency scenarios for the L-DCF architecture, considering silent (s) and arbitrary (a) failure modes. Diagonal elements: single-point faults ("OK" marks an MRM or better). Off-diagonal elements: dual-point faults ("OK" marks blind braking or better). Entries in [brackets] are failure modes excluded in the source material.

	P (s)	P (a)	PSG (s)	PSG (a)	S (s)	S (a)	SSG (s)	SSG (a)	PS1 (s)	PS2 (s)
P (s)	OK	n/a	OK	[incorrect (unsafe)]	OK	OK	OK (if MRM buffered at actuators)	[incorrect (unsafe)]	OK	OK
P (a)		OK	OK	[incorrect (unsafe)]	OK	OK	OK (if MRM buffered at actuators)	[incorrect (unsafe)]	OK	OK
PSG (s)			OK	[n/a]	OK (if MRM buffered at actuators)	OK (if MRM buffered at actuators)	OK (if MRM buffered at actuators)	[incorrect (unsafe)]	OK	OK
PSG (a)				[incorrect (unsafe)]	[incorrect (unsafe)]	[incorrect (unsafe)]	[incorrect (unsafe)]	[incorrect (unsafe)]	[incorrect (unsafe)]	[incorrect (unsafe)]
S (s)					OK	n/a	OK	[OK]	OK	OK
S (a)						OK	OK	[OK]	OK	OK
SSG (s)							OK	[n/a]	OK	OK
SSG (a)								[OK]	[OK]	[OK]
PS1 (s)									OK	OK (if MRM buffered at actuators)
PS2 (s)										OK

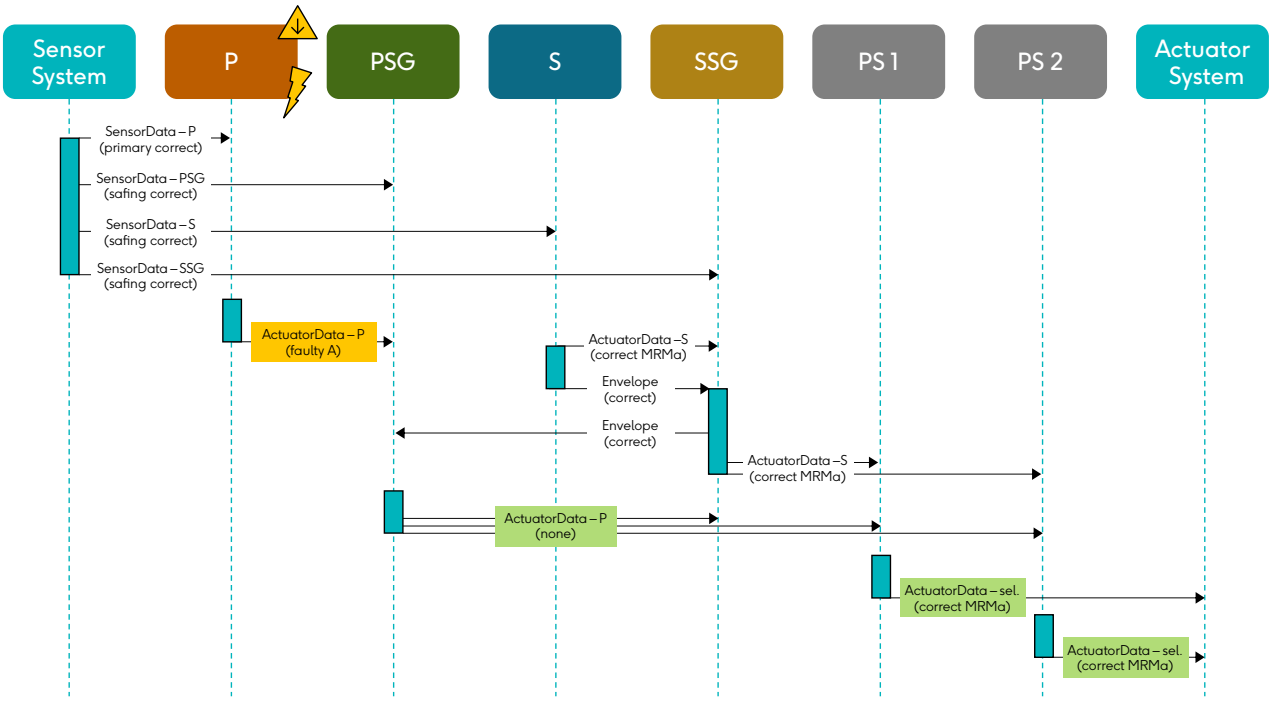


Figure 48: Sequence diagram of the L-DCF architecture. The case with a fault or a functional insufficiency in the Primary unit is shown. Faulty messages are highlighted in yellow, fault containment in green.

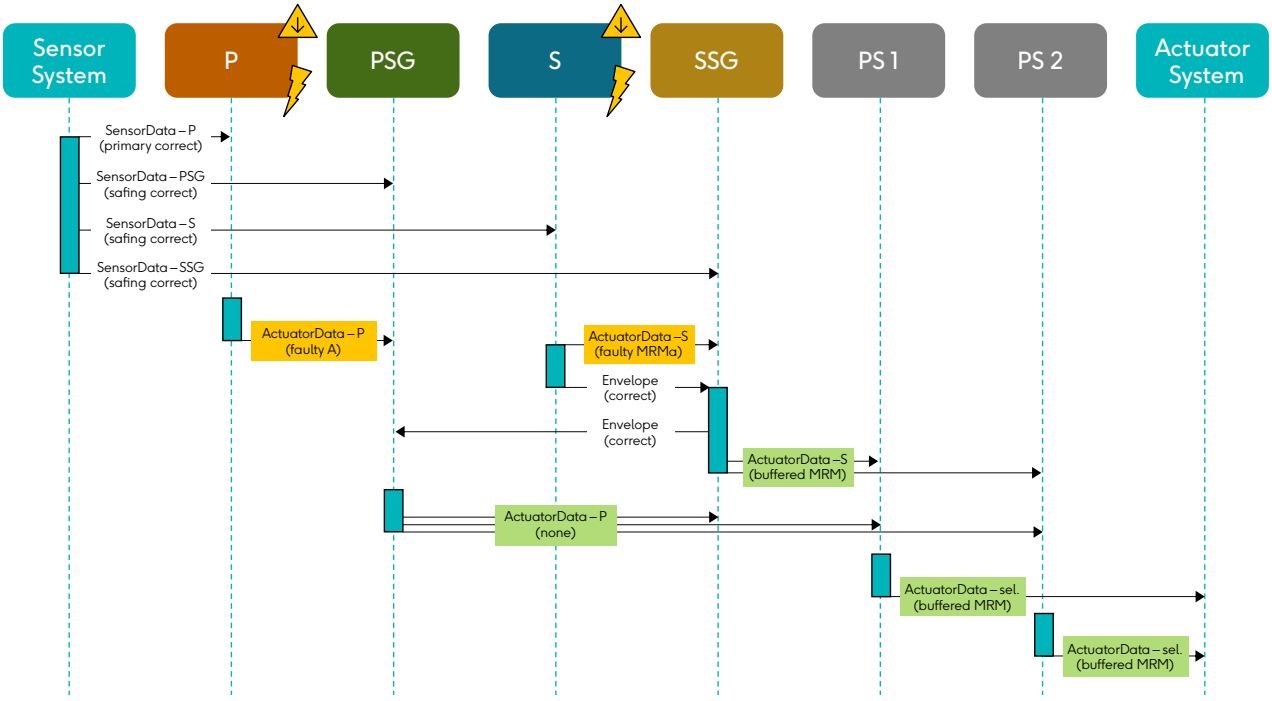


Figure 49: Sequence diagram of the L-DCF architecture. The case with faults or functional insufficiencies in the Primary and Safing units is shown. Faulty messages are highlighted in yellow, fault containment in green.

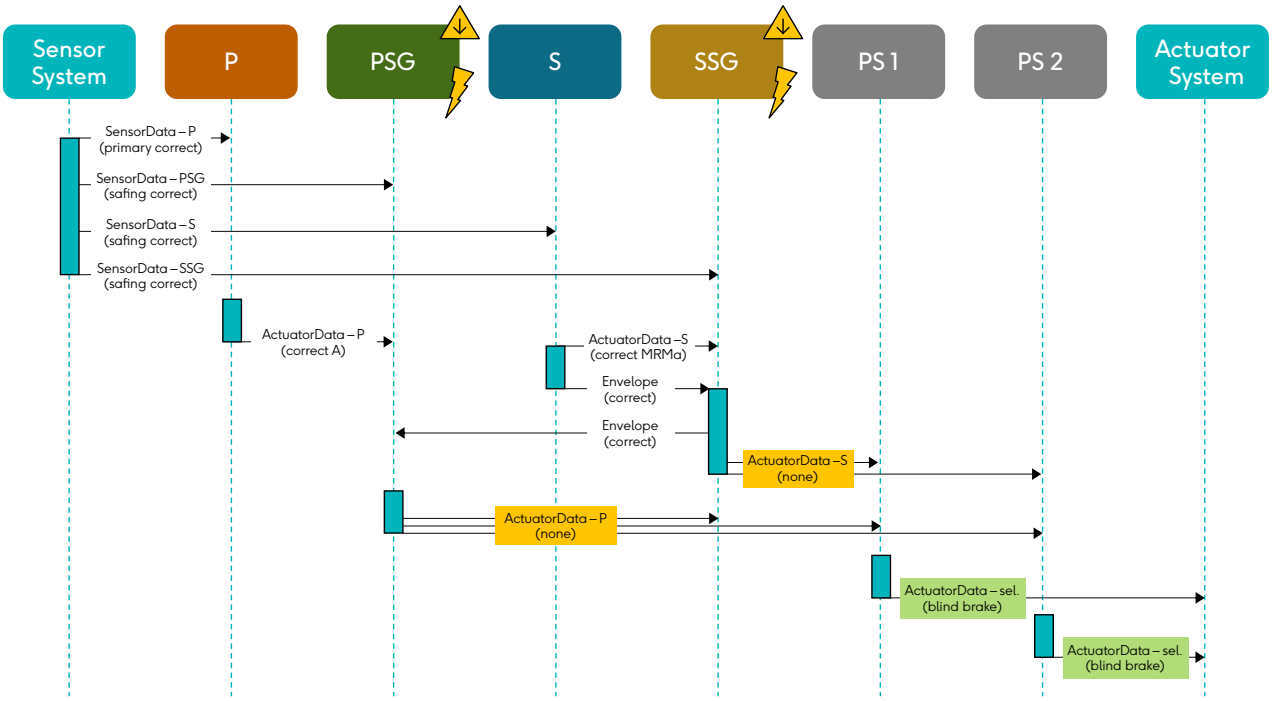


Figure 50: Sequence diagram of the L-DCF architecture. The case with faults or functional insufficiencies in the Primary Safety Gate and Safing Safety Gate is shown. Faulty messages are highlighted in yellow, fault containment in green.

Diagnostics scheme

The L-DCF is based on largely independent Doer / Checker pairs.

- The different subsystems are not aware of each other's condition.
- The different subsystems do not adapt their behavior based on faults elsewhere in the system.

Degradation scheme

This architecture provides several levels of degradation of functionality under different fault conditions.

- A permanent switch from the Primary channel to the (degraded) Safing channel would be noticeable. The source material does not state explicitly whether back-and-forth switching to cover transient faults is allowed. Some faults or functional insufficiencies could be mitigated via the envelope feedback pattern where the Safing Safety Gate provides its checking envelope to the Primary subsystem. This can help reduce the necessity for back-and-forth switching.
- While the different subsystems do not adjust their behavior internally, their different roles allow for four levels of degradation: the nominal function (provided by the Primary subsystem), a graceful MRM (provided by the Safing subsystem), a pre-planned MRM (buffered by the Safety Gates), and a minimal blind braking maneuver (provided by the Priority Selectors).

4.2.6.2 NOMINAL FUNCTIONALITY

Availability of nominal functionality

The L-DCF architecture relies on two independent channels, each comprised of a trajectory-producing subsystem and a Safety Gate.

- The envelope feedback can help reduce the sensitivity of the architecture to false positives in the Safety Gates (i.e., where they "see" inexistent objects).
- The reliability of the system hinges on the Safety Gates, which are intended to have higher integrity levels than the Primary or Safing units. The multi-layered degradation strategy with MRMs and pre-planned MRMs should help reduce the probability for nuisances, where the vehicle might block traffic due to an in-lane stop.
- The source material does not specify details of how the checks in the Safety Gates are constructed. It may be challenging to reach the demanded high integrity levels (and fail-silent property) in subsystems that include complex sensor data processing. The arbitration in the Priority Selectors on the other hand should be simple and straightforward.
- The "permissive envelope", which is roughly equivalent to the envelope feedback architectural pattern (see section 3.2.1.4), is not described in detail. While it may decrease false positives, it must be carefully implemented to avoid internal shortcuts in trajectory generation that would jeopardize independence between the Primary and Secondary channels.

4.2.6.3 CYBERSECURITY

Interactions between subsystems

The subsystems are strongly separated, but the sharing of input sensor data from the Safing channel requires a broad communication interface to transfer a large amount of data.

- The Primary and Safing subsystems do not communicate directly. However, there are several interactions between the Safety Gates and safing sensor data is also shared with the Primary Safety Gate. Adding more layers to the architecture increases the number of units and interfaces.
- Most interfaces are narrow, except for the sensor data sent to the Primary Safety Gate.
- Defined and restricted interfaces can be used.

Interactions with external systems

While all subsystems in this architecture will need updates, some subsystems are relatively loosely coupled. This could allow the use of different and potentially more secure update mechanisms, making it harder to compromise the ADI with a single attack.

At least the Primary and Safing subsystems require communication with off-board systems. As the provider of the nominal functionality, the Primary subsystem will likely need constant connectivity to access HD maps or similar.
All complex subsystems will require regular updates, likely via OTA. The Priority Selectors may – if they are so simple that they can be thoroughly verified – not need updates, or at most very rarely. A different update mechanism could be used for this.

4.2.6.4 SCALABILITY

Scalability towards higher availability

The L-DCF architecture does not foresee extensibility. Each channel may be split into several layers, which may help with increasing availability.

- As each channel can be split into several layers, with independent Primary and Safing units and associated Safety Gates, it may be possible to feed later layers with the validated output of earlier layers, e.g., Primary Safety Gate of perception layer feeds into both Primary and Safing channels of planning layer. This may increase availability to some extent.
- Each channel could be split into multiple layers with distinct subsystems, e.g., Primary Perception unit, Primary Perception Safety Gate, Primary Planning unit, etc. This will need careful mapping to maintain fault containment units.

Scalability towards different offering levels

It may be possible to carry over components from an existing SAE L2 system.

- The Primary and Safing subsystems have similar functionality and requirements to an SAE L2 ADAS. These components could potentially be reused in a lower-tier vehicle with only an L2 system, resulting in some cost savings.
- The Safety Gates and the Priority Selector are specific to SAE L4 and higher systems and would need to be newly developed for such use cases.

4.2.6.5 SIMPLICITY

Number, complexity, and performance of subsystems

The L-DCF architecture consists of both simple and complex subsystems. Using more than one layer increases the number of subsystems.

- The architecture is comprised of four complex subsystems (P, S, PSG, and SSG) and two simple subsystems (PS1 and PS2). This number scales with the number of layers the ADI is structured into. Two disjoint sensor sets are proposed.
- All complex subsystems are likely to involve AI-based approaches.
- It may be difficult to balance two conflicting design goals for the Safety Gates: ensuring their (very high) integrity while keeping them sufficiently independent from their respective function counterparts.
- The performance requirements for the Safing subsystem and the Safety Gates, which focus on emergence reactions and monitoring only, are likely lower than for the Primary subsystem.
- The Priority Selectors are much simpler and have low performance requirements. If they can be fully verified, homogeneous redundancy could be employed.

Required level of diversity

This architecture relies on sufficient independence between the Primary and Safing channel and also between the core function and the Safety Gate in each channel.

- The Primary and Safing channels should be based on heterogeneous redundancy, i.e., diversity. In addition, the functional subsystem (P or S) and respective Safety Gate will also need to be independent. This may be facilitated by the fact that they have different roles and capability levels.
- The Priority Selectors can be homogeneously redundant. They are much simpler and could be fully verified.

Complexity of validation

This architecture has several diverse subsystems, only some of which allow for independent verification.

- In each channel, the function subsystem (P or S) is coupled with the respective Safety Gate and will likely need to be jointly validated. There is also some coupling between the Primary and Safing channels.
- The required validation effort may increase further if more than one layer is used.
- Ensuring sufficient independence between the P, S, PSG, and SSG subsystems is facilitated by them having to some extent different roles. This can allow for using different functional approaches, which may be specific to the respective layer, supporting arguments that common cause failures are addressed.

4.2.6.6 SAFETY OF THE INTENDED FUNCTIONALITY

Support to accommodate functional insufficiencies

This architecture uses different roles for some of the complex subsystems.

- The Primary and Safing subsystems have different roles. The two Safety Gates also have a different role. This helps address the different ODDs and driving policies.
- A high level of coverage of triggering conditions and functional insufficiencies can be supported if diverse algorithmic implementation is applied.
- A diverse sensor set can be used. Independent primary and safing sensor sets are suggested.

Support to manage operational conditions

This architecture does not define specific ways to handle the ODD in a centralized way.

- Each channel has to monitor the ODD separately. No central mechanism for deciding the reaction to an ODD exit or similar is foreseen.
- When a functional insufficiency occurs in the Primary channel, the Safing channel takes over control of the vehicle. If it is also not available, a more strongly degraded reaction can be applied. This is less reliant on an ODD.
- No specific way for monitoring safety performance is defined for this architecture.

4.2.7 EVALUATION OF THE DSM ARCHITECTURE

4.2.7.1 AVAILABILITY

Availability of the system

The DSM architecture is an extension of the E-GAS architecture with availability in mind.

- This architecture maintains safety (both integrity and availability) after the failure of any single subsystem. If the FUN subsystem is faulty (see Figure 51), the SFM subsystem detects this and silences it, allowing the VSM to take over with a graceful MRM. Each subsystem is monitored by another subsystem, which can silence it. The fail-silent failure mode assumption of the VSM may be harder to achieve as this subsystem needs to process sensor data. Unlike most other architectures, the reaction may be delayed by one cycle, i.e., until the faulty subsystem has been turned off. The source material also proposes redundant sensor sets and communication networks.
- The architecture can also tolerate some dual-point faults. Simultaneous failures of the FUN and VSM subsystems (see Figure 52) are detected by the CSM subsystem, which takes over with a degraded MRM.

Table 22: Possible fault / functional insufficiency scenarios for the DSM architecture, considering silent (s) and arbitrary (a) failure modes. Diagonal elements: single-point faults ("OK" marks an MRM or better). Off-diagonal elements: dual-point faults ("OK" marks blind braking or better). Entries in [brackets] are failure modes excluded in the source material.

	FUN (s)	FUN (a)	SFM (s)	SFM (a)	CSM (s)	CSM (a)	VSM (s)	VSM [a]
FUN (s)	OK	n/a	OK	OK	OK	OK (if monitored by VSM)	OK	[incorrect (unsafe)]
FUN (a)		OK	OK	Incorrect (unsafe)	Incorrect (unsafe)	Incorrect (unsafe)	OK	[incorrect (unsafe)]
SFM (s)			OK	n/a	OK	OK (if monitored by VSM)	OK	[incorrect (unsafe)]
SFM (a)				OK	OK	OK (if monitored by VSM)	OK	[incorrect (unsafe)]
CSM (s)					OK	n/a	OK	[incorrect (unsafe)]
CSM (a)						OK (if monitored by VSM)	Incorrect (unsafe)	[incorrect (unsafe)]
VSM (s)							OK	[n/a]
VSM [a]								[OK]

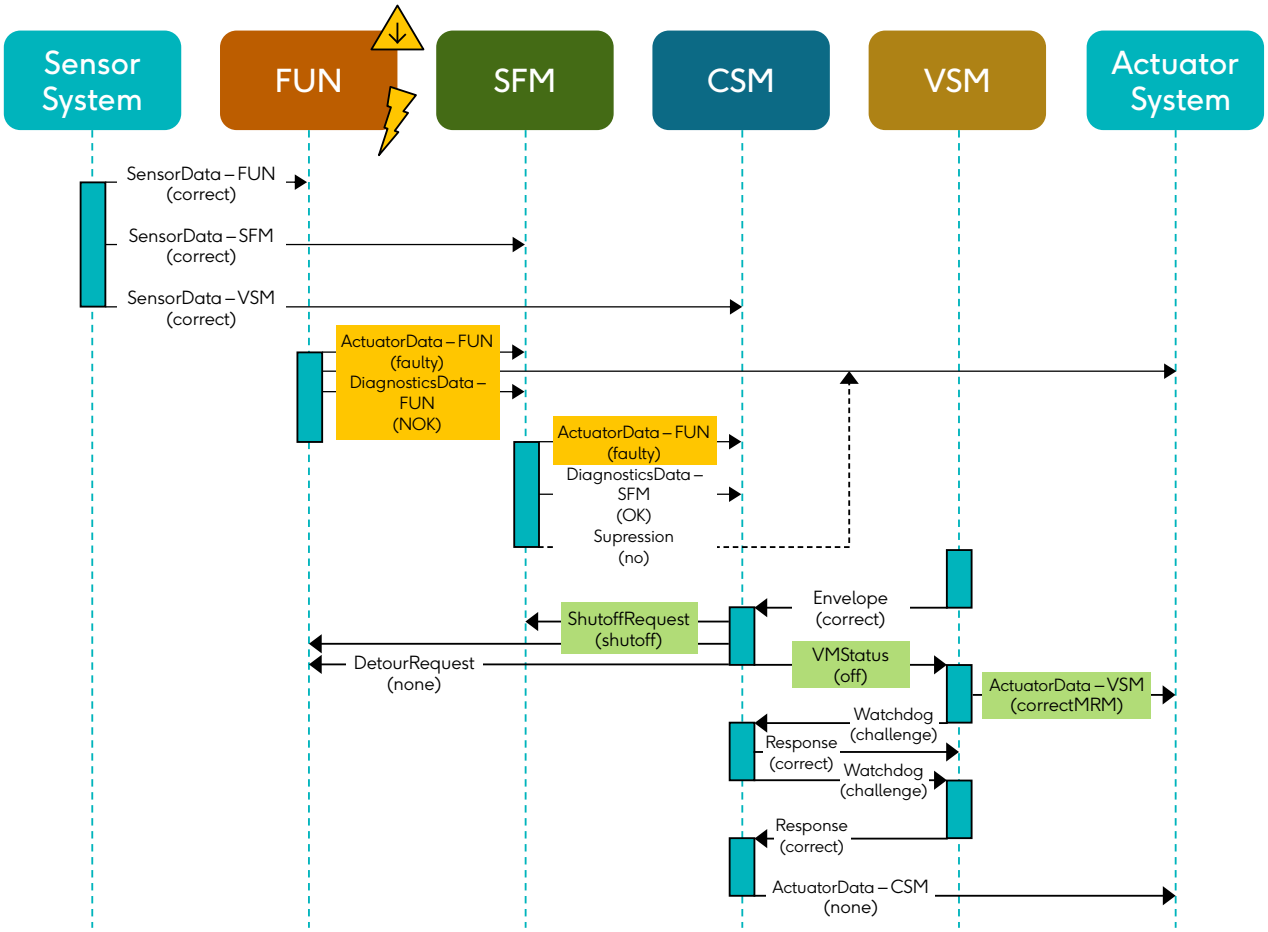


Figure 51: Sequence diagram of the DSM architecture. The case with a fault or functional insufficiency in the FUN subsystem is shown. Faulty messages are highlighted in yellow, fault containment in green.

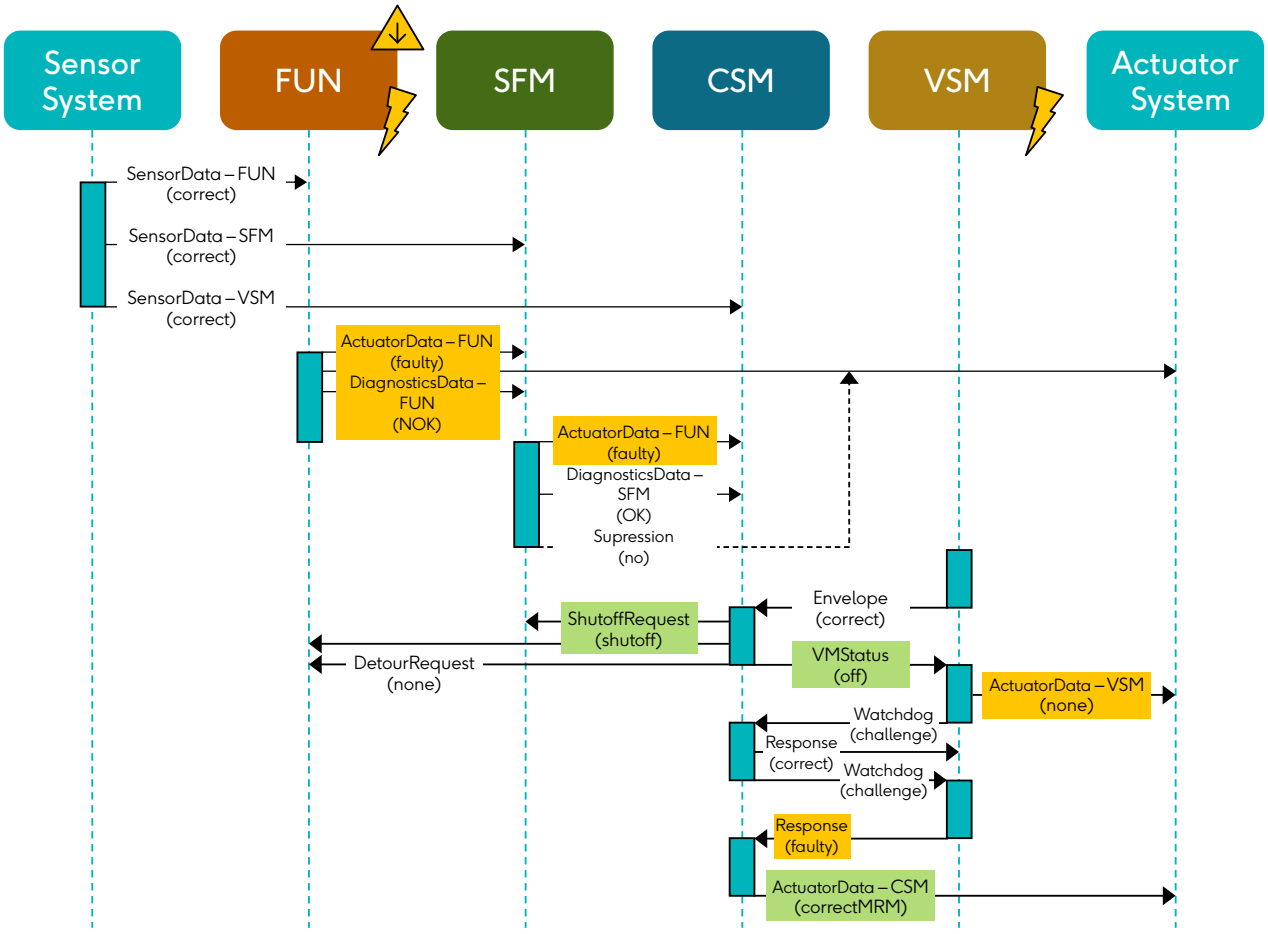


Figure 52: Sequence diagram of the DSM architecture. The case with faults or functional insufficiencies in the FUN and VSM subsystem is shown. Faulty messages are highlighted in yellow, fault containment in green.

Diagnostics scheme

The architecture includes multiple layers of diagnostic checking.

- Continuous diagnostic checking is performed on the functional sensors, safety sensors, as well as between various subsystems within the architecture. If a subsystem is found to be faulty, it is shut down by the subsystem monitoring it.
- A continuous (cross-checking) challenge and response mechanism between the VSM and CSM confirms that both subsystems are within their safe operational parameters.

Degradation scheme

The DSM architecture provides for a controlled degradation of functionality under failure conditions.

- Transient faults can occur frequently. As this architecture does not foresee back-and-forth switching – faulty subsystems are shut down – these would be immediately noticeable to the end user.
- Degradation occurs in a controlled manner, depending on the nature of faults detected. There are five different modes of operation, each providing diminishing levels of comfort to the passengers: Full AD mode, a “detour” mode when repair is needed, a “comfort stop” occurring at the next opportunity, an immediate “safety stop” when the fault(s) require urgent reaction, and an “emergency stop” when it is no longer safe to continue operating.

4.2.7.2 NOMINAL FUNCTIONALITY

Availability of nominal functionality

The DSM architecture provides two subsystems capable of fully autonomous operation. Only the FUN subsystem can provide the nominal functionality.

- Recovery options are not explicitly discussed in the source material. Without recovery, a false positive in the SFM or CSM subsystem could lead to a loss of nominal functionality as the FUN subsystem would be shut down.
- The active checking approach performed by the CSM is unclearly defined and may be impractical if the asked-for agreement is too strict. Differences between FUN and VSM actuator commands must be expected due to differences in sensor inputs and computational algorithms. These differences may not indicate failure but rather different correct decisions. The CSM may thus need to resort to the MRM, which reduces the availability of the nominal functionality.
- The DSM architecture requires a complex mix of hardware and software. The resulting architectural footprint may result in a higher potential defect rate within subsystems or in the interactions between subsystems.

4.2.7.3 CYBERSECURITY

Interactions between subsystems

Several subsystems in this architecture are capable of shutting down other subsystems, which can have a significant impact.

- There are a lot of interactions between subsystems, all of which are complex. In addition, several subsystems are capable of shutting down other subsystems. A single compromised subsystem can therefore have a large impact.
- All interfaces are narrow, i.e., only trajectories, actuator setpoints, diagnostics data, etc.
- Defined and restricted interfaces can be used.

Interactions with external systems

It is expected that this architecture will require communication outside of the system. All subsystems are likely to require regular updates. As a result, the system would likely be connected to a network to perform these updates.

- All subsystems require communication with the outside world.
- The FUN subsystem provides the nominal functionality and will therefore likely need constant connectivity to access HD maps or similar.
- Several subsystems (mostly the FUN and only to a lesser extent the associated SFM) are highly complex and likely to require frequent updates, which are thus likely to use OTA. The other subsystems (CSM and VSM) are slightly simpler and likely to require less frequent updates, which may thus use more secure update mechanisms.

4.2.7.4 SCALABILITY

Scalability towards higher availability

This architecture appears to be extensible to achieve higher availability, e.g., for fully driverless AD use cases.

- This architecture explicitly has scalability in mind to achieve higher availability if needed.
- There is the possibility to add more VMs (adding more FUN and SFM modules) for higher availability or for load balancing if performance is an issue. However, there is only one VSM, which could prove a bottleneck for increasing availability if it is prone to failure.

Scalability towards different offering levels

It may be possible to carry over components from an existing SAE L2.

- The FUN has similar functionality and requirements as an SAE L2 ADAS. Parts of this component could potentially be reused in a lower-tier vehicle with only an L2 system, resulting in some cost savings.
- Other components of the system are specific to SAE L4 or higher (SFM, CSM, and VSM). These components would likely be developed and manufactured only for the fully functional AD system.

4.2.7.5 SIMPLICITY

Number, complexity, and performance of subsystems

The DSM architecture consists of complex subsystems which interact in a complicated manner.

- All subsystems (FUN, SFM, CSM, and VSM) are relatively complex.
- All subsystems performing perception by processing raw sensor data are likely to involve AI-based approaches.
- While FUN and SFM are the most complex, the CSM subsystem may have only moderate complexity and low performance requirements, but this depends on the complexity of the implemented checking approach. The VSM subsystem has at least moderate complexity, but it may be challenging to build a fail-silent, high integrity subsystem that implements perception and trajectory generation. The source material discusses using two disjoint sensor sets and redundant high-bandwidth communication networks, which are implementation considerations. Both of these imply higher manufacturing costs and complexity.

Required level of diversity

This architecture relies on sufficient independence between its four subsystems.

- The FUN and the SFM are complementary, making it easier to ensure sufficient independence. The FUN, SFM, CSM, and the VSM perform different functions within the vehicle. The VSM is capable of driving fully autonomously, however it makes use of a different set of input sensors than the FUN and is intended only for short-term use when the FUN is failing. The VSM is also expected to be a somewhat simpler control algorithm than the FUN. Most likely there would be little commonality between the two. Therefore, all four of these components would have very different functional requirements, resulting in very diverse development, verification, and validation.
- Most of the diverse subsystems have moderate to high complexity and performance requirements.

Complexity of validation

This architecture has a high number of diverse subsystems requiring independent validation.

- The FUN, SFM, CSM, and VSM subsystems perform different functions within the vehicle. The FUN and SFM subsystems are most closely coupled to each other and probably require joint validation.
- The CSM and VSM are less closely coupled to the FUN and SFM, which could decrease the validation effort compared to a fully integrated system.
- Ensuring sufficient independence between the FUN, SFM, CSM, and VSM subsystems is facilitated by them having different roles. This can allow for using different functional approaches, e.g., for perception, environment modeling, prediction, and/or planning.

4.2.7.6 SAFETY OF THE INTENDED FUNCTIONALITY

Support to accommodate functional insufficiencies

This architecture uses different roles with respect to SOTIF aspects for the subsystems.

- The SFM is explicitly intended to act as an ODD checker.
- The FUN, SFM, and VSM perform different functions within the vehicle. They can be implemented in a diverse way on the algorithmic level.
- A diverse set of sensors can be used. A sensor set split between functional and safety aspects is suggested.

Support to manage operational conditions

The DSM architecture considers ODDs (or their absence) for each subsystem.

- The SFM is explicitly intended to act as an ODD checker.
- The different levels of degradation (e.g., “comfort stop”, “safe stop”) each imply a different ODD.
- No specific way for data collection is defined for this architecture.

4.2.8 EVALUATION OF THE AD-EYE ARCHITECTURE

4.2.8.1 AVAILABILITY

Availability of the system

AD-EYE is an asymmetric architecture which allows for two independent sensor stacks and computational channels, enabling robustness against most single-point faults and functional insufficiencies. Robustness is achieved through an Operational Envelope enforced by Channel 2a, which constrains Channel 1 under degraded conditions. However, the mechanism is sensitive to certain perception faults in Channel 2b. In rare cases of persistent sensor or perception issues, the envelope may remain overly restrictive, unnecessarily triggering a same-lane MRM (see Figure 54 and Figure 55). This is considered an acceptable tradeoff, as it significantly reduces the risk of more common functional insufficiencies, at the cost of a non-ideal MRM outcome in rare scenarios.

- Both channels are capable of executing an MRM in case of faults in the other. A simpler sensor stack in Channel 2, and its short time horizon for planning, allows for comparatively simpler design of Channel 2.
- AD-EYE employs structural diversity between Channel 1 and Channel 2 to mitigate dual-point fault modes (see Table 23). Channel 1 supports graceful degradation, and both channels can independently execute MRMs to maintain safety.

Table 23: Possible fault / functional insufficiency scenarios for the AD-EYE architecture, considering silent (s) and arbitrary (a) failure modes. Diagonal elements: single-point faults ("OK" marks an MRM or better). Off-diagonal elements: dual-point faults ("OK" marks blind braking or better).

	C1 (s)	C1 (a)	C2a (s)	C2a (a)	C2b (s)	C2b (a)	S1 (s)	S2 (s)
C1 (s)	OK	n/a	OK (blind braking)	OK	OK (blind braking)	Incorrect (unsafe)	OK	OK
C1 (a)		OK	Incorrect (unsafe)	Incorrect (unsafe)	Incorrect (unsafe)	Incorrect (unsafe)	OK	OK
C2a (s)			OK	n/a	OK	OK	OK	OK
C2a (a)				Same lane MRM (worst case)	OK (same lane MRM or blind braking)	Incorrect (unsafe)	OK	OK
C2b (s)					OK	n/a	OK	OK
C2b (a)						Same lane MRM (worst case)	OK	OK
S1 (s)							OK	OK (blind braking)
S2 (s)								OK

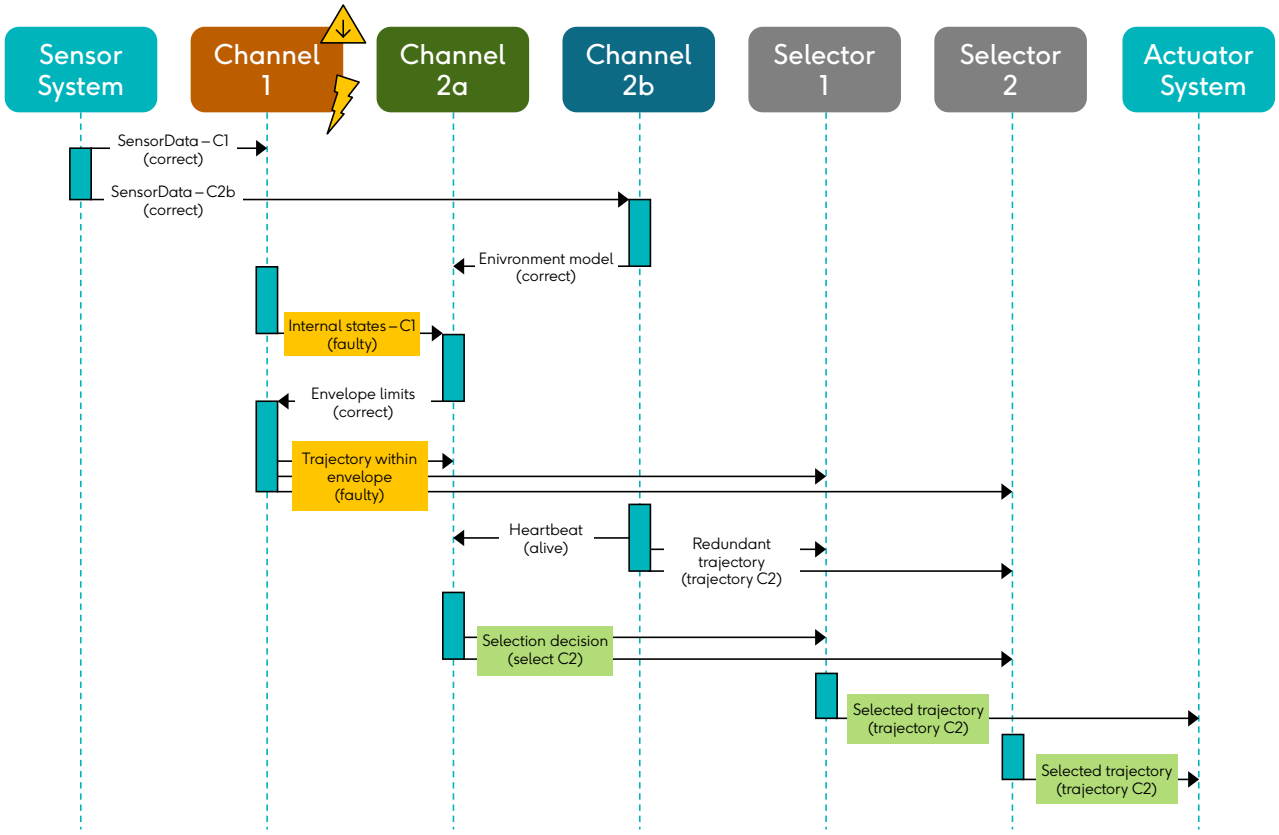


Figure 53: Sequence diagram of the AD-EYE architecture. The case with a fault or functional insufficiency in Channel 1 is shown. Faulty messages are highlighted in yellow, fault containment in green.

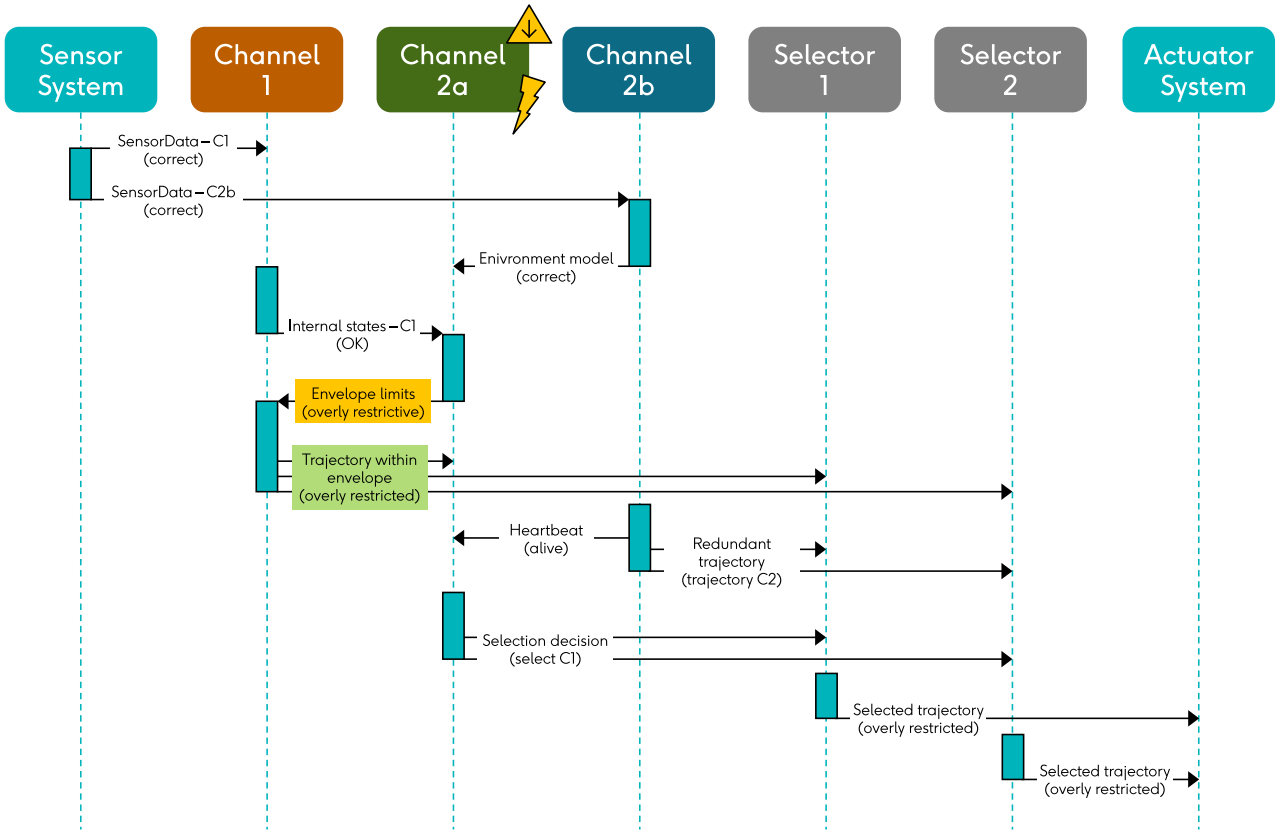


Figure 54: Sequence diagram of the AD-EYE architecture. The case with a fault or functional insufficiency in Channel 2a is shown. Faulty messages are highlighted in yellow, fault containment in green.

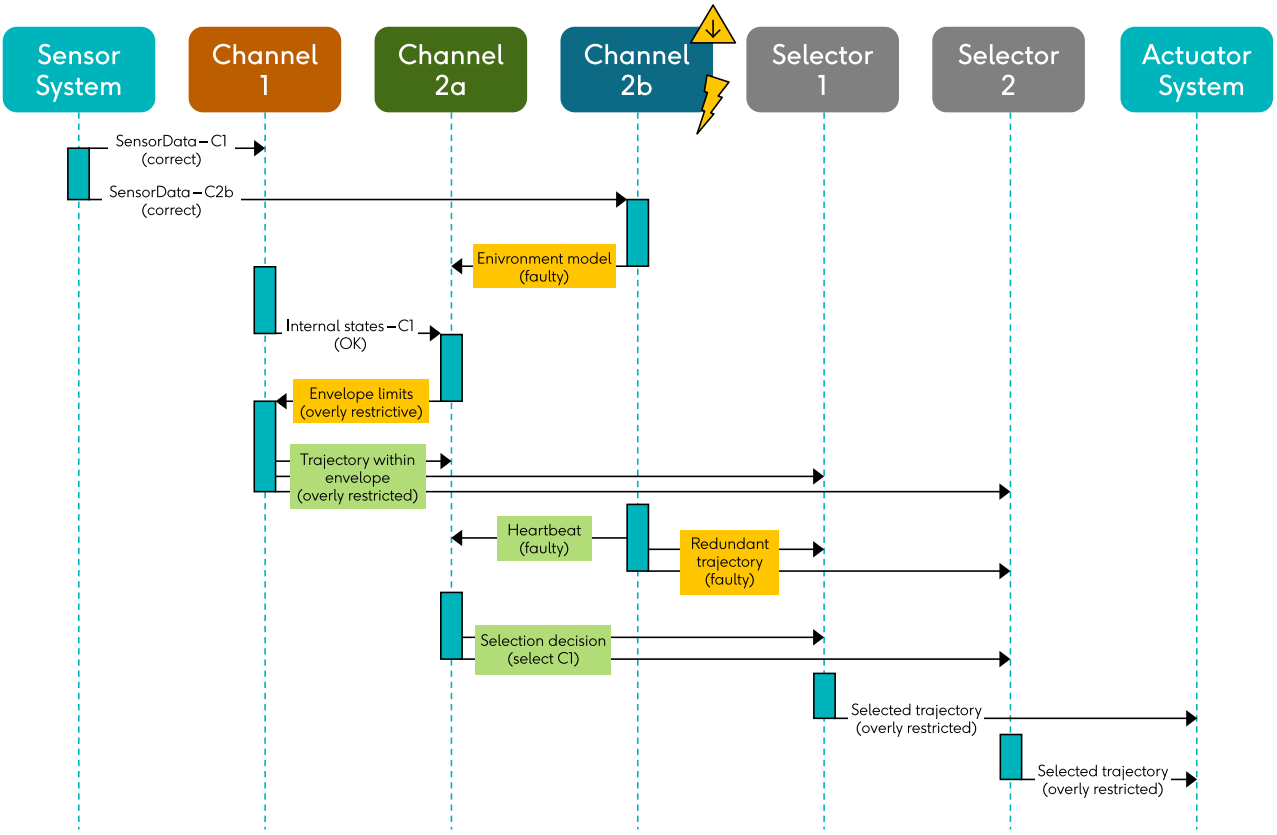


Figure 55: Sequence diagram of the AD-EYE architecture. The case with a fault or functional insufficiency in Channel 2b is shown. Faulty messages are highlighted in yellow, fault containment in green.

Diagnostics scheme

The AD-EYE architecture supports mutual awareness between channels.

- Channel 2 monitors the outputs, internal states, and actuator commands generated by Channel 1, verifying plausibility and consistency relative to its own independent perception and environmental model. Channel 2 defines and enforces an operational envelope within which Channel 1 must operate. Channel 1 is aware of the condition of Channel 2 through system diagnostics and responds by initiating an MRM in case of an error in Channel 2.
- Adaptation is based on predefined mechanisms triggered by cross-channel diagnostics. Channel 2 supervises the operational envelope of Channel 1, progressively constraining its actions if degradations are detected, or relaxing constraints if conditions recover. Upon detecting a fault or implausibility in Channel 1, Channel 2 inhibits actuator commands from Channel 1 and commands execution of a fallback trajectory. Conversely, if Channel 2 becomes faulty, Channel 1 autonomously initiates an MRM.

Degradation scheme

The AD-EYE architecture supports multiple levels of degradation to maintain safety and predictability. One of the design principles it is built on is the effective handling of transient faults that arise as a result of complex AD functionality.

- Errors result in controlled transitions that are noticeable but not abrupt. Graceful degradation leads to limited functionality without compromising driving stability, while fallback to MRM produces deliberate stopping behavior that is visible and understandable to the user.
- Channel 1 can degrade gracefully by continuously constraining its nominal functionality when non-critical faults are detected, thereby preserving user experience and progress towards goal, and healing from transient faults. If conditions worsen, the system progresses to full fallback by executing an MRM. This staged approach ensures that degradation occurs gradually, with predefined transitions that prioritize user safety. For non-severe detected faults in Channel 1, Channel 2 can also trigger a change in Channel 1's goal to a safer location, such as an off-highway area or workshop, if available. This action, used in conjunction with the Operational Envelope, further enhances traffic safety.

4.2.8.2 NOMINAL FUNCTIONALITY

Availability of nominal functionality

Availability of the nominal functionality is addressed through structured simplicity, redundancy, and degradation handling. See section 4.2.8.1 for details on the Operational Envelope degradation.

- False positives are an expected side effect of complex perception systems, the architecture is designed to handle these, rather than avoid them entirely. The separation of duties across channels ensures that functional insufficiencies or transient faults in Channel 1 do not immediately trigger an MRM, but rather a constraining of its Operational Envelope.
- Degradation is gradual and continuous, with a limp-home mode built in to take the vehicle off the roads to safe stop locations outside if necessary.
- The arbitration component, the Selector, acts as a multiplexer and does not have decision-making or evaluation capabilities.
- The decisions made by the Selector can be represented by a few lines of pseudocode.

4.2.8.3 CYBERSECURITY

Interactions between subsystems

The separation between Channel 1, Channel 2 and the selector Unit creates defined Fault Containment Units and requires several simultaneous exploits before compromise.

- The Subsystems communicate over strictly defined interfaces to limit internal propagation paths.
- A fault or insufficiency in Channel 2 could lead to a lack of availability by a constrained Operational Envelope, but it cannot directly influence or invalidate the trajectory from Channel 1.
- The communication between Subsystems is frequent but quite limited in extensiveness.
- The interfaces are well defined and static, without large amounts or dynamically defined data being sent across subsystems.

Interactions with external systems

External interactions in the AD-EYE architecture are limited and controlled to maintain system integrity. It is assumed that the vehicle can establish secure communication to the OEM cloud to ensure updates of the maps data, and that SW can be updated securely via service centres or the cloud.

- Only Channel 1 is expected to potentially interface with external systems during run time. Channel 2 during run time remains isolated (no external connectivity) due to its importance, reducing the attack surface. Both Channels are expected to have some communication with the outside world for the purposes of SW download and calibration data. The Selector is not expected to require communication.
- As the provider of the nominal functionality, Channel 1 will likely need constant connectivity for HD maps, V2X, etc. Channel 2 only needs access to less detailed and more slowly changing information such as lanes or driving directions.
- Channel 1 requires regular updates, probably OTA. Channel 2 and Selector updates are less frequent and can be done separately.

4.2.8.4 SCALABILITY

Scalability towards higher availability

The AD-EYE architecture supports scalability in functionality and availability through modular design and configurable recovery mechanisms.

- The architecture supports availability goals beyond those of the reference case by enabling handling and recovery from transient errors, with its Operational Envelope and supporting limp-home modes in Channel 1. Channel modularity and distinct design objectives allow expansion of functional capabilities, while fallback mechanisms remain independent of minor extensions. Both the enhancement of channel functionality and the customization of MRMs are possible within the existing architectural framework without requiring fundamental redesign.
- While not explicitly foreseen in the source material, for higher availability targets Channel 1 could be enhanced through duplication of internal components or by duplicating the entire channel, with a decision-making subsystem selecting the output trajectory for the Selector.

Scalability towards different offering levels

The architecture's modular design enables efficient variant management and tailoring across vehicle classes and feature sets. Reuse and scalability across offering levels are key design principles, allowing additional capabilities—such as higher speeds or new AD features—without requiring changes to the fundamental structure.

- Channel 1 can be extended with additional functionality or performance improvements, with corresponding adjustments to Channel 2 depending on the scale of feature growth.
- All subsystems are needed across different offering levels, as they are intended to operate together. The architecture was designed for L3 and above AD functions, and reuse towards L4 (and potentially L5) is considered as a key design principle. However, reuse from an L2 system towards L4 was not considered the design intent and requires considerable effort to add, e.g., Operational Envelopes and analyze in new context.

4.2.8.5 SIMPLICITY

Number, complexity, and performance of subsystems

The AD-EYE architecture defines a limited set of subsystems with well-defined complexity and performance demands:

- Five main subsystems exist in this architecture.
- Channel 1, designed for performance, is the most complex subsystem. Channel 2 has lower complexity due to its limited time horizon, simpler sensor sets, and more deterministic, formally verifiable planning algorithms. Channel 2a has deterministic rule-based algorithms and monitors. Channel 2b's primary complexity arises from its nondeterministic perception stack. The Selector subsystems are the simplest and are fully formally verifiable.
- Performance, power consumption, and hardware demands follow the complexity order: Channel 1, Channel 2b, Channel 2a and the Selectors.

Required level of diversity

The AD-EYE architecture specifies where diversity is required between subsystems:

- Heterogeneity is required between Channels 1 and 2 to ensure sufficient independence. Channels 1 and 2 need diverse implementations including independent sensor stacks and actuation code. This may include sufficiently different hardware platforms[52], software implementations, and sensing modalities to mitigate common cause failures.
- A core design principle is to limit the number of complex channels and duplication where possible and limit the number of Channels. The design does not mandate high amounts of duplication such as with symmetric patterns like TMR.

[52] Some high-performance hardware components achieve ASIL D for systematic faults; in this case diverse hardware may not be required.

Complexity of validation

Each large Fault Containment Unit, roughly the channels and the Selector component, can be developed and verified individually, simplifying the validation process of the entire system.

- The individual subsystems can be verified independently of each other for their nominal functionality. Channel 1 and Channel 2 will have to be validated together to ensure the validation of the system.
- The architecture reduces the validation effort of the system with structured verification but does not remove the need to have system validation of the integrated result.
- Ensuring sufficient independence between Channel 1 and Channel 2 is facilitated by them having different roles and Channel 2's simplicity.

4.2.8.6 SAFETY OF THE INTENDED FUNCTIONALITY

Support to accommodate functional insufficiencies

Functional diversity between Channel 1 and Channel 2b supports SOTIF objectives. By design, the fallback channel (2b) is simpler and based on different assumptions and processing, minimizing common insufficiencies between the channels.

- Each channel handles the ODD, OEDR, maneuvers, and traffic rules independently.
- The diversity in the channels incl. sensors help complement each other and compensate for SOTIF issues. In case of mismatch, common with complex sensors, the system handles the issue akin to a transient error and constrains the operational envelope, allowing for a recovery before resorting to an MRM if the error persists.
- A diverse set of sensor modalities is required for the channel and is addressed. The architecture does not specify or recommend specific sensor modalities to be limited to specific channels.

Support to manage operational conditions

ODDs are monitored in each channel independently, with no overlap of perception stacks.

- ODDs are monitored in each channel independently.
- The architecture does not define driver control during operation. Activation of the system is only allowed when Channel 2 allows it. Once activated, the system does not disengage until an MRC is reached. Degraded modes and emergency modes are handled by the architecture.
- The architecture supports the collection of SPIs via aggregating data and the connectivity in Channel 1. The data collection could be in real time, but there are no recommendations provided on data collection.

4.3 SPECIFIC EVALUATION IN THE CONTEXT OF THE REFERENCE AD USE CASE

4.3.1 RELEVANCE OF THE EVALUATION CRITERIA IN THE CONTEXT OF THE REFERENCE AD USE CASE

Depending on the selected use case, some KPIs may be more relevant than others. For instance, scalability (defined as a measure of an architecture's capability to be stepwise developed by extending SAE L2 systems) will likely not be relevant for urban SAE L5 robotaxis, which tend to be developed from scratch and are not intended to be sold as optional functions of standard OEM offerings to end users.

Conversely, scalability may be highly relevant for a Highway Pilot function, which might be developed as a natural extension of highway-oriented L2 applications. In the following, we attempt to give an assessment of the selected KPIs in the context of the reference use case of an SAE L4 Highway Pilot. We employ the following ratings for the KPIs:

- Must-have
- Important
- Beneficial

Readers of this document are encouraged to apply their own ratings to match their specific use cases, innovation space, and constraints from commercial, technical, or legacy requirements.

AVAILABILITY

In the context of an L4 Highway Pilot, availability of the system until successful completion of an MRM will become a formal safety goal with an ISO 26262 ASIL D target to be met (pending a formal HARA being conducted), as a system failure in a dense highway traffic situation will usually not be controllable by the driver and the consequences of a crash might be fatal. Therefore, availability is rated as a **must-have** for the reference use case.

NOMINAL FUNCTIONALITY

In the reference use case of an SAE L4 Highway Pilot, reliability (defined as the continuous availability of the full, nominal functionality) is highly desirable from a vehicle user's perspective and shall be maximized. A switch to degraded functionality, e.g., executing an MRM, will be at least an annoyance or more likely disturbing for the passengers; frequent ones will lead to severe customer complaints, but will at least not lead to harm. Therefore, reliability is rated as **important** for the reference use case.

CYBERSECURITY

Vulnerability to cybersecurity threats impacts system safety, as an intruder might deactivate an essential safety mechanism or even maliciously manipulate essential autonomous driving functions like sensor inputs or trajectory planning. Still, an ADI by itself will not be able to fully avert cybersecurity risks, as many system functions (e.g., the sensors and actuators) are outside its scope and additional mechanisms like gateways and firewalls are needed. Therefore, although resilience to cybersecurity attacks is a must-have for the vehicle and the AD system as a whole, it is "just" considered **important** for the ADI system in the context of the reference use case.

SCALABILITY

Since the chosen reference use case of an SAE L4 Highway Pilot may be developed as a natural extension of highway-oriented legacy SAE L2 functions, parts of those functions (legacy sensors, ECUs, application components) might need to be incorporated into the realization of the L4 system. Conversely, L2 functions might be implemented by a subset of components of the L4 system, which has been developed from scratch. In both cases, the L4 functionality might be marketed as optional equipment, and potentially a significant share of the overall vehicle volume might support L2 functions only. To implement such a concept in a commercially viable way, scalability is considered **beneficial** for the reference use case.

SIMPLICITY

Although simpler than SAE L5 functions or urban use cases, the algorithmic and system complexity for an SAE L4 Highway Pilot remains high and verification and validation efforts might be prohibitive if not supported by a suitable system architecture. The established concept of "divide and conquer", i.e., a conceptually clean, modular architecture with a well-arranged number of components of clear purpose, simple interfaces, and clear delimitations to each other, will be at least **important if not a must-have** for the reference use case.

SAFETY OF THE INTENDED FUNCTIONALITY

SOTIF is probably the key property and requirement that laypersons and the general public associate with autonomous driving functions, and technical as well as authority reports about incidents with autonomous vehicles mostly focus on function aspects and deficiencies, e.g., object detection capabilities. In the context of the reference use case, SOTIF is considered a **must-have.**

4.3.2 ASSESSMENT OF THE CANDIDATE ARCHITECTURES UNDER THE EVALUATION CRITERIA

4.3.2.1 AVAILABILITY ASSESSMENT

Variant	Assessment
Single-Channel	LOW This architecture is obviously very sensitive to single points of failure. To make it somewhat resilient to such failures, several "internal" redundancy measures will likely need to be installed in a detailed architecture phase or even in an implementation phase, potentially in an ad-hoc way. This makes it harder to verify their sufficiency and completeness.
Majority Voting	LOW While this architecture with homogeneous channels addresses random HW faults well, it is susceptible to common cause failures, as the complex AD algorithms will not usually be suitable for full ASIL D development. Conversely, heterogeneous channels are not suitable for voting, as channels might each exhibit different (but valid) driving policies, and therefore a faulty channel might not be identifiable. This is especially true for the practical case of 2oo3 (TMR), where the problem is likely not solvable.
Cross-Checking Pair	MEDIUM This architecture can only properly deal with failure modes that can be detected via a cross-check with another channel (e.g., FuSa faults). However, in the general case, a single fault can lead to an irresolvable conflict which leads to resorting to a pre-planned MRM, which would only be acceptable for very rare fault combinations.
Daruma	MEDIUM This architecture does not exhibit any obvious single point of failure. It relies on three diverse channels to reduce common cause faults, though this can be harder to achieve if the roles of all channels are similar, e.g., providing similar functionality. It relies on an independent subsystem performing cross-checks to arbitrate between channels. However, a failure of this subsystem can immediately lead to a severely degraded MRM.
Channel-Wise DCF	HIGH This architecture exhibits no obvious single point of failure (provided that the D-System is implemented in a fault-tolerant way) and due to the asymmetric approach, with its diversity of the channels, also has a high potential to rule out common cause faults.

Variant	Assessment
Layer-Wise DCF	HIGH This architecture can rely on its Primary and Safing channels plus the MSTOP (blind stop) capability. However, the published description suggests potential single points of failure that would need to be avoided, e.g., using the same occupancy grid in the primary and secondary channels (whereas using that same input for the planners and safety gates of each channel can be beneficial to avoid false positives - if used correctly to restrict the planner's decision space, not to extend it). Also, the similar structure of the Primary and Safing channels suggests sensitivity to common cause faults in the underlying implementation, which would need to be avoided.
DSM	HIGH This architecture is intended to support the availability KPI, due to its multiple layers and redundancy mechanisms. It has some similarity to the Channel-Wise DCF architecture but offers additional degradation steps, giving in principle the potential for higher availability; in the concrete implementation proposed, it does not have a clear separation of the functional (FUN) and monitoring (SFM) channels, but implements both within the same SOC and virtual machine, and seems therefore highly sensitive to common cause faults. We consider this an implementation aspect and therefore exclude it from our analysis.
AD-EYE	MEDIUM This architecture does not exhibit any obvious single point of failure. However, a failure of one of the subsystems in Channel 2 can in the worst case lead to an in-lane emergency stop.

CONCLUSION:
For the reference AD use case of an SAE L4 Highway Pilot, the Channel-Wise DCF and the Layer-Wise DCF seem to be the architectures of choice from an availability point of view. DSM is considered problematic due to its common cause failure sensitivity. Majority Voting is considered highly problematic (if not unsuitable) in the practical case of non-deterministic channels. Single-Channel is considered unsuitable.

4.3.2.2 NOMINAL FUNCTIONALITY ASSESSMENT

Variant	Assessment
Single-Channel	MEDIUM The resilience to functional deficiencies is not supported by any architectural measure but depends strictly on the internal implementation of its subcomponents.
Majority Voting	LOW This architecture can only provide high reliability if the output data structures are easily comparable (ideally binary yes/no). In this case, the similar capability level of each channel allows for full nominal functionality. For more complex data structures (e.g., trajectories), the voting process becomes prone to disagreement even without faults or functional insufficiencies.
Cross-Checking Pair	MEDIUM As both channels in this architecture are of similar capability level, the rate of disagreements (false positives) could be lower than for asymmetric architectures.
Daruma	HIGH The degree of degradation in this architecture strongly depends on the relative capability of the channels, i.e., from full nominal functionality to degraded MRM. The ranking of channels can adapt quickly and dynamically and choose the most appropriate channel for each situation.
Channel-Wise DCF	MEDIUM The degree of reliability in this architecture depends on the concrete capability level of its L2- and M-Systems, and on the parameterization of the M-System (which initiates the potentially degraded mode of the F-System), to not produce false positives. For the L2-System, being aware of the limits that will be enforced by the M-System would be a helpful addition to the architecture.
Layer-Wise DCF	HIGH The degree of reliability in this architecture depends on the concrete capability level of its Primary channel and the monitoring subsystems contained therein. To not produce false positives, it may foresee precautions like restricting the primary's decision space by the limits imposed by the monitor (although this is not detailed in the published description).
DSM	MEDIUM The Distributed Safety Mechanisms architecture will provide high reliability, due to its multiple and differentiated degradation steps.
AD-EYE	HIGH This architecture allows for a gradual constraining of Channel 1 via envelope feedback. The simpler Channel 2b is only used if Channel 1 isn't available at all.

CONCLUSION:
For the reference AD use case of an SAE L4 Highway Pilot, Majority Voting, Cross-Checking Pair,

Daruma, DSM, and AD-EYE seem to be the most capable architectures to sustain nominal functionality. However, the latter can easily suffer from disagreements in the voting process. Channel-Wise DCF and Layer-Wise DCF can approximate this to some extent but will fall into degraded mode more often (due to their focus on safety). Single-Channel does not support reliability at the architectural level; it is strictly implementation-dependent.

4.3.2.3 CYBERSECURITY ASSESSMENT

Variant	Assessment
Single-Channel	LOW This architecture is critical from a cybersecurity point of view, as its single channel does not provide any architectural partitioning but exposes its complete functionality to a malicious intruder.
Majority Voting	HIGH In this architecture, the channels are highly separated and exchange little (if any) information, and the voting component itself is expected to be simple and well-separated. However, if the channels are implemented homogeneously, it could be highly susceptible to exposing a common vulnerability.
Cross-Checking Pair	LOW In this architecture, almost all subsystems communicate with each other. In addition, a successful attack on either one of the channels would compromise the overall system.
Daruma	HIGH In this architecture, the different subsystems are clearly separated, diverse, and (unidirectionally) exchange only small amounts of data.
Channel-Wise DCF	HIGH In this architecture, clearly separated components exchange only a small amount of well-defined information and are highly diverse, potentially avoiding common vulnerabilities. The (simple) decision logic poses the only single point of attack.
Layer-Wise DCF	HIGH This architecture makes use of clearly separated components. However, a successful attack on one single (of several) safety gates would compromise the overall system.
DSM	MEDIUM This architecture seems to be more vulnerable from a cybersecurity point of view, as its high number of interactions between subsystems and (partly) missing separation might make it more exposed to attackers.
AD-EYE	HIGH This architecture uses clearly separated components. The channels could be updated independently from each other.

CONCLUSION:
For the reference AD use case of an L4 Highway Pilot, Daruma, Channel-Wise DCF, and AD-EYE have the highest resilience against cybersecurity attacks, followed by Layer-Wise DCF with its higher exposure due to the larger number of safety gates. DSM seems to be more vulnerable due to its high number of interactions and tightly coupled components. Majority Voting will be

vulnerable in the case of homogeneous channels. Cross-Checking Pair suffers from similar problems as under the reliability KPI. Architecture-wise, Single-Channel does not provide any protection from cybersecurity threats.

4.3.2.4 SCALABILITY ASSESSMENT

Variant	Assessment
Single-Channel	LOW This architecture does not provide any scaling options.
Majority Voting	HIGH This architecture appears to be highly scalable, as one of its channels can be used to downscale to an SAE L2 system or could be derived by extending an existing L2 system. Likewise, it could be upscaled by adding more channels.
Cross-Checking Pair	LOW While this architecture could technically be extended with more channels, this would also have an impact on existing channels. This is due to the fact that the cross-checks are integrated in the channels, which also makes re-use as an SAE L2 system more difficult.
Daruma	HIGH By design, this architecture is intended to be highly scalable. Existing SAE L2 systems can be re-used as channels by only adding a few standardized interfaces. More channels can also be added without changing existing channels as the cross-checks are performed in the dedicated Daruma subsystem.
Channel-Wise DCF	MEDIUM The main component of this architecture (L2-System) can be used to downscale to an SAE L2 system or could be derived by extending an existing L2 system.
Layer-Wise DCF	MEDIUM The primary components of this architecture can be used to downscale to an SAE L2 system or could be derived by extending an existing L2 system.
DSM	HIGH Some of the components of this architecture may be derived by extending an existing SAE L2 system. It is not obvious, however, how it could be downscaled to an L2 system by essentially just removing L4-related components. It could be upscaled to L5 by adding FUN/SFM components, but its VSM would need to be substantially extended.
AD-EYE	MEDIUM Significant modifications, e.g., to support envelope feedback, would be necessary to reuse an existing L2 system as a subsystem within this architecture. Scaling to higher availability would require adding internal complexity to Channel 1 or duplicating it with some additional decision logic.

CONCLUSION:
For the reference AD use case of an SAE L4 Highway Pilot, Majority Voting and Daruma seem

to be the best options to both downscale to L2 or upscale to L5. Channel-Wise and Layer-Wise DCF seem to provide good capabilities to downscale to an L2 system or leverage L2 system developments, whereas DSM seem to be a better fit for upscaling to an L5 system. AD-EYE requires some significant modifications to scale, while Cross-Checking Pair requires many. Finally, Single-Channel does not support scaling at all.

4.3.2.5 SIMPLICITY ASSESSMENT

Variant	Assessment
Single-Channel	LOW This architecture superficially seems to be simple, but its monolithic approach and lack of clearly separated subsystems will lead to high complexity and effort for implementation, verification, and validation.
Majority Voting	LOW The regular structure of this architecture implies simplicity. However, each individual channel might have similar properties to the Single-Channel architecture, with comparable consequences for implementation and verification efforts. Relaxation on the individual channels due to the subsequent voting might be offset by efforts to identify faulty channels correctly. This might be a challenge for the system integrator, and for 2oo3 (TMR) as a practical option, it is questionable whether this can be solved at all.
Cross-Checking Pair	HIGH This architecture has fewer subsystems than most other architecture candidates, although both of its channels have similar functional capability. The cross-check between them may complicate independent development and verification.
Daruma	MEDIUM The subsystems in this architecture are not directly coupled, which can facilitate independent development and verification. The implementation effort strongly depends on the relative functional capability of the channels, e.g., whether more than one of them provides the full nominal functionality, which can be costlier than asymmetric architectures.
Channel-Wise DCF	HIGH This architecture is conceptually simple, as its clearly separated components with distinguished purposes and well-defined message exchange enable modular, separate development and simpler overall safety assessment. The L2-System will be the most complex component, but less effort than one of the Majority Voting's channels, due to the highly diverse M-System supervision. Efforts for the system integrator seem to be reasonable.
Layer-Wise DCF	MEDIUM This architecture has clearly separated components with distinguished purposes and well-defined message exchange, but the higher number of components and level of information exchange (compared to the Channel-Wise DCF) will increase the burden on the system integrator and complicate overall safety assessment. On the other hand, the layered approach might enable a more modular, separate development.

Variant	Assessment
DSM	MEDIUM The FUN, SFM, CSM, and the VSM modules of the Distributed Safety Mechanisms are separated, enabling separate development and verification, but interact in complex and highly diverse ways, thus putting a high burden on the system integrator and complicating the overall safety assessment. The proposed architecture also mixes functional aspects (FUN, SFM, VSM) with system integrity aspects (CSM) and implementation aspects (middleware, virtual machine) – it will thus be critical to decompose and assign the required system properties to the entities of the architecture in a clear, consistent, and complete way.
AD-EYE	HIGH This architecture is conceptually very simple. It consists of two diverse channels with distinct roles. One of these comprises two subsystems that share perception components, minimizing implementation and production cost.

CONCLUSION:
For the reference AD use case of an SAE L4 Highway Pilot, Cross-Checking Pair, Daruma, Channel-Wise DCF, Layer-Wise DCF, and AD-EYE seem to be preferable with respect to simplicity, i.e., reasonable integration and validation efforts. DSM is highly complex, especially for the system integrator. Majority Voting seems simple, but the individual channels' complexity is high, as is their integration in the case of heterogeneous channels. Single-Channel is superficially the simplest but is expected to require high verification and validation efforts.

4.3.2.6 SOTIF ASSESSMENT

Variant	Assessment
Single-Channel	LOW Due to its monolithic nature, this architecture is obviously very sensitive to functional deficiencies and to deviations from the nominal conditions, especially since it will need to rely on machine learning to achieve the performance goals and there is no visible means of supervision or diversity.
Majority Voting	LOW If homogeneous channels are used in this architecture, it is susceptible to common cause failures by functional deficiencies or deviations from the nominal conditions. Conversely, heterogeneous channels are not suitable for voting, as channels might each exhibit different (but valid) driving policies, and therefore a faulty channel might not be identifiable. This is especially true for the practical case of 2oo3 (TMR), where the problem is likely not solvable.
Cross-Checking Pair	HIGH The two channels in this architecture can compensate for each other's weaknesses, specifically combined with diversity. Some complex situations may still lead to a conflict and deadlock.

Variant	Assessment
Daruma	HIGH This architecture explicitly foresees that channels may each focus on particular driving situations and compensate for each other's weaknesses. This may also allow it to address deviations from the nominal conditions.
Channel-Wise DCF	HIGH This architecture exhibits a natural diversity between the L2-System and the M-System. The F subsystem (being explicitly foreseen for out-of-ODD operation) is likely implemented very differently from the L2-System. This architecture also quite naturally manages changes to the nominal conditions.
Layer-Wise DCF	HIGH This architecture employs multiple checkers and safety gates both on its primary and secondary channels, promoting modularity and diversity, which also has a positive impact on development and V&V. It may be capable of addressing deviations from the nominal conditions well, but this seems not to be explicitly foreseen.
DSM	HIGH This architecture highly promotes SOTIF, due to the different functions performed by its FUN, SFM, and VSM modules. Management of off-nominal conditions is also explicitly foreseen.
AD-EYE	HIGH This architecture employs two diverse channels with independent, diverse perception stacks such that they can compensate for each other's functional insufficiencies.

CONCLUSION:
For the reference AD use case of an L4 Highway Pilot, the Daruma, Channel-Wise DCF, the Layer-Wise DCF, DSM, and AD-EYE are the architectures of choice from a SOTIF point of view. Majority Voting is highly problematic (if not unsuitable) in the case of heterogeneous channels, as correct voting cannot be ensured. Majority Voting with homogeneous channels and the Single-Channel architecture are highly sensitive to functional deficiencies and off-nominal conditions, and therefore likely unsuitable.

4.3.3 EVALUATION SUMMARY

Table 24 summarizes the evaluation findings, i.e., whether and how well the criteria are supported by the respective candidate architectures. It should be noted that this evaluation is only qualitative, i.e., these are not scores and should not be overinterpreted or summed up.

Table 24: Summary of the specific evaluation.

	Availability	Nominal Functionality	Cybersecurity	Scalability	Simplicity	SOTIF
Single-Channel	LOW Not supported	MEDIUM Higher internal complexity and no backup to support degradations	LOW Not supported	LOW Not supported	LOW High inner complexity	LOW Not supported
Majority Voting	LOW Disagreements between channels can be problematic	LOW Voting can lead to more problematic disagreements	HIGH Limited interactions	HIGH Omit/add channels to scale	LOW Simple architecture, complex channels	LOW Not supported due to homogeneous architecture
Cross-Checking Pair	MEDIUM Disagreements between channels can lead to preplanned MRM	MEDIUM Reasonable reliability	LOW Strong coupling between channels	LOW Some reuse possible	HIGH Simple architecture; fewer channels	HIGH Structure supports SOTIF
Daruma	MEDIUM Non-redundant checking subsystem can be problematic	HIGH Multiple ranked channels can provide nominal functionality	HIGH Limited interactions	HIGH Omit/add channels to scale	MEDIUM Simple architecture; channel complexity implementation-specific	HIGH Structure supports SOTIF
Channel-Wise DCF	HIGH Concept focuses on availability	MEDIUM Reasonable reliability; implementation-dependent	HIGH Diverse structure with high resilience	MEDIUM Omit channels to downscale to L2	HIGH Simple architecture; low required channel complexity	HIGH Structure supports SOTIF

	Availability	Nominal Functionality	Cybersecurity	Scalability	Simplicity	SOTIF
Layer-Wise DCF	HIGH Concept focuses on availability; risk of single point failures	HIGH Reasonable reliability; with precautions by architecture	HIGH Diverse structure, but multiple single-attack points	MEDIUM Omit channels to downscale to L2	MEDIUM Structured, medium complexity architecture; more channels	HIGH Structure supports SOTIF
DSM	HIGH Sensitivity to common cause failures	MEDIUM Reasonable reliability with differentiated degradation steps	MEDIUM Diverse structure, but complexity might expose vulnerabilities	HIGH Downscaling to L2 not straightforward, but potential to upscale to L5	MEDIUM High complexity for the integrator	HIGH Structure supports SOTIF
AD-EYE	MEDIUM Failure in Channel 2 can in the worst case lead to in-lane stop	HIGH Gradual degradation possible via envelope feedback	HIGH Diverse structure with high resilience	MEDIUM Some significant modifications necessary to scale	HIGH Simple architecture with minimal duplication of functionality	HIGH Structure supports SOTIF

In general, we find that asymmetric architectures (Channel-Wise DCF, Layer-Wise DCF, DSM, and AD-EYE) are better suited than symmetric ones (Single-Channel and Majority Voting) for the complexity in Automated Driving. The Cross-Checking Pair and Daruma architectures, while technically symmetric, can also be constructed with rather different channels and resemble the asymmetric architectures.

Their quite naturally independently developed and complementary channels can compensate for each other's weaknesses, compared to the essentially identical or potentially even monolithic implementations of symmetric architectures.

The asymmetric architectures basically employ two design patterns and combine them in different ways:

- **Doer / Checker:** One subsystem performs the function, the other one monitors it.
- **Active / Hot Stand-By:** One subsystem is active, and the other is on stand-by; if the active is unavailable or unsafe, the stand-by takes over.

A combination of these patterns allows for a sound partitioning into modules with simple and purposeful interfaces that can be independently verified and whose integration is straightforward and readily verifiable. Also, these patterns lead to a limited, well-arranged and predictable number of system states under both nominal and off-nominal conditions. A solid and verifiable safety argumentation can then be constructed systematically based on the individually developed modules and their successful integration.

5 IMPLEMENTATION CONSIDERATIONS

The implementation of the AD Intelligence's conceptual system architecture is embodied in concrete HW and SW architectures. The following section collects implementation considerations for the process of mapping conceptual architectures to such more detailed designs. Sections 5.1 and 5.2 summarize considerations related to HW and SW, respectively. The design and implementation must follow applicable standards, summarized in section 5.3, and satisfy the general requirements listed in section 2.2, prominently among them sufficient independence, which is discussed in more detail in section 5.4. Finally, surrounding systems (see section 1.2) also need to be considered, but this goes beyond the scope of this document.

5.1 HW CONSIDERATIONS

In this section, a short introduction to some aspects of HW refinement of the conceptual system architecture is given. This is not a complete list of aspects.

5.1.1 HIGH AVAILABILITY AND VEHICLE OPERATING STATES

Architectures in the Automated Driving (AD) context in general should not limit their functionality to a dedicated vehicle operating state (like parking, standing still, driving slowly etc.), but should work in fault-free condition in all vehicle states. But the loss of the functionality can lead to a hazardous event only in specific vehicle operating states. One valid approach for degradation of Automated Driving functionality is to change the operating state to lower severity or exposure, for instance by lowering the vehicle speed in a controlled way. In this context, please check ISO 26262-10:2018 §12.

Emergency operation exposure time as reaction to a fault should be limited if the ASIL capability of the item is lower than the ASIL rating of the possible hazard. If after the occurrence of the fault, the vehicle operating states are not changed, then the ASIL is the same as that derived from the HARA and no ASIL decomposition of main path and a potential redundant path is allowed.

For redundant paths, a dependent failure analysis should be executed to find and eliminate common cause initiators.

5.1.2 COMMON CAUSE INITIATORS

ISO 26262:2018-9 §7 requires assessment of Common Cause Initiators (CCI):

a. **Random HW faults**
   Many of the system architectures use redundant channels to mitigate random HW faults in one channel by providing the same function in the redundant channel. There is a very small chance that in the Safety Goal-relevant time interval a random HW fault is detected in the redundant channel as well. A good strategy to treat this case is to use diverse configurations of fault reactions in the first and redundant channel.

b. **Development faults**
   Those are covered by ASIL D development process, requirement-driven development flow and tool chain qualification process. A Development Process Documentation (DPD) can provide information related to the following topics:

i. Development Process

ii. Development Environment

iii. Requirement Management

A Quality Process Documentation (QPD) can provide references to cover various quality management and production-related topics. Both contribute to Quality Process Management with the goal of providing the best avoidance of systematic development faults.

c. Manufacturing faults

The manufacturer should monitor the compliance with the related standards, e.g., by Audits, Production Assessment and Process FMEA activities. Particular focus should be put on the verification by testing of characteristics determined by analogue circuitries, including in quasi-digital parts such as memory. ISO/TS16949 certificates should be provided by all manufacturing sites documenting process compliance. An ISO 9001 certificate covers all sites, locations and organizational units of a manufacturer. TS16949 certificates cover all production sites, headquarters, automotive design centers, and sales. AS 9100 certificates cover production sites in America.

The Production Part Approval Process (Produktionsteil-Abnahmeverfahren, PPAP) [AIAG] is comparable to the PPA Production Process and Product Approval (PFF Produktionsprozess- und Produkt- Freigabe) [VDA]. Both procedures are reflected in the ZVEI PPAP Guideline. The Automotive Industry Action Group (AIAG) has developed a common PPAP standard as part of the Advanced Product Quality Planning (APQP) to use a common terminology and standard forms to document project status. Companies may have their own individual requirements. PPAP is the documentation (snapshot) of the current state of the product design, functionality, and reliability as well as the production processes used.

d. Installation faults + e. Repair faults

This CCI category shall be mainly addressed by OEM and TIER1 suppliers. The guidance given by suppliers in their user manuals and safety application guidelines shall be obeyed.

f. + h. Environmental factor incl. stress

The prototypes and series components of ADS should be subject to Environmental Stress as defined by semiconductor standards and by the automotive industry, such as AEC-Q100. OEM and their TIER1 suppliers shall analyze the potential impact on their diversity claim.

g. Common external resources

The functional safety of external resources, such as power supply, debug support and communication interfaces shall be analyzed for potential common causes to redundant channels.

5.1.3 CLOCK, POWER, RESET, DEBUG, AND TEST FAILURES

Infrastructure functions in Automated Driving systems are typically common cause initiators on the hardware level.

The clock configuration of an automated driving system is defined during the development phase and is usually static during runtime. Therefore, any systematic failure affecting its functionality or monitoring capability is assumed to be found during integration verification. A diverse crystal oscillator type or PCB layout for redundant and diverse paths can reduce dependent failures. Diverse configuration settings of clock upscaling and distribution can reduce dependent failure.

Systematic faults in power supply circuits can affect the voltage regulator characteristic. Extreme corner cases (which could escape system validation) are unlikely to happen identically in

redundant and diverse paths for an Automated Driving system.

During the operation of an Automated Driving system, a reset of hardware components can be used as a reaction to detected faults, but also has a high impact on the availability of the system. Reset as failure reaction should be used only for failures which cannot be handled otherwise. A diverse configuration of all reset sources (especially fault reaction) can reduce dependent failures. Under certain conditions, the redundant (hot standby) path can be configured to ignore all reset sources (in the case of a fault detected in the active path).

Debugging is meant to be used during SW development only, therefore its systematic failures do not affect the functionality of the Automated Driving system during runtime. During safety application, all debug functionality should be disabled. The only remaining systematic faults could result from SW activation with critical failure mode "unintended debug". Diverse SW implementation can reduce dependent failure (e.g., redundant path without any debug SW parts compiled).

Test functionalities based on Built-In Self-Test (BIST) are executed during startup only, therefore its systematic failures do not affect the functionality of the Automated Driving system during runtime. During safety application, all test functionality should be removed. Only remaining systematic faults could result from SW activation with critical failure mode "unintended test". Diverse SW implementation can reduce dependent failure (e.g., redundant path without any test SW parts compiled).

5.2 SW CONSIDERATIONS

This section contains some selected topics to consider when analyzing the system architecture towards further refinement of the technical aspects at the software level.

5.2.1 SOFTWARE ARCHITECTURAL STYLES

The main aspect to consider is the software architectural design itself. An exhaustive description of software architecture styles (e.g., layered, monolithic, microkernel, pipes and filters, client-server, publisher-subscriber, event-driven) and their applications is beyond the scope of this report, but we can recommend using [64] to get a good overview. Furthermore, this report does not address how safety measures are appropriately integrated into such software architectures.

Regardless of the choice of architectural styles for individual software elements, there are common safety measures, which are listed in the following non-exhaustive list:

- Graceful degradation behavior by ensuring that there is no single point of failure, especially for middleware and service-oriented software components.
- Error detection and handling mechanisms, as described in ISO 26262 [2] Parts 6 and 10, as well as the capability to store diagnostic data.
- Use of adequate programming languages and techniques, including the application of design and coding guidelines, such as MISRA C, AUTOSAR C++, CERT.
- Performing architecture analysis, such as Failure Modes and Effects Analysis (FMEA) and Architecture Trade-off Analysis Method (ATAM) [64].
- Evaluation and optimization of metrics related to the quality aspects of the architecture (e.g., complexity, dependencies, stability of code and interfaces).

An important aspect to mention here is the shift from federated to centralized architectures in

automotive systems. In such centralized architectures, the software is executed redundantly using the mechanisms of virtualization and containerization (i.e., with hypervisors coordinating resources and virtual machines processes). Consequently, the system is more flexible and hardware costs can be reduced. On the other hand, distributing processes to virtual machines brings some challenges in terms of integration and testing, as well as cybersecurity.

5.2.2 PROPERTY OF TECHNICAL INDEPENDENCE

As stated in ISO 26262 [2] Part 9, to achieve technical independence between components of the system, cascading and common cause failures that compromise a safety requirement shall be avoided. While the factors listed in 5.1.2 apply to hardware, similar classes of coupling factors shall be considered for software elements:

- Shared resources, e.g., use of identical software modules without further independence measures, use of mathematical or other software libraries.
- Shared information input, e.g., global variables, data or messages used by more than one software element.
- Systematic coupling, e.g., same software tools, same programming or modeling language, reuse of assumptions and requirements for different software implementations.
- Components of identical type, e.g., same source code generated twice.
- Communication, e.g., global variables, messaging, function calls with arguments passed.
- Unintended interface, e.g., same memory space.

5.2.3 SOFTWARE REUSE

Reusable software (e.g., third party software, libraries, FOSS) can significantly reduce the development effort for AD systems. Variant management, software configuration and the integration into different architectures are aspects that need to be done carefully to avoid dependability issues.

In addition to the requirements and recommendations for the development of SW-SEooCs and the qualification of software components contained in ISO 26262 [2] Parts 8 and 10, there are new standardization efforts that complement these and provide further guidance:

- ISO/PAS 8926:2024 Road vehicles – Functional safety – Use of pre-existing software architectural elements [65].
- Public initiatives such as the project Enabling Linux in Safety Applications (ELISA) [66].

5.2.4 SOFTWARE UPDATES

With the move to software-defined vehicles, architectures of AD systems must ensure regular, continuous updates of software elements in a safe manner for the long term, including over-the-air (OTA) ones. This capability is closely related to quality aspects such as modularity, modifiability, portability, extensibility, and verifiability, along with the challenge of additional safety and cybersecurity risks (e.g., risks associated with the use of cloud services).

To standardize the software update engineering process, ISO 24089 [67] has been recently published. It contains requirements and recommendations on planning, risk management, V&V, deployment, and monitoring of software updates, but does not include specific technologies or solutions.

5.2.5 REAL-TIME OPERATING SYSTEMS (RTOS) AND MIDDLEWARE

The software responsible for providing basic services and interfacing the software applications with the hardware (i.e., the electronic buses, CPUs, and ECUs) requires a high level of integrity. The well-known standard AUTOSAR (AUTomotive Open System Architecture) [68] has been largely used in its original version (i.e., Classic Platform) as the basis for traditional automotive functionalities such as engine control and transmission. For AD systems, however, more complex software applications and high-performance computations are to be supported. Thus, a middleware based on the new AUTOSAR Adaptive Platform includes advanced functionalities, such as:

- Runtime configuration
- OTA software updates
- Ethernet inter-ECU communication for the transmission of large data
- High-performance hardware
- Service-oriented communication
- Compatibility with other operating systems (e.g., Linux, Android)

Other capabilities that go beyond the AUTOSAR Adaptive standard might be required, for example scheduling and real-time guarantees for event chains across complex, multi-partition or multi-SOC architectures.

5.2.6 MACHINE LEARNING AND DATA-DRIVEN APPROACHES

The use of machine learning (ML) in the automotive industry was essentially introduced to address the challenges of the perception tasks (e.g., object recognition, pedestrian detection, signs recognition, road intersection detection). While the neural network model used might play a relevant role to ensure safe outputs, the performance of ML-based software depends mostly on data engineering aspects. Typical issues to avoid during development of such software are:

- Bias in data collection
- Patterns of mislabeling in training data
- Poor design of experiments for simulation validation

Another important aspect is the potentially non-deterministic behavior of ML-based software due to the inclusion of stochastic aspects in the training process or the concrete implementation. Architectures that integrate ML-based software require safety mechanisms such as redundancy and plausibility checks (e.g., safety wrappers) which make them more complex. In general, analyzing reliability- and safety-related failure modes and mitigating them with appropriate error detection and handling mechanisms is one of the key challenges for ML-based software. ISO/IEC TR 5469 and ISO/PAS 8800 address the safety of AI and are discussed in sections 5.3.5 and 5.3.6, respectively.

5.2.7 DATA MANAGEMENT

In addition to the safety implications of data-driven approaches in the context of ML, other aspects related to data management also require decision-making at the architecture and implementation level. Some safety aspects related to data management are:

- Data and software configuration management (e.g., to support variant management and software updates).
- Assuring safety of internal-to-vehicle data management (e.g., internal maps used for vehicle localization are uncorrupted and of a compatible version).
- Integrity of collecting, storing, and transmitting engineering field feedback data, including safety performance indicators.

5.2.8 TOOL QUALIFICATION

Tool evaluation and qualification processes are described in ISO 26262 [2] Parts 8 and 10. Due to the increasing complexity of the software development environment and the technologies used, the topic has become crucial. Continuous Integration (CI), static and dynamic code analysis, testing and simulation tools, model-based code generation, documentation generation: essentially all software engineering processes are becoming automated. While this is necessary to manage development and maintenance efforts, challenges arise from increasing reliance on the tool chain and infrastructure, with all the associated risks related to troubleshooting, cybersecurity, privacy, and safety.

To the degree that simulation is used to supplant vehicle testing, tool qualification of simulations, simulation models, and simulation orchestrators will become more critical. The same applies to the tools needed to mitigate the risk of data bias, inaccurate data labels, and data corruption in such simulation-based validations.

From a system architecture point of view, the use of different tools for redundant subsystems may be necessary to rule out systematic coupling of failures due to malfunctions of the tool.

5.2.9 CRYPTO AGILITY

Crypto primitives such as ciphers or hash functions are built on the hardness of certain mathematical problems. When such hardness is broken, e.g., for SHA1, it becomes necessary that crypto primitives be replaced with appropriate ones that still have the required hardness. The main focus thus far has been on providing such hardness resistant to classical computers. With the advent of quantum computers, it is now expected that within 20-25 years the capability to break certain problems such as integer factorization or discrete logarithm problems will be available.

Asymmetric key cryptographic algorithms such as Rivest-Shamir-Adleman (RSA) and Elliptic Curve Cryptography (ECC) can be broken in polynomial time on a quantum computer due to Shor's algorithm. Moreover, symmetric key cryptographic algorithms such as AES-128 could have reduced security due to Grover's algorithm. This can be mitigated by increasing the key length, e.g., AES-256.

For automotive systems this has an impact on functionalities such as software updates done either via over-the-air or through diagnostic systems. The verification of software updates is generally done using digital signatures based on asymmetric key cryptography. Software updates are also crucial for safety and nominal functionality of the system. Hence it is imperative to have methods for verifying software updates such that overall system integrity is maintained. Moreover, any potential compromise of the software update process would also have an impact on the resilience of the vehicle architecture – opening up further opportunities for security compromise regardless of the employed design.

NIST initiated a competition in 2017 to discover algorithms that are resistant to attacks employing a quantum computer. The finalized set of Post Quantum Cryptography (PQC) algorithms from NIST that need to be supported are shown in Figure 56.

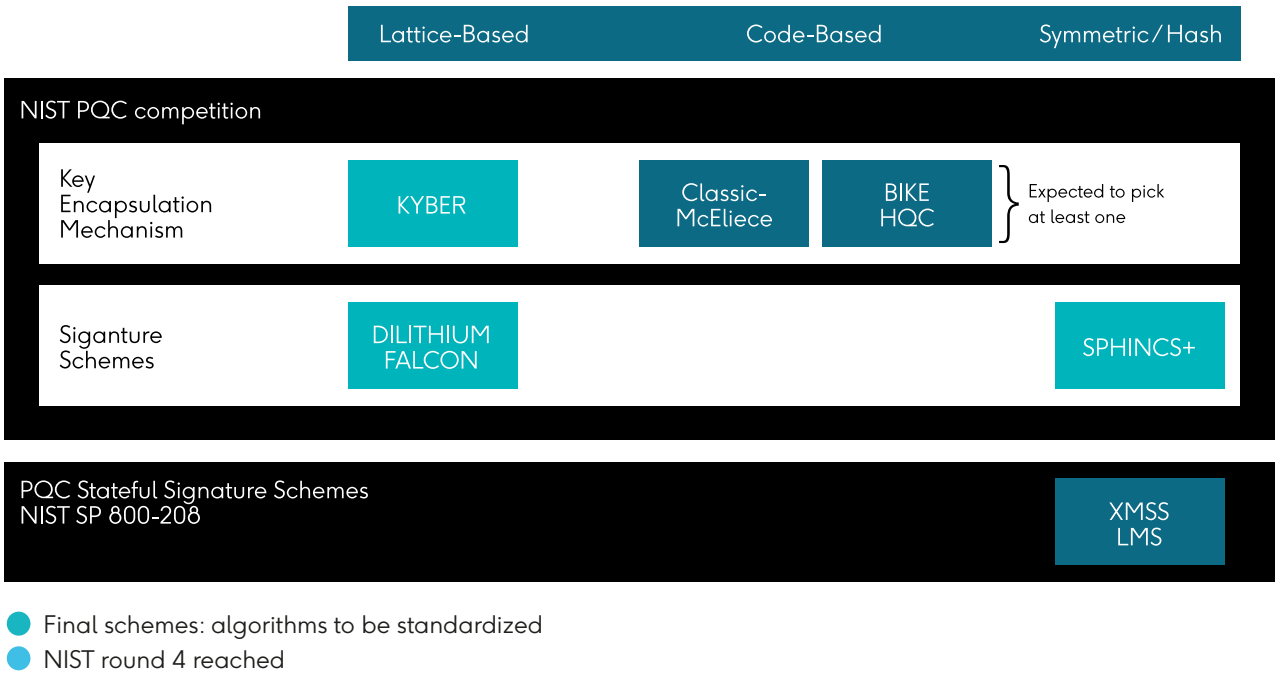


Figure 56: New algorithms in Post Quantum Cryptography (PQC).

With new PQC algorithms come new challenges for integration in use cases (e.g., secure boot, software update) and protocols (e.g., TLS). PQC algorithms are not drop-in replacements; rather they need systematic engineering to mitigate potential challenges of signal processing and increased memory consumption. This would mean changes to the hardware and software ecosystem to meet real-time platform requirements.

The evaluation of PQC underscores the importance of 'Crypto Agility', which enables the seamless exchange of cryptographic algorithms on hardware and software platforms without compromising the quality of service for road vehicles. The definition and requirements of Crypto Agility can vary depending on the perspective, encompassing solutions such as flexible hardware capable of running or accelerating multiple cryptographic algorithms, as well as generic and configurable cryptographic protocols or applications.

With strong type approval requirements from UNECE R155 and R156 [69] [70], it becomes even more evident to have hardware trust anchors that support the feature of crypto agility both for classical and post-quantum crypto algorithms in road vehicles.

5.2.10 OPEN-LOOP

At first glance, the processing chain within the ADI seems linear, often progressing from a perception layer via a sensor fusion layer to a planning layer. The performance of some of these layers can be improved by awareness of the actuator commands that will be executed in the next timestep. The perception and sensor fusion layers can benefit from anticipating how objects are likely to shift relative to the ego vehicle and the planning layer can better anticipate how other traffic participants will react.

We denote a system or subsystem that is in control of the vehicle as running “closed-loop”, i.e., its outputs directly affect the environment and therefore also the sensor inputs it will receive in the future. In contrast, a system or subsystem that is not in control of the actuators is running “open-loop”.

Conceptual system architectures that consist of multiple channels involve some mechanism for

switching between the currently active channel and a channel running in hot stand-by. Such a switch is not trivial: to allow for accurate execution, the "new" trajectory needs to continue smoothly from the "old" trajectory, i.e., first and second order derivatives (speed and acceleration) need to match, and the third order derivative (jerk) needs to be small. In practice, this may involve carrying over some trajectory points from the "old" trajectory. This poses additional constraints on the channel running open-loop in hot stand-by and slows down a potential emergency reaction. It should be noted that all AD systems need to be capable of running open-loop to a certain extent, e.g., to cover the time span between their activation and engagement.

Symmetric conceptual system architectures often allow rapid back-and-forth switching between different channels, i.e., for each situation the best-suited channel controls the vehicle. Asymmetric conceptual system architectures, on the other hand, are sometimes restricted to a one-way switch, i.e., only from a nominal channel to a fallback channel. Comfort is of secondary importance for an MRM, so the burden of running open-loop is less pronounced here.

5.2.11 COLD AND HOT STAND-BY

If a conceptual system architecture consists of more than one channel, all but the currently active channel are on stand-by (see also section 5.2.10). All architectures considered in this report use hot stand-by (see also section 3.2.1.3), where inactive channels continuously run open-loop. This ensures that there is no interruption of service if a change of the active channel occurs.

Theoretically, conceptual system architectures could also incorporate warm or cold stand-by. These come with the trade-off of reduced power consumption, but worse availability due to needing some time to become fully operational and being capable of performing the Dynamic Driving Task (DDT). For complex ECUs involved in the ADI, a cold start can take up to several seconds and not only includes HW initialization and boot-up, but also synchronization between multiple ECUs and establishing a converged environment model over several sampling frames. The latter would remain a potential problem even if such an ECU is already booted but idle; therefore, we do not consider warm or cold standby a viable solution in our context.

5.2.12 FORESEEABLE MISUSE

For SAE L2 systems the driver holds all responsibility for the vehicle. As this is part of the safety analysis and argumentation, suitable measures are necessary to prevent foreseeable misuse, e.g., the driver turning to other activities, falling asleep, or leaving the driver's seat. While for SAE L3 and higher systems the vehicle temporarily assumes responsibility for the Dynamic Driving Task (DDT), certain assumptions regarding the primary passenger remain important to the safety argumentation. For SAE L3 systems, the DDT Fallback is most prominent. However, there are also safety aspects not directly related to the DDT, e.g., ensuring orderly behavior of other passengers, a proper vehicle state (closed doors, nothing sticking out of windows, etc.), or resuming the drive from a minimal risk condition after the AD system fails and executes an MRM[53].

A Driver Monitoring System (DMS) can be used to ensure that the condition of the driver matches the underlying assumptions of the AD system, e.g., driver is not asleep, not incapacitated and in general ready to take over the vehicle if requested by the system to do so. If the DMS detects a violation of an assumption, it can at first try to remediate it by alerting the driver, and – if the driver still does not respond appropriately – trigger a system reaction, which may include the following:

- DMS alerts the Diagnostics System.

[53] An SAE L4 system does not rely on the primary passenger (driver) taking over quickly after a failure or ODD exit occurs. However, depending on the minimal risk condition it manages to enter after completing the MRM (e.g., pulling over to the side of the road or coming to a stop in-lane), it may require someone to resume driving, move the vehicle to a permanently safe location, or put up warning signs. Some of these activities may be incompatible with the driver falling asleep while the AD system is active.

- Diagnostics System requests one or more subsystems of the ADI to enter a degraded mode (e.g., pull over). This may involve a degraded mode that prevents continuation of the nominal function.

We assume that any system (even the Single-Channel architecture) will include functionality to prevent or react to foreseeable misuse, and hence see no impact of this functionality on the conceptual architecture.

5.3 STANDARDS FOR DEVELOPMENT OF SAFE AD SYSTEMS

5.3.1 APPLICABLE SAFETY STANDARDS

With respect to safety, the primary relevant standards are ISO 26262 (Functional Safety) [2] and ISO 21448 (Safety of the Intended Functionality) [3], which should be followed throughout the development of an AD system. Although compliance with those standards is not a formal (legal) requirement for vehicle homologation, they are considered "state of the art" (also in a legal sense) and therefore most OEMs adhere to them in their development processes and prescribe them to their suppliers.

Additional standards that are intended to be used to support the development of AD systems are ISO/TS 5083 ("Road Vehicles – Safety for automated driving systems – Design, verification and validation") [71] and UL 4600 ("Safety for the Evaluation of Autonomous Products") [22]. As AD systems very likely will contain elements of or even largely be based on artificial intelligence (AI), recently published standards like ISO TR 5469 ("Artificial intelligence – Functional safety and AI systems") [72] and ISO 8800 ("Road Vehicles — Safety and artificial intelligence") [73] should be considered too.

Finally, the UNECE Regulation No. 157 ("R157") [74] [75] is relevant for type approval of vehicles incorporating "Automated Lane Keeping Systems" (ALKS, essentially Highway Pilot-like functions) in a large number of countries and as such needs to be considered when developing AD systems to be deployed e.g., in the European Union.

In the following subsections, we summarize the architectural impact of each of those standards, i.e., how architectural considerations are either prescribed by the standards or how an appropriate architecture choice can help achieve standards compliance.

5.3.2 ISO 26262 (FUNCTIONAL SAFETY) CONSIDERATIONS FOR THE ADI IMPLEMENTATION

ASIL ASSIGNMENT

It is safe to assume that an AD system for an L4 Highway Pilot will get the ASIL D level assigned, as all three relevant factors will contribute and lead to this highest classification in terms of ISO 26262 [2]:

- Severity will be S3 (highest), as a malfunction of the AD system during autonomous operation can lead to a fatal crash.
- Exposure will be E4 (highest), as autonomous operation on a highway will be very common for a vehicle equipped with a Highway Pilot system.
- Controllability will be C3 (highest), as a malfunction while in autonomous operation will not be controllable by the passengers (would require an immediate attention shift and takeover by the "driver")[54].

[54] Note that ISO 26262 refers to "the driver or other persons involved in the operational situation" for classifying the Controllability. Strictly speaking, there is no "driver" for an L4 function, but we apply the definition analogously, because for a Highway Pilot we still assume a person present in the driver's seat, who needs to control the vehicle anyway until highway entry and after highway exit.

For a concrete system, of course, a Hazard Analysis and Risk Assessment (HARA) needs to be conducted to derive safety goals and associated ASILs, but without doubt will lead to the classifications above for many of a Highway Pilot's safety goals.

ASIL DECOMPOSITION

Note that the ASIL D assignment is valid for the AD item as a whole and may be lowered for some of its constituents by appropriate ASIL decomposition. This involves partitioning the system into sub-components with lower ASIL that jointly realize the safety goal and must be integrated using ASIL D-compliant technical measures and processes.
In the context of an L4 Highway Pilot, decomposition is also a practical necessity, as many components required to implement it are too complex to completely fulfill ASIL D criteria – like high-end SOCs, operating systems, or application software components[55].

AVAILABILITY AS A SAFETY GOAL

ISO 26262 was created with traditional powertrain, steering, braking, or even ADAS (L1 or L2) functions in mind. Such systems are usually developed with a correctness goal defined and a fail-silent system reaction, i.e., switch-off, in case of a malfunction.

For an L4 Highway Pilot with its fail-operational/fail-degraded requirement, however, the availability of the AD system becomes a safety goal with ASIL D too and needs to be met by applying appropriate technical and process measures – as any sudden non-availability of the AD system during L4 operation will not be controllable by the passengers and will likely cause harm (up to fatalities). Specifically, technical measures like fault detection and shutdown are not sufficient anymore and need to be replaced by component implementations that ensure fault avoidance and, potentially, fault tolerance; special attention is required for legacy components like communication stacks, which are based on error detection layers and keep the core functionality in QM safety level only. Formally this pattern will not comply with the availability safety goal pertinent to Level 4 systems.

REDUNDANCY

On the architecture level, the availability goal is usually addressed by appropriate redundancy measures, which are installed to cope with the unavoidable failure of individual components of the system (for example, due to permanent or transient electronic faults). All architectures evaluated in this report exhibit such redundancy, except for the Single-Channel architecture; note that mitigating residual SW errors requires diverse redundancy, which is provided naturally by asymmetric architectures.

SUFFICIENT INDEPENDENCE

Redundancy is not sufficient to ensure availability: Sufficient Independence of the redundant components must also be ensured, i.e., common cause and cascading failures avoided which would cause redundant components to fail jointly and render the whole AD system unavailable. The same requirement applies to intra-channel pairs of an intended functionality and a matching safety mechanism to ensure a channel's correctness. Furthermore, Sufficient Independence is a precondition for applying an ASIL decomposition: it is only allowed if the constituent architectural elements to which the initial requirements are allocated redundantly are independent of each other and do not dependently violate the functional safety goals.
In the case of an SAE L4 Highway Pilot, this means that redundant system channels to which decomposed requirements were allocated must be implemented in a sufficiently independent fashion, to rule out dependent failures that might impede either functional correctness (e.g., in

[55] Machine learning algorithms are often cited as intractable for development according to ISO 26262. Actually, their regular structure for the inference phase makes them quite easily compliant. Rather, it is the learning phase with its non-deterministic properties that stands in the way. The main problem to solve lies in the SOTIF domain, not in the functional safety.

a Doer-Checker configuration) or system availability (e.g., in a Doer-Fallback configuration). In addition to the well-known key question within the safety domain of all industries, "How safe is safe enough?", there now arises an additional challenge for AD systems: "How independent is independent enough?". Since the issue is of utmost importance not only for the achievement of Functional Safety but also SOTIF of an ADI, section 5.4 does a deep dive into the topic.

IS ASIL D ENOUGH?

ISO 26262 does not give any reference or consideration to the system's complexity or size – a SW component implemented in an ASIL D compliant process is considered to be safe, irrespective of the number of its lines of code (LOC). However, according to [10], a system with more than 10 kLOCs is likely to exhibit residual systematic SW errors, even when developed to the highest safety standards.

An L4 Highway Pilot can be considered a highly complex system and will certainly involve much more than 10 kLOCs for its implementation. Therefore, even the application of the ASIL D process to a complex system channel might not ensure system safety – instead, partitioning such a channel into smaller constituents with lower complexity, which are developed to ASIL D and can be readily (individually) verified, should be considered.

COMPLEX INTERACTIONS

All the investigated ADI architectures except the Single-Channel architecture exhibit some functionally and/or availability-motivated partitioning that reflects the need for redundancy and decomposition. Some are already initially quite complex (Daruma, Layer-Wise DCF, and DSM), but even the simpler ones (Majority Voting, CCP, Channel-Wise DCF, and AD-EYE) are likely to require further partitioning within their channels, to arrive at technically tractable implementations. Such partitioning will also support the ASIL D argument, as reflected above.

In any case, one might end up with a significant number of components with a (combined) potentially enormous number of system states and complex internal interaction. To prove that the integrated system possesses the expected safety and availability properties, and does not exhibit any unintended emergent behavior, formal modeling and verification can be employed. Such a process can give mathematical correctness assurances, where human cognitive limits are exceeded, and manual verification would be too error-prone.

5.3.3 ISO 21448 (SOTIF) CONSIDERATIONS FOR THE ADI IMPLEMENTATION

The standard SOTIF-ensuring process according to ISO 21448 [3] shall be followed when implementing the ADI, both on a system level and when implementing the channels and components of the architectures presented here.

Architecture considerations or dedicated architecture design and evaluation steps are not reflected in that standard; mentioned system modification steps to ensure SOTIF mostly focus on functional aspects and not architectural measures. Still, the chosen architecture will have a decisive impact on the efforts required to develop and validate an AD system.

However, certain architecture aspects are mentioned in ISO 21448, and closer examination shows that the architectures presented here quite naturally support the standard and help to reduce the effort required for ensuring SOTIF.

SENSE-PLAN-ACT

The established Sense-Plan-Act paradigm is used in ISO 21448 to motivate a modular specification, design, and V&V approach, with individual qualitative and quantitative development goals for each layer. In the architectures described in this report, the analogous layering is exp-

licitly foreseen only in the Layer-Wise DCF architecture. However, the other architectures can also be broken down into a similar structure for their top-level building blocks. This applies to the channels of the Single-Channel, Majority Voting, CCP, Daruma, Channel-Wise DCF, and AD-EYE architectures, as well as the FUN layer of the DSM architecture. It should be noted that such a break-down represents the classical ("white box") approach to systems design and runs counter to end-to-end AI ("black box") approaches.

DDT FALLBACK

The "DDT fallback" is assumed as a functional entity (not necessarily an architecture element) in ISO 21448. This element is present in all the architectures discussed here, except the single-channel architecture, in an explicit manner (Channel-Wise DCF, Layer-Wise DCF, DSM) and implicitly through the multiple channels of the Majority Voting and Daruma architectures.

ARCHITECTURE IMPACT ON V&V

ISO 21448 acknowledges that a suitable architecture can support more efficient verification and validation by enabling a modular approach and reducing the effort for V&V of individual components (compare also G3: Need for safety by design and D1: Fault Containment Units). Diversity and independence arguments (compare also D3: Diversity and redundancy for complex subsystems) are used, albeit in example form only and not as an integral part of the process to achieve SOTIF.

5.3.4 UNECE R157 CONSIDERATIONS FOR THE ADI IMPLEMENTATION

UNECE Regulation No. 157 [74] applies to passenger cars and busses incorporating Automated Lane Keeping Systems (ALKS) originally up to a maximum speed of 60 km/h but has been extended in 2023 to include lane change maneuvers, speeds up to 130 km/h, and light trucks. The target use case of an ALKS, which *"controls the lateral and longitudinal movement of the vehicle for extended periods without further driver command"*, includes our definition of a Highway Pilot, by requiring that the system *"shall perform the driving task instead of the driver, i.e., manage all situations including failures"* and by describing the intended ODD as *"pedestrians and cyclists are prohibited … equipped with a physical separation that divides the traffic moving in opposite directions and prevent traffic from cutting across the path of the vehicle"*.

With respect to autonomy level, UNECE R157 does not follow the taxonomy of SAE J3016, but requires a "transition demand" to the driver in case of severe ALKS failures that would lead to violation of R157 requirements, and an MRM capability (which is obligatory for Level 4, but may be limited for Level 3 according to SAE J3016) in case the driver fails to respond to the takeover request within 10 seconds. However, a "minimum risk maneuver may be initiated immediately in case of a severe ALKS failure" while "it shall aim at enabling a safe transition of control back to the driver". UNECE R157 prescribes a combination of driver and system reactions to unplanned events like system failures, and it is up to the OEM to define the concrete escalation steps. In contrast, SAE J3016 distinguishes between Level 3 and 4 with different system responsibilities.

The regulation is relevant for vehicle type approval in the countries that adhere to "World Forum for Harmonization of Vehicle Regulations (WP.29)" regulations (primarily in the European Union but also including a number of non-European countries). Manufacturers are required to demonstrate compliance to the Type Approval Authority or a nominated "Technical Service" acting on behalf of it. In its subchapters, R157 defines requirements for the following functionalities in considerable detail (we cite the chapter titles and list the main content):

- *"System Safety and Fail-safe Response"*: normal operation (DDT, Lane Change Procedures LCP), fault reactions (MRM) and evasive maneuvers
- *"Human Machine Interface/operator information"*: driver monitoring, system activation,

warnings, takeover procedure

- *"Object and Event Detection and Response (OEDR)"*: sensing capabilities, detection ranges, environmental conditions
- *"Data Storage System for Automated Driving (DSSAD)"*: record content (triggers, data elements), availability, retrieval procedure
- *"Cyber Security and Software Updates"*: mostly references to adjacent regulations UNECE R155 and R156

Extensive formal documentation needs to be submitted to the Type Approval Authority, describing the above functionalities and including a system description, its components and architecture, the safety concept, redundancy mechanisms, degradation strategy, HMI, DSSAD, the cyber security and update schemes etc. A development process in compliance with ISO 26262 and 21448 will presumably already create significant parts of the necessary documentation and evidence for R157.

Regarding the functional behavior, R157 not surprisingly requires the system to essentially drive safely, obey the traffic rules, show anticipatory behavior, avoid collisions, and perform an MRM "after a transition demand without driver response or in the case of a severe ALKS or vehicle failure" (i.e., to perform self-checks to detect failures and to confirm its performance continuously). This is specified in a detailed list of requirements that may be sufficient to achieve safety, but that reflects the open nature of the problem by being sometimes formulated in a rather general way, e.g.,

- *"The activated system shall adapt the vehicle speed to infrastructural and environmental conditions (e.g., narrow curve radii, inclement weather)."*, or
- *"The ALKS shall ensure sufficient lateral and longitudinal distance to road boundaries, vehicles and other road users."*

In addition, an annex is devoted to the three scenarios of cut-in, cut-out and deceleration of "other vehicles". Those are deemed particularly critical and described in detail using kinematic parameters and conditions that shall be kept in order to avoid collisions.
Fulfillment of the general functional and safety requirements, as well as under the three specific scenarios shall be demonstrated and will be tested by the authorities, both on test tracks (including fault injection) and on public roads.

When setting up an ADI development and seeking compliance with R157 to successfully pass type approval, two key questions arise that naturally point to the architecture focus of this report:

1. How to ensure fulfillment of the requirements in all real-world scenarios, i.e., including unspecified scenarios (edge cases), ODD violations and arbitrary faults?
2. How to ensure fulfillment of the requirements systematically and not just exemplarily (i.e., not just under a limited set of test cases that might have been selected at random, for example, just because they can be carried out in a reasonable time)?

Clearly, in addition to taking over and tracing to the concrete functional requirements of R157 throughout the ADI development process, a well-designed architecture can systematically support fulfilment of the safety requirements of R157:

- Provide independent monitoring functions and diagnostic capabilities, to ensure adherence to the safety requirements set forth by R157 in an explicit and systematic way; provide an independent MRM capability under arbitrary faults[56] by appropriate redundancy.

[56] For non-severe faults, the system needs to continue for at least 10 seconds before initiating an MRM.

- This approach can enable R157 compliance "by design" and continuously during the development phase – as opposed to just relying on selected test cases at the end of the development phase and at type approval, which are necessarily late, of limited coverage, and do not ensure full compliance.
- An example of this would be to complement an AI-based system with heterogeneous (rule-based) monitoring functions, and appropriate redundancy that provides an independent fallback function in case of faults or driver non-availability (for example, if a planned ODD exit is approaching and the driver is not responding to a takeover request).

Under this viewpoint, several of the architectures investigated in this report support R157 compliance in a straightforward way; such architectures contain monitoring elements which enable adherence to the safety requirements. Also, they foresee elements that can take over and execute an MRM in case of an imminent violation of those rules:

- Cross-Checking Pair: Validators in each channel, redundant channel configuration with Selectors
- Daruma: Daruma validator, redundant channel configuration with Selectors
- Channel-Wise DCF: M-System, F-System
- Layer-Wise DCF: Primary Planner/Executor, Safety Gates, Safing Planner/Executor
- DSM: Sensor and Function Monitor (SFM), Controller (CSM), and Vehicle (VSM) Safety Mechanism
- AD-EYE: Channel 2a and Channel 2b

The Single-Channel and the Majority Voting architectures are not well suited to support R157 compliance, as they require implementing the safety requirements directly in their functional channels. In those architectures, safety rules might be implemented only implicitly and exemplary (e.g., by inclusion of relevant training scenarios in the case of AI-based functions) and accordingly can only be verified through dedicated test phases. Evidence of compliance is then sought by probing the system and not also ensured by design.

5.3.5 ISO/IEC TR 5469 CONSIDERATIONS FOR THE ADI IMPLEMENTATION

ISO/IEC TR 5469 "Artificial Intelligence - Functional safety and AI systems" [72] has been published in its first edition in January 2024. It is designated as a Technical Report (TR) and as such not normative, but informative and can be understood as presenting the perceived "state-of-the-art" about usage of AI in the context of safety-related systems. It is meant as a generic TR, i.e., not confined to road vehicles, although many references and examples point to autonomous driving as a key application area.

It is worth noting that throughout TR 5469, the term "Functional Safety" is used quite broadly and does not differentiate between E/E system malfunctions (ISO 26262) and violations of SOTIF (ISO 21448) – probably stemming from the heavily cited IEC 61508, which as a generic, non-automotive standard does not differentiate either. In fact, most fault scenarios described in TR 5469 are related to SOTIF and not to Functional Safety (per the definition from ISO 26262 and used in this report) and accordingly the measures proposed by TR 5469 are meant to address both sources of risk.

TR 5469 identifies four Usage Levels (A through D), that define the applicability of the report's clauses:

a. Use of AI inside a safety-related function to realize the functionality
b. Use of AI systems to design and develop safety-related functionality
c. Use of AI systems, but have only an "indirect" impact on safety

d. Use of AI system, but no impact on safety

TR 5469 also mentions a broad spectrum of "AI" technologies – from simple linear functions to artificial neural networks (ANNs) – and defines three *Technology Classes (I through III)* for these:

i. AI fully developed according to an existing Functional Safety standard
ii. AI developed with "complementary" risk reduction methods
iii. AI with no risk reduction

Of these, we consider usage level A and technology classes II/III to be the relevant classifications in the context of this report, as AI is mostly used in perception/decision-making tasks and is usually implemented as deep neural networks (DNNs), which are unlikely to be realizable according to an existing safety standard – in particular, considering the SOTIF impact of the learning approach typical of DNNs, as opposed to a traditional development.

Based on the above classification, TR 5469 promotes a systematic approach to achieve risk reduction, by structuring the development effort in 3 stages of AI implementation:

1. Data acquisition
2. Knowledge induction (i.e., training)
3. Output generation (i.e., inference)

For each stage, desirable properties, requirements and KPIs shall be defined and verified during development. To this end, TR 5469 also provides a description of typical risk factors of AI systems, such as lack of transparency, explainability and predictability, vagueness of specification, data and concept drift, and sensitivity to adversarial or intentional malicious inputs; potential measures and considerations are described like orienting training data on HARA, robust learning, attention mechanisms, compression technologies, statistical analysis, appropriate metrics, testing, simulation and field observation, and an extensive list of supporting technical literature is provided.

However, though it seems likely that such measures can contribute to improved reliability of the AI function, given the current state of the art they will not attain the same level of objective safety evidence as traditional development methods. Probably acknowledging these limitations, and considering that improving AI quality alone is not sufficient to ensure robustness and fault tolerance, TR 5469 in addition to the described systematic development approach proposes architectural considerations and design patterns. For illustrative purposes, Figure 57 shows all of these at once, though it should be noted that TR 5469 states that a single one may be sufficient and does not demand nor explicitly recommend a combination of multiple architectural measures.

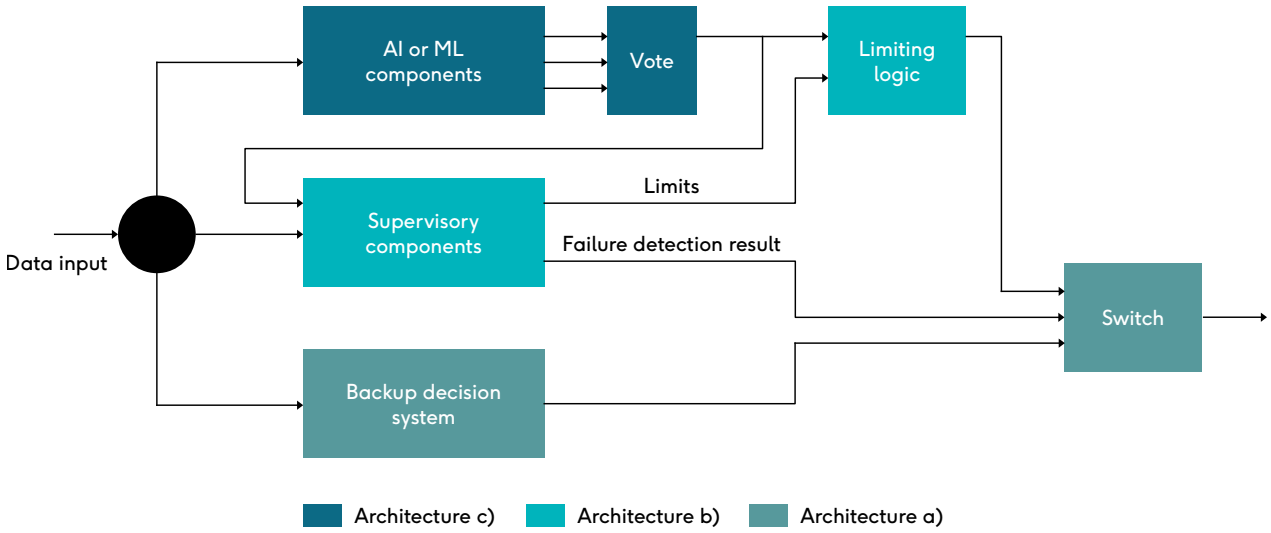


Figure 57: Architectural patterns proposed by ISO/IEC TR 5469.

- Use a set of "AI or ML components" augmented by a voting logic to provide the nominal, AI-based functionality of the overall system. This seems suitable for classification tasks, but not for planning tasks, where different but equally correct solutions may exist (see this report's discussion of the voting patterns).
- Add a Supervisor with diverse (potentially non-AI) implementation, evaluating the output of an AI component, followed by a limiter to confine the AI component output to a valid range; the latter may not be suitable or necessary for classification tasks, where the AI component is designed to just output a set of admissible values anyway.
- Add a Backup System with diverse (non-AI) implementation and a decision element (switch) to choose between the main function and the backup function.

While the primary focus of the architectural measures described in TR 5469 is to ensure the functional safety of individual AI-based components, the same architectural measures can be scaled and applied to the conceptual system architecture of the ADI. A combination of AI Component, Supervisor, and Backup System resembles what we described as the Channel-Wise DCF architecture; Voting on several AI instances and Limiting Logic might be used alternatively or in addition to enhance system reliability and robustness.

To summarize, ISO/IEC TR 5469 proposes two complementary workstreams for the development of safe systems based on AI:

[1] Structuring the development of AI as a 3-stage approach (data acquisition, learning, inference) and systematically defining and tracking desirable properties, requirements, and KPIs for each stage – this will mainly improve the availability of the nominal functionality (as defined in this report).

[2] An architecture approach including non-AI elements to ensure system availability and safety. The latter confirms the "architecture-first" approach taken by this report and its positive evaluation of the architectures containing diverse supervision and validation components (Cross-Checking Pair, Daruma, C-DCF, L-DCF, DSM, and AD-EYE). For the symmetric architectures (Cross-Checking Pair and Daruma) it would need to be ensured, of course, that non-identical AI elements are deployed in redundant channels.

5.3.6 ISO/PAS 8800 CONSIDERATIONS FOR THE ADI IMPLEMENTATION

ISO/CD PAS 8800 "Road Vehicles - Safety and Artificial Intelligence" [73] has been published in December 2024. This standard addresses the safety of AI systems in road vehicles, and it tailors or extends existing approaches currently defined within the ISO 26262 series (functional safety) and ISO 21448 (safety of the intended functionality). For instance:

- The ISO/PAS 8800 standard considers and adapts to road vehicles the general framework described in ISO/IEC TR 5469 on safety properties, virtual testing and physical testing, confidence in use of AI development frameworks and architectural redundancy patterns.
- AI-specific definitions listed in ISO/PAS 8800 are used from ISO/IEC 22989 [76], unless in conflict with safety-specific definitions. So, safety-related properties are a subset of generic AI properties described in ISO/IEC 22989.
- On the other hand, Annex B of ISO/TS 5083 applies the guidelines and principles outlined in ISO/PAS 8800 to the context of Automated Driving Systems.

5.3.6.1 ARCHITECTURAL MEASURES FOR AI SYSTEMS

This section will explore various architectural measures proposed by ISO/PAS 8800 to enhance critical AI properties such as robustness, resilience, reliability, and predictability, etc. Architectural measures are key design elements that help AI systems to maintain optimal performance even under challenging conditions. These measures aim to support safe, efficient, and predictable operation of the AI system. As with ISO/IEC TR 5469, architectural measures can be applied to individual AI-based components or scaled up to the conceptual system architecture of the ADI.

5.3.6.1.1 SAFETY-RELATED PROPERTIES

Architectural measures are the structural elements of an AI system that support and maintain the safety-related properties essential for its safe operation. ISO/PAS 8800 outlines and lists these safety-related properties, which typically include:

- **AI robustness**: the ability to maintain an acceptable level of performance under the presence of semantically insignificant but reasonably expected changes to the input.
- **AI reliability**: the ability of the AI element to perform the AI task without AI error under a range of operational conditions and for a specified period of time.
- **AI predictability**: the ability of the AI system to produce trusted predictions. This implies reliable confidence information, i.e., the ability of an AI model to reliably indicate if its prediction can be trusted or not. This is not always true for all kinds of models. For example, the output of a SoftMax function is frequently misinterpreted as some kind of posterior distribution which indicates confidence.
- **AI controllability**: the ability of an external agent to overwrite the behavior or output of an AI system.
- **AI explainability**: the ability to explain in natural language which factors influence the AI element decision and how.
- **AI resilience**: the ability of the AI element to recover and continue performing the AI task after the occurrence of an AI error.
- **AI generalization capability**: the ability of a model to adapt and perform well on the previously unseen data during inference.

5.3.6.1.2 MEASURES FOR ARCHITECTURAL REDUNDANCY

In Annex G1, ISO/PAS 8800 adopts the same architectural patterns and elements proposed by ISO/IEC TR 5469, which describes some architectural redundancy patterns for systems using AI technology components, as illustrated in Figure 57. Figure 58 translates them in the context of the reference architecture for an AI system.

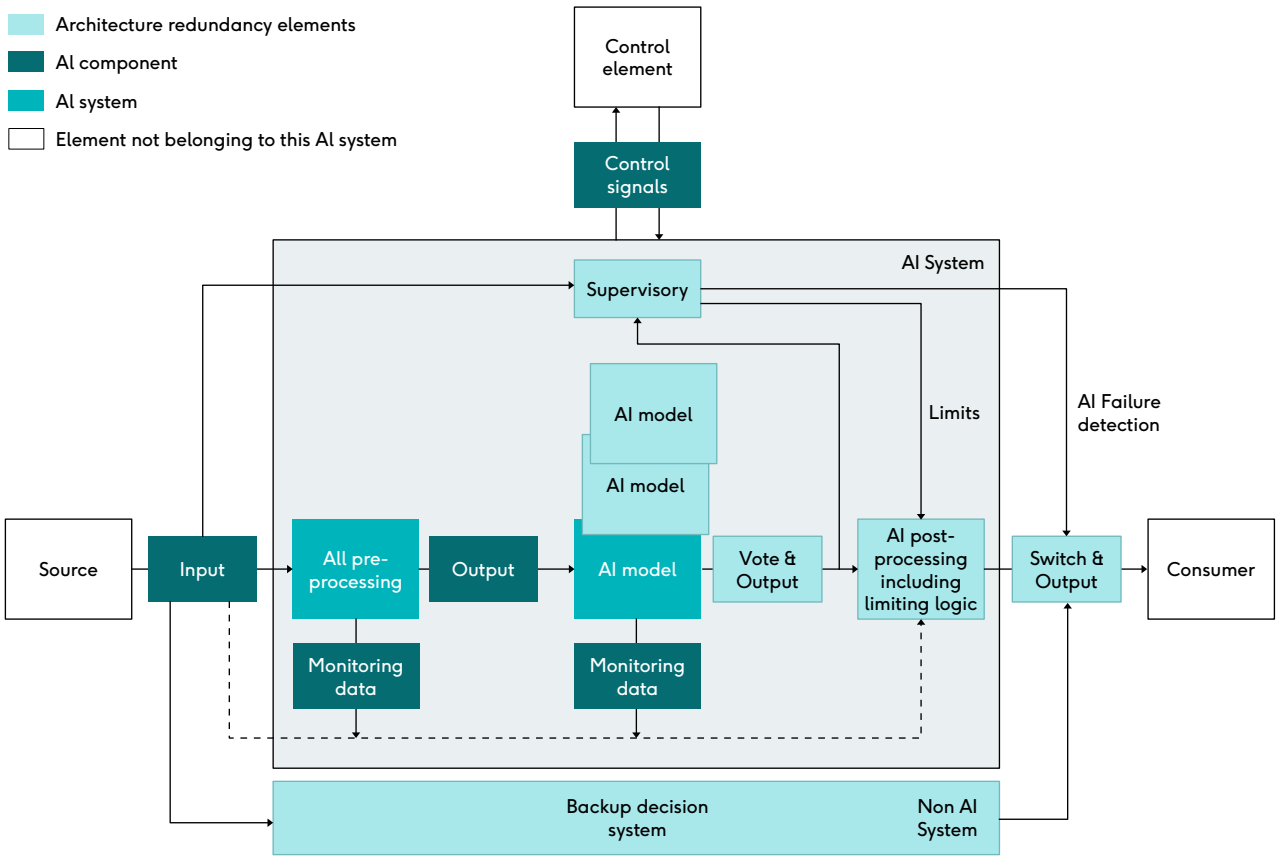


Figure 58: Architectural redundancy patterns for an AI system, from ISO/PAS 8800.

Architectural redundancy is designed to enhance the AI properties listed above, detect and mitigate failures of the AI model, maintain the system operation in the event of a failure, and ensure a safe system failure. Various architectural redundancy patterns can be employed to detect and react to AI errors:

Diverse redundant models: Redundancy can be achieved by voting between diverse models. It involves combining multiple AI technologies fulfilling the same functionality, but implemented starting from different problem formulations, using different training data or different models. Model ensembles: combine the predictions obtained by multiple different models to create a consolidated output. The multiple models can be of different architecture and have different hyperparameters or of similar architecture but trained on different data sets.

N-version diverse programming: Multiple independent versions of an AI model that are built to predict the same output when the same input is provided. Independence and diversity are attained via using different training data, different AI model architectures or a different training process.

Supervisory, limiting logic and non-AI backup system:

- Supervisory system: A monitoring layer (safe envelope) that oversees an AI component's decisions and actions.
- Limiting Logic: Predefined rules that restrict AI behavior to prevent dangerous or unintended outputs.
- Non-AI Backup System: A fallback system that does not rely on AI.

The Channel-Wise DCF architecture explicitly supports the use of a supervisory layer and non-

AI backup systems by organizing them into distinct functional channels. So, ISO/PAS 8800 recognizes it as a suitable method for addressing the safety challenges of AI-based systems. Other architectures described in this report with diverse architectures do not match the ISO/PAS 8800 proposal directly, but provide other means to achieve similar effects.

Selection techniques for architectural redundancy (voting and switching):

- Voting-based decision procedure for architectural redundancy patterns:
  - Hard voting (majority voting): every individual classifier votes for a class, and the majority wins.
  - Soft voting: every individual classifier provides a probability that a specific data point belongs to a particular target class. The predictions are weighted by the classifier's importance. Then the target label with the greatest sum of weighted probabilities wins the vote.
- Switching-based approach: the selection of the base predictor model is made based on predefined rules rather than voting:
  - Threshold switching: the model whose predictions satisfy certain conditions, or a set of performance thresholds is selected.
  - Performance switching: the model with the best recent performance is selected.

Usage of "AI model" and "conventional software": Components using conventional software can be used to perform the plausibility checks or verification of the output generated by the AI components.

5.3.6.1.3 QUALITATIVE AND QUANTITATIVE ANALYSIS OF AI ARCHITECTURES

ISO/PAS 8800 proposes qualitative and quantitative analysis activities to ensure the safety of the AI architecture; qualitative analysis focuses on structural and functional aspects of AI architecture, whereas quantitative analysis evaluates the measurable performance of AI architecture.
AI work products can be analyzed for systematic faults through the following methods:

- Peer Reviews to assess hyperparameter selection and optimization.
- Requirement-based verification to check whether the AI architecture meets the AI safety requirements, or if individual elements violate AI safety requirements allocated to them.
- Failure analysis at interpretable system interfaces to identify and mitigate potential failures by integrating appropriate monitoring mechanisms.

Architecture analysis, aiming to identify potential AI-related errors, will assess:

- Tolerance: evaluating system resilience against incorrect inputs, computation errors, and adversarial attempts.
- Adaptability: ensuring computational behavior remains stable before, during, and after adaptation[57].
- Information flow of the proposed architectural concept: reviewing how data moves between subsystems and interacting components to ensure transparency and maintainability.
- Safety-related properties of AI systems, including robustness, reliability, predictability, and resilience, can be analyzed using both quantitative and qualitative architectural measures.

5.3.6.2 ISO/PAS 8800 STANDARD CONCLUSIONS

The ISO/PAS 8800 standard addresses relevant architectural patterns that guide architectural choices and help achieve compliance with its guidelines. For example, voting-based classifi-

[57] In the context of AI, adaptation refers to modifying the AI system's behavior, e.g., by training with more data or changing the algorithms.

cation or decision procedures are useful for redundancy patterns, although this report does not endorse the full system-level Majority Voting architecture.

The use of modular architectures, such as the Channel- or Layer-Wise DCF, DSM, or AD-EYE architectures, is another way to comply with ISO/PAS 8800. These layered designs aim to enhance system safety and help maintain functionality during faults. If their fallback mechanisms are implemented in non-AI means – as suggested by ISO/PAS 8800 – such architectures ensure continued operation if AI-driven decisions fail.

5.3.7 UL 4600 CONSIDERATIONS FOR THE ADI IMPLEMENTATION

The "UL Standard for Safety for Evaluation of Autonomous Products, UL 4600" [22] was first published in 2020 and in its third edition in March 2023. It aims at defining a comprehensive safety case for autonomous vehicles both in urban and highway use cases, and a structured set of safety claims, arguments and evidence shall be collected.

Extensive chapters and clauses cover a definition of the safety case structure and content, a systematic risk assessment, human interaction aspects, functional and architecture aspects, dependability claims, data storage and networking topics, verification and validation, tools considerations (including COTS and legacy components), product lifecycle and maintenance, safety performance indicators (SPIs) and finally conformance assessment and monitoring.

UL 4600 is not meant as a process standard and does not prescribe a specific sequence of development steps but instead focuses on the artifacts that shall be created to constitute the safety case; it is designed to be compatible with ISO 26262 and ISO 21448, concretizing and detailing those standards for the use case of autonomous functions.

Compliance to UL 4600 is not legally required anywhere but would show an OEM's commitment to safety. Also, UL 4600 contributes to the "state of the art", and as such would constitute a relevant safety baseline in case of legal (warranty) claims.

With respect to the architectural focus of this report, UL 4600 mandates a structured ADI architecture approach both on the logical and physical level, but (as expected from a standard) no specific architecture is prescribed. Still, architectural considerations are implicitly or explicitly requested in many chapters and clauses:

- Established architectural concepts like Fault Containment Regions (FCR) and Elements-out-of-Context (EOOC) are used.
- The Doer-Checker pattern is referenced frequently as a viable architectural option and even singled out in the standard's "Terms" section; defective construction patterns shall be avoided ("command-override" is mentioned as an example).
- UL 4600 requirements to identify and mitigate faults can realistically only be met with a suitable architecture that contains monitoring and fallback elements.
- UL 4600's chapter "Human Interaction" lists extensive requirements to deal with emergency horns, manual directions by police, occupant entry and exit, ensuring safety of operational staff, misuse incl. malicious misuse etc.; asymmetric (diverse) architectures might be better suited to handle this complexity, e.g., even if detection of an emergency vehicle or police intervention would fail in a main system, an orthogonal "don't hit anything" policy performed by the fallback could mitigate the risk. This approach would address the requirements in a general way, by virtue of system design and architecture, instead of directly specifying and addressing case after case (with potential misses caused by the long tail of edge cases).
- Chapter "Autonomy functions and support" explicitly requires redundancy. Many functional

requirements of this chapter can realistically only be met with an architectural approach that supports them – e.g., supervisor components to detect ODD violations.

- Sufficient Independence, i.e., avoidance of common cause faults of redundant channels is stated as required.
- Observability of perception functions is mandated "intentionally", and it is mentioned that this clause might rule out end-to-end AI approaches.
- Certain architecture elements, like monitors/supervisors and redundant channels, are implicitly required by many clauses, e.g., to mitigate faults, mitigate ODD violations, or to collect performance metrics and track SPIs (Safety Performance Indicators) mandated by UL 4600. For instance, with reference to UL 4600 a framework for tracking SPIs and detecting hazards is proposed in [77], and applied to the Daruma architecture along with example cross-channel SPIs and a proof-of-concept implementation.
- Architectural partitioning is encouraged to "effectively limit the scope of changes to be analyzed" in case of implementation changes.

For the design of an ADI architecture, the following key requirements can be identified when UL 4600 compliance is sought:

- A logical and a corresponding physical system architecture needs to be designed. This confirms the "architecture first" approach of this report.
- Redundancy is mandated, with the "design intent for redundancy" to be specified – reflecting additional clauses that require a fault model and corresponding mitigation measures that shall be defined. For "life critical risks", faults shall be mitigated that affect a single FCR; use of multiple FCRs i.e., redundancy is a logical and recommended measure.
- A layered channel architecture is recommended implicitly, as observability of the perception results is required. This effectively rules out monolithic end-to-end AI approaches, which likely do not provide such observability. However, segmented end-to-end approaches with separate perception/prediction and planning/trajectory layers would provide observability and would be admissible.
- Supervision elements are required for multiple reasons – e.g., to identify faults, collect metrics, and track SPIs.
- Sufficient independence to avoid common cause failures of redundant elements is to be ensured, implying a certain level of diversity as a high-integrity implementation of complex channels seems to be unrealistic. This preference of asymmetric architectures with heterogeneous FCRs is also supported by multiple references to the Doer-Checker pattern.

From the architectures investigated in this report, the Single-Channel architecture, and particularly an end-to-end AI-based implementation thereof, would not be compliant to UL 4600 for lack of redundancy, supervision, and observability (the latter in the case of a monolithic AI network).

The Majority Voting architecture provides a better basis to achieve UL 4600 compliance, but without further channel-internal measures also lacks observability and supervision elements and is sensitive to common cause faults.

The Cross-Checking Pair and the Daruma architectures improve on this by providing all capabilities that UL 4600 asks for: redundancy, observability and supervision, with Daruma appearing to provide a more differentiated evaluation and handling of faults. However, a "full computer redundancy" as suggested by these approaches is explicitly not mandated by UL 4600 (not to speak of the triple redundancy required for Majority Voting) and asymmetric Doer/Checker approaches are encouraged instead. To avoid common cause failures a heterogeneous implementation of their channels will also be needed.

The Channel-Wise DCF architecture provides a sound basis for achieving UL 4600 compliance but would need a layering of its L2-System into separate perception (world model) and policy (trajectory planning) components to achieve the observability target for perception, and likely additional internal supervision capabilities for tracking of metrics and SPIs. The same applies to the DSM architecture with its FUN channel and to the AD-EYE architecture. In both architectures, employing an end-to-end AI in the Doer or FUN channels would formally not be compliant, if such AI would not offer the internal observation and supervision capabilities.

Finally, the Layer-Wise DCF architecture with its multiple layers of Doer/Checker configurations is a very good basis to achieve UL 4600 compliance, as besides the foreseen redundancy, the observation and supervision criteria are also already met on the conceptual level, and the asymmetric approach helps ensure immunity against common cause failures – as with the DCF architectures.

To summarize, UL 4600 endorses the architecture focus of this report, and several of the architectures discussed here provide a sound basis to achieve compliance with the standard. Besides the direct architectural impact of UL 4600 (redundancy, observability, supervision capabilities) there are many additional clauses which can be supported indirectly by a suitable ADI architecture.

5.3.8 ISO/TS 5083 CONSIDERATIONS FOR THE ADI IMPLEMENTATION

ISO/TS 5083 "Road vehicles – Safety for automated driving systems – Design, verification and validation" [71] was published in 2025. It derives from the whitepaper "Safety First for Automated Driving" (SaFAD) [4] published by a group of OEMs and Tier-1s in 2019. This technical specification gives guidance on how to develop and validate an AD system for road vehicles (excluding motorbikes) and how to structure the safety case for an AD system. It focuses on design, verification, and validation, but also outlines cybersecurity considerations.

ISO/TS 5083 includes normative requirements regarding safety strategy, safety by design, validation, and operation of an ADS-equipped vehicle. Throughout these sections, the document refers to capabilities that the AD system must provide, among these also a "DDT fallback" for the execution of MRMs. While these are required to be provided by the AD system, the document does not suggest how this should be accomplished, i.e., it intentionally leaves open what system architecture should be employed. This is intended to allow for different architectural solutions to match the specific AD use case and associated ODD.

ISO TS 5083 is architecture-agnostic and does not include any normative requirements regarding the conceptual system architecture. It only has an indirect impact on the architecture and achieving compliance is similar for all architecture candidates. Only in its Annex B on the use of AI models within the AD system are architectural measures mentioned, concretely the AI/Supervisor/non-AI Backup pattern suggested by ISO/PAS 8800, which may be employed either for dedicated AI subsystems or (equivalent to the Channel-Wise DCF architecture) on an ADI system level.

5.3.9 PROPOSED SUPPORTING STEPS FOR THE ADI IMPLEMENTATION

Reflecting the safety considerations from the previous sections, the following four development steps are suggested for the implementation of the architectures of an SAE L4 Highway Pilot described in this report:

STEP #1: SYSTEM PARTITIONING AND MAPPING

Partitioning the system into subsystems (see also D1: *Fault Containment Units*) that jointly implement the L4 Highway Pilot functionality is usually done as one integral step, to meet the system correctness goals, the availability goals, and ensure feasibility of the ASIL D requirements. In fact, the presented architectures (except the Single-Channel architecture) largely anticipate this step, introducing redundancy to ensure system availability and checking instances to ensure system correctness (integrity).

Development step	Goal	Description
System Partitioning	System correctness	Ensure correct system outputs (e.g., trajectories) under the environmental and vehicle conditions: • Voter in Majority Voting architecture • Validation and Selectors in Cross-Checking Pair architecture • Daruma evaluator + Selector in Daruma architecture • Doer-Checker configuration with Decider in Channel-Wise DCF • Doer-Checker configuration with Safety Gates in Layer-Wise DCF • FUN/SFM/CSM in the DSM architecture • Channel 2a in AD-EYE
	System availability	Add redundancy to ensure availability of the system under component failures: • Channels in Majority Voting, Cross-Checking Pair, and Daruma architectures • Doer-Fallback configuration in Layer- and Channel-Wise DCF and AD-EYE architectures • VSM and Primary/Secondary Networks in DSM architecture • Fault-tolerant (redundant) implementation of Voter (Majority Voting), Decider and Safing Gates (DCF, AD-EYE), Selectors (Cross-Checking Pair, Daruma), Daruma evaluator.
	ASIL D feasibility	Further partition (modularize) the channels to enable ASIL D capable subsystems under HW and SW component constraints. This is usually not visible on the conceptual architecture level, but a necessary practical step in all architectures. Examples are: • Separate perception components per sensor • Low-level vs. high-level fusion • Trajectory planning and validation • Voting and decision components

The partitioned system architectures also serve as a basis for the formal ASIL decomposition, which results in assignment of ASILs to the individual architecture elements. We anticipate the following assignments[58]:

[58] Formally, the ASIL assignment and decomposition needs to be carried out separately per Safety Goal, in our problem space usually for the *correctness* and the *availability* goals. If an element receives differing ASILs in this process, the stricter one applies.

Archi-tecture	ASIL	Components
Single-Channel	D	Whole architecture
Majority Voting (2oo3)	B(D)[59]	Channels
	D	Voter (also fail-operational)
Cross-Checking Pair	B(D)	Channels including Validators
	D	Dual Selectors
Daruma	B(D)	Channels
	D	Daruma validator, dual Selectors
Channel-Wise DCF	B(D)	L2-System, M-System, F-System
	D	D-System (also fail-operational)
Layer-Wise DCF	B(D)	Primary/Safing Planners, Primary/Safing Planner Safing Gates, Primary/Safing Trajectory Executor, Primary/Safing Trajectory Executor Gates (some integration functionality between these nodes may be ASIL D, but is not explicitly visible)
	D	Priority Selector, Vehicle Control (also fail-operational)
DSM	B(D)	FUN (Function), SFM (Sensor and Function Monitor), Primary Network, Secondary Network.
	D	CSM (Controller Safety Mechanism), VSM (Vehicle Safety Mechanism)
AD-EYE	B(D)	Channels
	D	Dual Selectors

[59] Instead of the B(D) + B(D) decompositions to achieve ASIL D, other variants are also possible wherever multiple components work together: A(D) + C(D) or QM(D) + D.

After the (logical) system partitioning and ASIL decomposition, the mapping of the components to available HW modules (SOCs) and SW partitions/operating systems needs to be performed, including selection of appropriate communication means between the components. This mapping will be largely guided by the functional demands (e.g., computing performance requirements of each logical component, network bandwidth required) and the available functional safety support by SOCs and the SW platform; it may be an iterative process until an optimum solution is found.

For the sake of generality, this report does not consider specific SOCs and SW platforms and therefore does not dive into the mapping task further.

STEP #2: FORMAL MODELING AND VERIFICATION

After the partitioning and mapping has been performed, a complex architecture with a significant number of components and interaction may be the result. To validate the result, formal methods can be used to deal with the potentially large, and often cognitively intractable number of system states and ensure the desired system properties with a mathematical proof.

Development step	Goal	Description
Formal Modeling and Validation	Logical consistency, correctness, and system availability	Create formal system model and formal description of desired properties, and simulate on logical (conceptual architecture) level: • Proof of desired properties • Absence of violations of such properties (e.g., absence of single points of failure) • Avoidance of unintended emergent behavior
	Logical - physical consistency, system availability	Add formal modeling of the mapping of the logical building blocks to physical components and simulate on physical (implemented architecture) level: • Preservation of desired properties • Absence of violations of such properties (e.g., absence of single points of failure under the physical mapping

STEP #3: SUFFICIENT INDEPENDENCE ANALYSIS

Design principle D7: Mitigation of common cause hazards purposefully prescribes introducing diversity on the architecture and implementation level. Nevertheless, in realistic implementation scenarios, the redundant channels of the conceptual system architectures might be implemented on homogeneous platforms (e.g., SOCs and OSs) and communication technologies. Also, the implementation of applications like AD algorithms might share a set of mathematical libraries, HW accelerator layers, etc. This might even be extended to joint perception and fusion components used by the different channels.

To achieve a sound safety argumentation, a systematic and detailed analysis of "sufficient independence" (Dependent Failure Analysis, DFA) will need to be performed, to identify plausible potentials for common cause and cascading failures and to resolve them. Note that such an analysis will be necessary even if the components of each channel are sourced from different suppliers or implemented by different teams, as they might use identical third-party components, derive from the same architecture specification (e.g., AUTOSAR), or use identical legacy IP blocks even within heterogeneous SOCs. For further details see section 5.4.

Develop-ment step	Goal	Description
Sufficient indepen-dence analysis	Ensure system availability and correctness	Systematically analyze HW and SW implementation (platform and application, including communication network), considering the conceptual interaction of the architecture's building blocks: • Prove absence of plausible potential for common cause faults in redundant elements, on platform and application level • Prove absence of plausible potential for cascading faults across the ADI architecture

STEP #4: MARKOV ANALYSIS

The overall failure rate needs to be determined and proven to be within a set target, usually in the order of magnitude of ASIL D metrics (10-8/h) (compare section 2.2.7). For complex, fault-tolerant architectures with redundant paths like the ones described in this report, one cannot simply add up the failure rates of the constituents but must consider their interaction. This amounts to describing the overall system's states (like normal operation, degradation etc.) and their transition probabilities (derived from the individual component failure rates) in a Markov model and calculating the resulting overall failure rate.

Develop-ment step	Goal	Description
Markov analysis	Evaluate system failure rate	Model system states and transition probabilities from in-dividual component failures; calculate overall failure rate. • Calculate overall failure rate, considering FuSa and SOTIF • Meet target failure rate

In addition to these supporting steps, the relevant standards like ISO 26262 and established engineering practices prescribe extensive testing and simulation for verifying the correctness, reliability and availability of complex AD systems and algorithms. For the use case of an SAE Level 4 Highway Pilot, these shall only be intensified compared to systems of lesser criticality. In this context, fault injection campaigns are of particular value and can challenge many design properties, such as sufficient independence of redundant channels, fault tolerance of arbiters and resilience of the overall systems against arbitrary faults in their components.

5.4 SUFFICIENT INDEPENDENCE

5.4.1 INTRODUCTION

As addressed in section 1.4, the evaluated architectural candidates are on an abstraction level which is referred to as system-level conceptual architecture. Design principle D1: Fault Containment Units is followed by all candidate architectures except "monolithic architecture". In other words, the Automated Driving Intelligence (ADI) consists of a set of redundant subsystems that each form a Fault Containment Unit (FCU), assumed to fail independently from other FCUs.
Design principle D1: *Fault Containment Units* is a logical consequence of the existence of the safety-related availability requirement for the ADI (see S2: *AD Intelligence output availability*),

which stems from an availability safety goal of the automated driving system (ADS) item. For example, the Layer-Wise Doer / Checker / Fallback (L-DCF) architecture satisfies the safety-related availability requirement by two redundant and assumedly independent fail-silent functional channels (see Figure 60), which is an application of the Duplex pattern (see section 3.2.1.5).

Incorrect results within the channels of such a two-channel approach have the potential to violate the correctness requirement of the ADI (see S3: *AD Intelligence output correctness*). That is why errors that lead to incorrect results of the intended functionality within the respective functional channels are detected by a checker which will either silence the channel (L-DCF) or alternatively invalidate its outputs. Both error reactions signal an incorrectness of the affected ADI channel to an arbiter, which consequentially will exclude the affected channel from controlling the actuators. Therefore, any error condition that leads to the failure of both redundant channels' intended functionality violates the availability requirement of the ADI. The same applies for error conditions that lead to false positives of the checkers of both functional channels. Remark: If only one of the two functional channels fails – either silently or with an invalidated output – the availability requirement of the ADI remains fulfilled. Because of the above-described requirements, a redundancy approach can only be meaningful if the involved channels fail independently.

In the following discussion, we will not refer to the specific architectures described in this report, but assume such a simplified two-channel configuration (see Figure 59 below), because it is a representative pattern for all architectures that employ redundancy.

Two elements (like functional channels) fail independently if there is an absence of statistically dependent failures of the two elements[60]. This characteristic also outlines the definition of the term "Independence" in ISO 26262[61] and translates into the absence of common cause failures AND cascading failures.

Independence is also a necessary precondition for meeting the target failure rate specified by ISO 26262 for the violation of the ASIL D availability safety goal of the ADS by random hardware faults (PMHF). The PMHF topic is a special challenge because conventional, fail-silent safety mechanisms that lead to a switched-off mode kind of safe state violate the availability safety goal. With respect to the availability safety goal, raw FIT rates of devices need to be taken for evaluating the PMHF, as any fault can lead to a shutdown of the channel. The raw FIT rates of realistic MCUs and SOCs are much higher than the needed PMHF, and only the independent, probabilistic combination of two redundant channels can achieve the ASIL D target.

Implementation-wise one of the main challenges of system-level conceptual architectures for the ADI is demonstrating Independence of redundant functional channels that contribute to the fulfillment of the availability safety goal of the ADS item. As stated above, each independent channel itself must fulfill the safety requirements in terms of correctness, which also brings along intra-channel independence requirements between the channels' intended functionalities and their safety mechanisms. Hint: These independence requirements are most often used as a base for an application of an ASIL decomposition, which reduces the development efforts.

The safety-related, availability-driven aspect results in an independence requirement between channels. However, the reader shall be aware that the correctness aspect is equally important, which results in independence requirements between the channels but also within each chan-

[60] In other words, two elements A and B fail independently if and only if the probability
P(A fails | B failed) = P(A fails) and consequentially P(A fails AND B fails) = P(A fails) * P(B fails).
[61] With "Independence" in this report "Technical Independence" is addressed which shall be distinguished from the independence of parties within an organization involved in safety actions.

nel. This report focuses on the independence requirement between the channels. Independence requirements within a channel (e.g., between a nominal function and a checking layer on top) are already well established in the domain of fail-silent systems. Figure 59 displays the inter- and intra-channel dependencies for a two-channel approach (as it is applied in the Layer-Wise Doer/Checker/Fallback (L-DCF) architecture as an example).

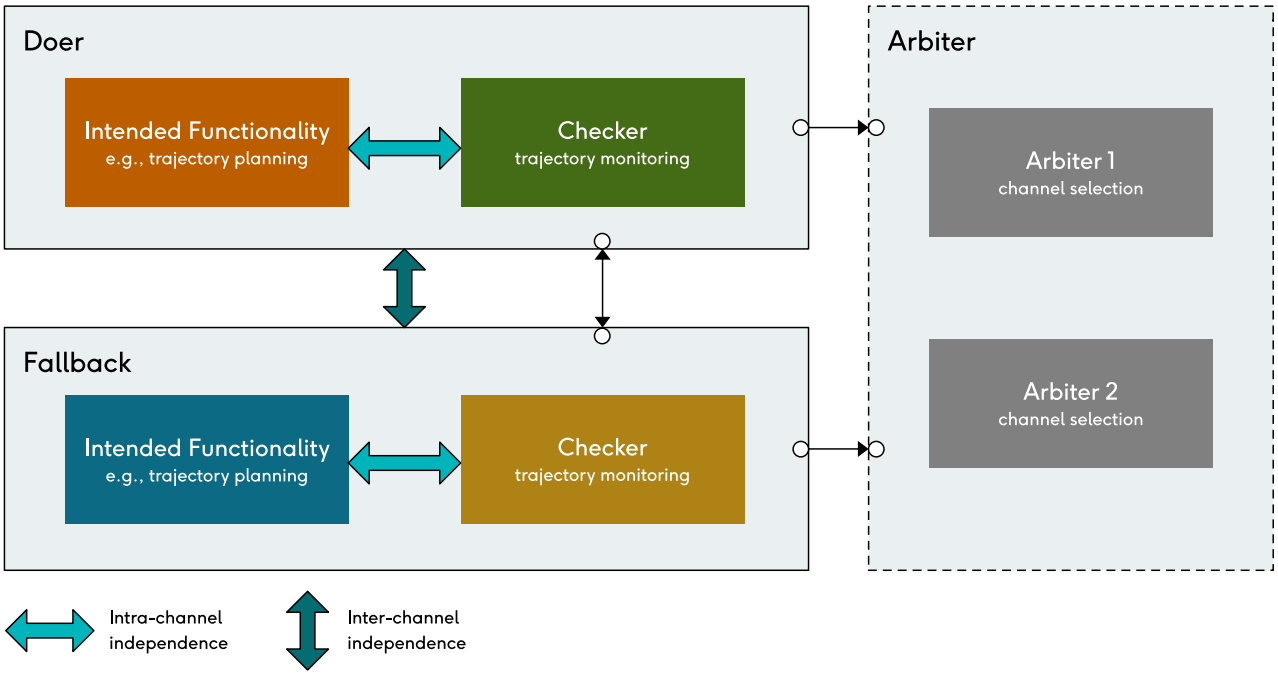


Figure 59: Inter- and intra-channel independences for a two-channel approach like in the L-DCF architecture

Intra-channel and inter-channel Independence have their characteristic challenges that need to be mastered when implementing the conceptual architectures (for more details see juxtaposition in section 5.4.3). At first glance, postulating the independence of the failures of specific elements of the conceptual architecture seems straightforward. However, the true challenge of Independence must be addressed at the frontline of HW- and SW implementation. This shall not downplay the important achievement of realizing that the elements shall be independent.

Every correct decision for Independence on higher abstraction levels can be jeopardized on lower abstractions levels. As an example, diversity of design is a well-known measure for avoiding common cause failures. However, the claim of Independence of allegedly diverse SW components cannot be made without further analysis if codes share the same libraries. Likewise, a hardware example could be the microcontrollers of two different vendors that are perceived as diverse and contributing to the independence of functional channels without the awareness that both use the same 3rd-party IP blocks. Consequently, an independence measure to target common cause failures that is based on diverse HW or SW components must be complemented by requirements aimed at managing diversity by those responsible for implementation. Hint: There are other diversity measures besides diversity of design of the components that are applicable, such as diversity of usage.

The Independence guidance given in this chapter of the report is restricted to a high level of HW architecture and SW architecture (i.e., on the granularity of HW and SW components). Typical challenges faced at this abstraction level in the form of coupling factors will be addressed and independence measures for the resolution thereof will be listed.

The prior paragraphs refer several times to the term “Independence”. It is important to empha-

size that absolute Independence is a characteristic of two or more elements that is striven for by functional safety engineers, yet it is hardly achievable. This insight is in line with the fundamentals of Functional Safety, whose objective has never been to make sure that there is no risk at all but the avoidance of any unreasonable risk. Thus, to achieve Independence the system must be systematically analyzed with high rigor by a multidisciplinary team embedded in an organization with a well fostered safety culture. As a further step, any combination of identified potential for dependent failures needs to be investigated by the team and resolved as far as reasonably achievable. This activity is performed to a rigor level which is referred to by the safety community as "Sufficient Independence".

Without a clear guideline, the borderline between Sufficient Independence and Insufficient Independence is subject to individual engineering judgements that are influenced by attitudes. This situation resembles the "How safe is safe enough" dispute that is present in all aspects of the safety community, not only Functional Safety. Typically, Functional Safety standards provide some general guidance in finding the right answers to this question, but as a matter of fact on some level of detail always leave room for interpretation, which requires expert judgement and the required sense of responsibility. It is important to consider that less rigorous safety procedures might be driven by the commercial goal to reduce costs. The demand for an objective approach for achieving Sufficient Independence in the realm of automated driving systems leads to the initial question of what the ISO 26262 definition of Sufficient Independence is. Although there is no clear definition of the term, there are points of reference:

The following statement can be found in the frame of the Independence, which as a requirement goes hand in hand with every ASIL decomposition applied during development:
These elements are sufficiently independent if the analysis of dependent failures (see Clause 7) does not find a plausible cause of dependent failures that can lead to the violation of an initial safety requirement, or if each identified cause of dependent failures is controlled by an adequate safety measure according to the ASIL of the initial safety requirement.

Which, for a more profound understanding, puts the focus now on the term "plausible". The understanding of the term "plausible" can be facilitated by ISO 26262-9:2018 clause 7.4.2. as another point of reference:
Each identified potential for dependent failures shall be evaluated to determine its plausibility, i.e., if a "reasonably foreseeable" cause exists which leads to the dependent failure and consequently violates a required independence, or freedom from interference, between given elements.

with a clear definition of "reasonably foreseeable" in ISO 26262-1:2018:
"technically possible and with a credible or measurable rate of occurrence"

A good understanding of this mesh of definitions and statements of ISO 26262 serves as a reasonable base for meeting the standard's objectives while elaborating a methodology for achieving Sufficient Independence.

The identification of the potential for dependent failures is done by a so-called dependent failure analysis. The best practice for a dependent failure analysis on system level is to perform as a first step a Fault Tree Analysis (FTA) with a violation of the availability safety goal as the main event.

The FTA takes as an input the system architectural design. Then the next step is to deductively identify plausible causes that lead to a violation of the main event. They are analyzed down to the HW- and SW component level which can be understood as the leaves of the fault tree. The components themselves are analyzed bottom-up by means of a Failure Mode and Effects Analysis (FMEA). This approach can be initiated early in the design process and the components' complete failure modes are identified. Each failure mode will be docked to the existing corre-

sponding leaves of the fault-tree. By understanding the system-level meaning of their resulting component-level failure modes by means of the fault tree, the faults identified in the FMEA can be potentially unmasked as the root cause of a dependent failure. To this aim, as a further step, a minimal cut set analysis shall be performed by one of the established tools for fault tree analysis[62]. This will reveal identical events instantiated multiple times and connected to AND gates in the existing fault tree. Furthermore, the different events of the cut sets are then analyzed for dependent failure initiators based on the outcome of the component-level FMEAs but also by a checklist-driven review performed by a multidisciplinary team. Hereby the abstraction level that is not covered anymore by the FTA is considered.

The mentioned checklist reflects the lessons learned of the industry with respect to dependent failures that helps to realize couplings between the failure modes of components that result from their respective FMEAs. Although ISO 26262 gives a good starting point with the inputs gathered on the basis of the experience of its topic group members, no organization as per ISO 26262 shall neglect to enhance the checklists by its own lessons learned.

Hint: From a holistic perspective, a safe automated driving system requires more than just the Functional Safety aspect. Functional Safety relates to the malfunctioning of E/E systems (faults and failures addressed above). In addition, functional insufficiencies of the intended functionality must be considered (see next chapter "The role of SOTIF in achieving Independence"). This approach is known as the Safety of the Intended Functionality as outlined by ISO 21448. As a consequence, the minimal cut sets addressed above also need to be reviewed for functional scenarios and accompanying triggering conditions that might result in common cause faults.

5.4.2 THE ROLE OF SOTIF IN ACHIEVING INDEPENDENCE

ISO 21448 (2022) defines "Safety of the intended functionality (SOTIF)" as the absence of unreasonable risk due to hazards resulting from functional insufficiencies of the intended functionality and its implementation.

Functional insufficiencies can be seen as unsafe system properties that can be categorized into two types:

- Insufficiencies of specification (e.g., unspecified behavior in a certain operational situation, insufficient specification of the ODD, false assumptions about environmental factors or traffic participants), and
- performance insufficiencies (e.g., limitations of the technical capability of sensor or algorithms, inability to handle misuse activation of the system outside the ODD).

SOTIF-related hazards are initiated by triggering conditions in concrete scenarios (e.g., adverse weather conditions, unknown traffic signs, special combination of driving scenarios). Such triggering conditions may activate functional insufficiencies present at the system level, in a subsystem or channel, as well as in lower architectural levels, in a hardware or software element. A single triggering condition may activate several performance insufficiencies or insufficiencies of specification (e.g., heavy rain impacting different sensors such as radar and camera). This is analogous to the concept of dependent failure initiator from the ISO 26262.
Several different triggering conditions may cause the same functional insufficiency of an element of the system. In addition, a specific combination of triggering conditions may (gradually) activate a functional insufficiency.

Figure 60 summarizes the cause-effect model of SOTIF-related hazards, from the perspective of the system and element level.

[62] Minimal cut sets can be understood as a set of basic events that lead to a violation of the top event when logically ANDed.

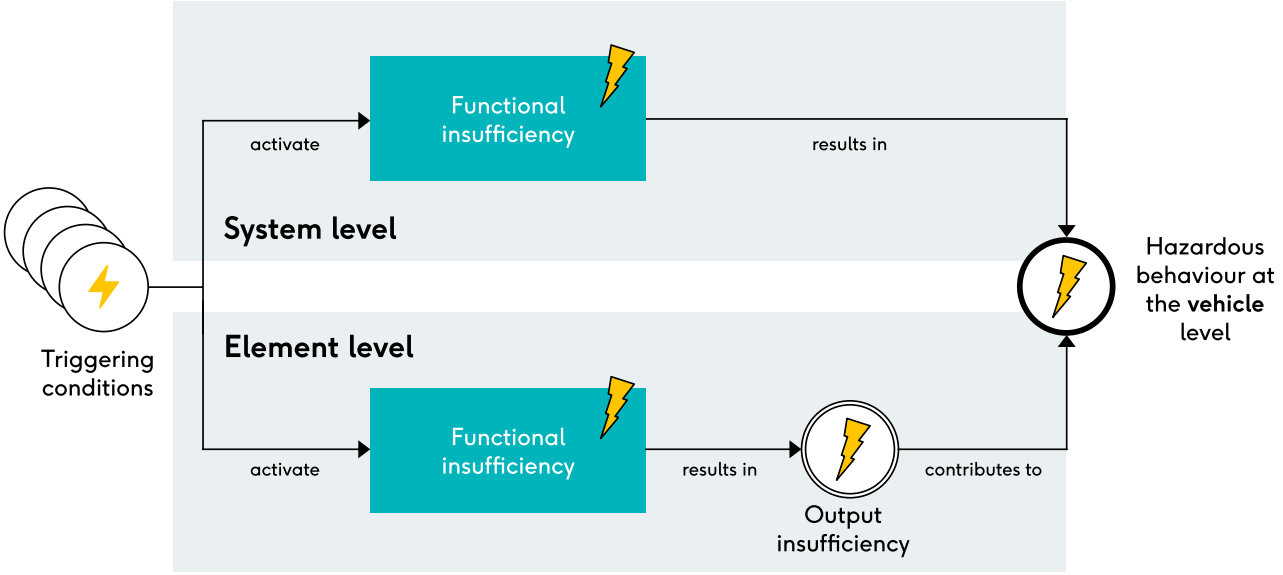


Figure 60: Cause and effect model of SOTIF. Adapted from ISO 21448:2022 Figure 3.

5.4.2.1 ANALYSIS OF DEPENDENCIES FOR SOTIF

When analyzing SOTIF-related hazardous behaviors, both contributing factors – potential functional insufficiencies and potential triggering conditions – as well as their combination, are identified. Other factors, which are not causal but contribute to the occurrence of harm, include:

- the scenario containing conditions in which the hazard can lead to harm, and
- the inability to gain sufficient control of the hazardous event.
- Functional insufficiencies on element level are classified into:
- Single-point functional insufficiency: those that can be activated by one or more triggering conditions and only involve one element.
- Multiple-point functional insufficiency: those that can only be activated in conjunction with functional insufficiencies of other elements, in the presence of one or more triggering conditions.

Such a general root cause classification can be combined with other categorizations to support SOTIF-related analysis, e.g., classifying functional insufficiencies according to the sense-plan-act model. See [39, 21] for a classification of output insufficiencies in four major categories: world model, motion plan, traffic rule, and operational design domain (ODD).
By using appropriate methods, e.g., cause[63] tree analysis or Bayesian networks, the effect chain of triggered functional insufficiencies can be analyzed. Both the functional and the technical aspects shall be considered. In particular, the analysis of common causes and coupling factors that may lead to output insufficiencies at the element level is part of such analysis.
SOTIF-related dependencies may include the following cases:

- Channels or components with the same functional insufficiencies.
- Channels or components with different functional insufficiencies, but dependent triggering conditions activating them (e.g., multiple triggering conditions that happen gradually).
- Combination or propagation of output insufficiencies.

Note that the classification of dependent issues into cascading and common cause ones is also applicable for SOTIF. If an element's output is "insufficient" (i.e., its performance is lower than a given acceptance criteria), the next element in the control or data chain may "cascade" the

[63] To avoid the term "fault", ISO 21448 refers to "cause tree analysis" instead of "fault tree analysis".

issue. Therefore, the propagation of such insufficiencies and the effects on the system's behavior need to be analyzed.

The architectural aspects of the system determine the type of dependent issue, cascading or common cause:

- the hierarchical structure of the elements, and
- the temporal behavior of the elements, including latent issues, i.e., accumulation of undetected/unhandled/mishandled output insufficiencies or faults over time that might combine to cause a SOTIF-related hazardous behavior.

5.4.2.2 INDEPENDENCE AND REDUNDANCY

System-level conceptual architectures can be designed in such a way that existing redundancies are improved by demanding sufficient independence not only from a Functional Safety but also from a SOTIF perspective.

Consider the redundant channel architecture shown in Figure 61:

- A function in a system is implemented by 2 or more redundant channels.
- Each channel can satisfy the sub-function by itself in a given set of driving conditions.
- The correct behavior of any of these redundant channels is sufficient to ensure the safety of the intended functionality.

Note that such an architecture resembles, e.g., the Active/Hot Stand-By pattern of the asymmetric architectural candidates, which require the redundant paths to fail independently.

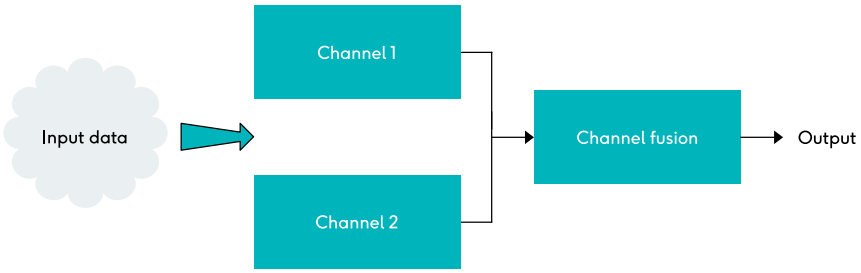


Figure 61: Example of a 2-channel system architecture. Taken from ISO 21448:2022 Figure C.17.

Both channels implement the same function, and each can avoid potential hazards independently. Channel 1 may have a functional insufficiency and Channel 2 may have a different functional insufficiency. Assuming that the channel fusion element has no functional insufficiency, then a potential functional insufficiency in either channel is a multiple-point functional insufficiency.

Figure 62 shows a simple causal model of dependent functional insufficiencies potentially causing hazardous behavior, considering the 2-channel example. The combinations of single or multiple triggering conditions, as well as commonalities in the functional insufficiencies of the channels determine this dependency.

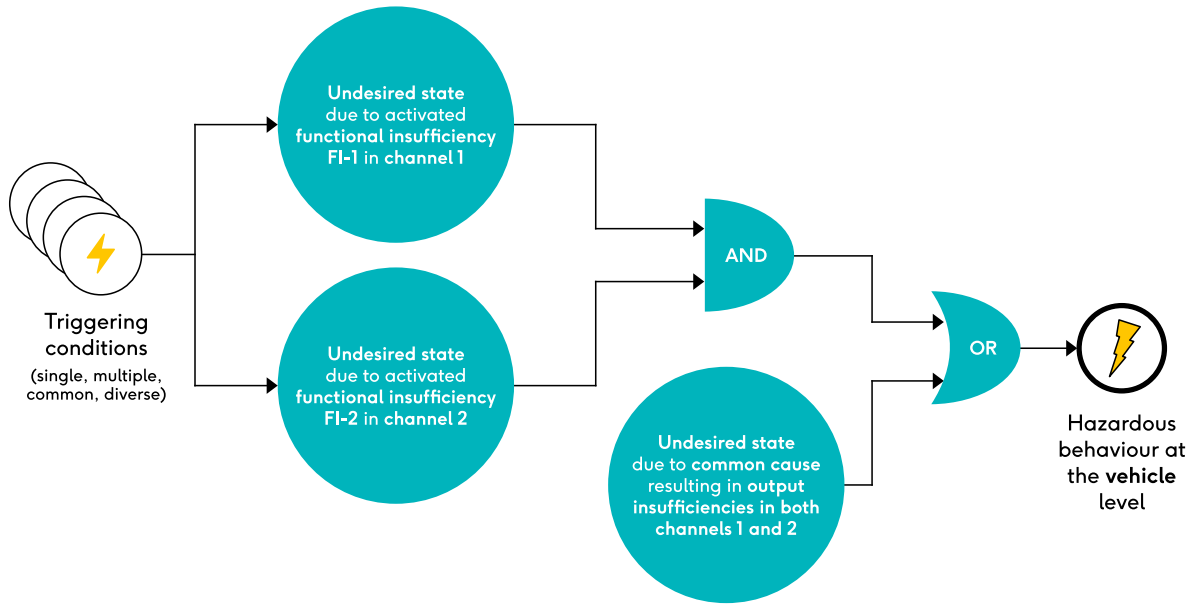


Figure 62: Causal model for dependent functional insufficiencies for a system of two channels. Adapted from ISO 21448:2022, Figure C.18.

At the system level, the mitigation of inter-channel dependencies is required. This is achieved by different methods:

- Analysis of dependencies:
  - Dependency analysis of the channels including known phenomena.
  - Analysis of functional insufficiencies observed in other systems using similar sensors or functions.
  - Analysis of single-channel functional insufficiencies observed during development or via field monitoring providing evidence that the other channel is not affected by this issue.
- "Error guessing" for verification and validation:
  - Testing and simulation to show that the system does not expose guessed common cause or cascading output insufficiencies.
  - Systematically designing the validation field tests in a way that all the known or hypothesized technical weaknesses of the system elements are sufficiently tested.
- SOTIF-related safety measures:
  - Mechanisms that are shown to mitigate common cause functional limitation
  - Functional modifications
  - Addition of new elements (e.g., diverse sensor modality)

5.4.3 COUPLING FACTORS AND DEPENDENT FAILURE/FUNCTIONAL AND OUTPUT INSUFFICIENCY INITIATORS

When two architectural elements fail simultaneously (i.e., within a short enough time interval to have the effect of simultaneous failures), they might fail not just on pure coincidence but on the basis of a common characteristic or due to a relationship with each other. In accordance with ISO 26262 we call such a basis for a joint failure a coupling factor. Yet a coupling factor alone does not make a dependent failure. A dependent failure is enabled by a coupling factor but needs to be triggered by what ISO 26262 calls a dependent failure initiator (DFI). The interaction between the two aspects is well described in the following sub-chapters dedicated to Functional Safety and SOTIF. A DFI can be either a random hardware fault (e.g., a fault in a shared resource with a physical root cause such as electromigration) or a systematic fault (e.g., a development fault or a manufacturing fault) that manifests as a root cause.

The dependent failure initiators addressed in the two chapters below in many cases refer in a generalizing way to incorrectness and non-availability. It shall be understood that incorrectness of an intended functionality (irrespective of its cause) can lead to the unavailability of its outputs in a second step when a checker detects the incorrectness, and the outputs are discarded by the system (see also introduction chapter). The following example shall make that aspect more transparent to the reader: A CAN message might be unavailable which the schedule-aware receiver can detect by a timeout monitor right away. On the other hand, if a CRC check monitoring the intended transmission of a CAN message reveals that its signals have been corrupted, the message will most likely be discarded by its receiver and will no longer be available. So, both initiators result in non-availability of the signals.

5.4.3.1 FUNCTIONAL SAFETY COUPLING FACTORS AND DFIS

Various coupling factors exist that can lead to a dependence of failure of two architectural elements in combination with a single root cause. ISO 26262-9:2018 Annex C separates them into the following coupling factor classes (see also Figure 63):

5.4.3.1.1 SHARED INFORMATION INPUT

The two architectural elements are connected to the same source of information and therefore process the same data. If this data is incorrect or not available (due to systematic faults or random HW faults, both elements will be affected. The resulting dependent failure is a common cause failure.

5.4.3.1.2 COMMUNICATION

One of the two architectural elements receives information from the other one through a communication channel. If this information is incorrect or not available (whether due to systematic faults or random HW faults), both elements will be affected. The resulting dependent failure is a cascading failure.

5.4.3.1.3 COMPONENTS OF IDENTICAL TYPE

The two architectural elements employ instances of identical or very similar components. This is also referred to as homogenous redundancy of these components. If these components malfunction in such a way that their outputs are either incorrect or not available due to an (identical) systematic fault within a relevant time span (emergency operation time interval), a dependent failure arises, which is a common cause failure. Per the nature of this coupling factor, this common characteristic of the elements relates only to systematic faults.

Random hardware faults can occur in any components of the two architectural elements, irrespective of whether they are of identical type or diverse (in other words with a coupling or without a coupling). A joint failure of redundant elements because of a simultaneous appearance of independent random hardware faults is usually considered to be very unlikely.

5.4.3.1.4 (POTENTIAL) UNINTENDED INTERFACE

The two architectural elements have an unknown potential for an unintended interface, through which a cascading fault can let both elements fail simultaneously. The unintended interface is evoked by a dependent failure initiator which is of either systematic faults or random nature.

5.4.3.1.5 SYSTEMATIC COUPLING

The two architectural elements fail due to a common systematic tool error or a common systematic human error. As the naming of the coupling factor reveals, the associated dependent failure initiator is of a systematic nature.

5.4.3.1.6 INSUFFICIENT ENVIRONMENTAL IMMUNITY

The two architectural elements have the same or similar physical characteristics, weakness, or sensitivity which can be affected by the same external environmental disturbance which is the dependent failure initiator. The resulting dependent failure is a common cause failure.

5.4.3.1.7 SHARED RESOURCE

The two architectural elements use the same system, hardware, or software element instance. If this common element instance is faulty or unavailable, both architectural elements will be affected. The resulting dependent failure is a common cause failure. The dependent failure initiator can be either a systematic fault or a random hardware fault of the shared resource.

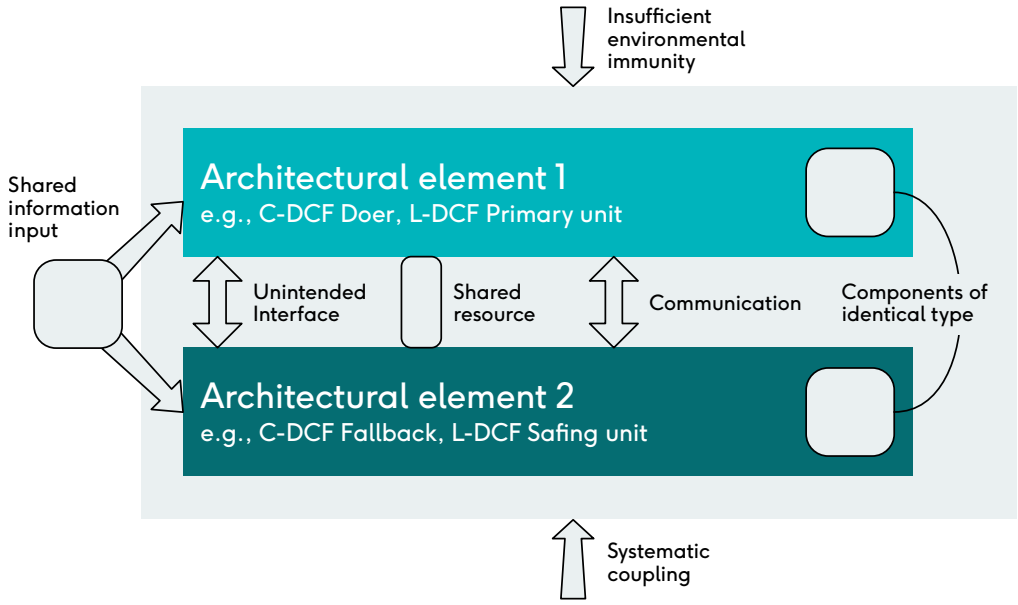


Figure 63: Overview of possible coupling factors. Adapted from ISO 26262-9:2018, Figure C.1.

As already mentioned in 5.4.1, intra- and inter-channel Independence follow the same basic approach, but each presents distinct challenges that must be addressed while considering all availability and correctness aspects. Table 25 constitutes a juxtaposition of the respective importance of the coupling factors which makes this statement more tangible.

Note: A typical intra-channel Independence is the one between an intended functionality and its monitor (safety mechanism). This independence is also the basis for applying an ASIL decomposition between them.

Table 25: Juxtaposition of importance of coupling factors for intra- and inter-channel independence.

Coupling factor class	Intra-channel independence	Inter-channel independence
Communi-cation	**Relevance: high** Remark: Communication between independent elements within a channel usually cannot be avoided (the related plausible dependent failures must be resolved) Resolution: Receiving components needs to be robust with respect to erroneous inputs (e.g., E2E check, robustness for babbling idiot, voltage spikes, …) and the sending components must provide data with the highest ASIL.	**Relevance: medium** Resolution: The coupling factor can be removed or avoided by taking the design decision not to communicate between independent channels. If not feasible, the receiving channel needs to be robust against erroneous inputs and/or the data must be provided by the respective channel with the highest ASIL (fault avoidance).
Components of identical type	**Relevance: low** Remark: Every component has its specific purpose.	**Relevance: high** Remark: It is very likely that a decision for an extra channel results in components of identical type being used.
Unintended interface	**Relevance: high** Remark: As an example, it is very likely that the independent SW components of intended functionality and its monitoring share the same RAM which can result in interference. Furthermore, it is likely that HW components are close to each other, leading to crosstalk.	**Relevance: medium** Remark: While physical separation can come naturally by deploying each channel on separate HW, cost pressure can quickly lead to both channels being exposed to potential unintended interfaces by shared HW. These unintended interfaces can be between SW components or HW components.
Shared resources	**Relevance: high** Remark: The relevance is high because of: • SW components are executed on same HW • usage of same libraries for SW	**Relevance: medium** Remark 1: While deploying each channel on separate HW excludes the sharing of resources, cost pressure can quickly lead to two channels sharing, for instance, infrastructure like clock, power rails Remark 2: Shared resources for SW also need to be considered, as usage of the same libraries has the same exposure as within a channel. Remark 3: Special attention is needed for shared resources that are external to the channels (e.g., same power supply).

Coupling factor class	Intra-channel independence	Inter-channel independence
Shared information input	**Relevance: low** Remark: the Intended functionality and corresponding checker usually use different inputs	**Relevance: high** Remark: Addition of an extra channel within an ADI is prone to shared information input, e.g., same sensor inputs.
Systematic coupling	**Relevance: high** Remark: it is likely that the components within a channel are developed by using the same tools or are produced following the same production process.	**Relevance: high** Remark: it is likely that the components within a channel are developed by using the same tools or are produced following the same production process.
Insufficient environmental immunity	**Relevance: high** Remark: Environmental influences can affect all independent HW components in a channel.	**Relevance: high** Remark: Environmental influences can affect all independent HW components in two independent channels.

A general difference between intra- and inter-channel independence is the resulting scale of components required to be independent, since independent channels means that every component of one channel needs to be independent from every single component of the other channel (n:n relation of independent components). Accordingly, effort-wise there is a major difference when analyzing the required independence.

5.4.3.2 SOTIF COUPLING FACTORS AND DIIS

In this section we apply the concept of dependent failures used in ISO 26262 to address the independence topic in the context of SOTIF. We use the term coupling factor in an analogous way and refer to Dependent output Insufficiency Initiator (DII) instead of dependent failure initiator (DFI).

Table 26 contains a non-exhaustive list of coupling factor classes that are applicable for SOTIF. Note that some coupling factors defined for ISO 26262 dependent failures are still considered, while new (sub)classes are also proposed. The intention is to facilitate the identification of SOTIF-related coupling factors and therefore support a more systematic, comprehensive, and complete dependent insufficiencies analysis.

Table 26: SOTIF-related coupling factors.

Coupling factor class	Application/Examples
Components of identical type/similar sensors	• Different channels using the same sensor modalities and/or technologies, which are not able to cope with specific triggering conditions, e.g., adverse weather conditions, as DII. • Different sensor modalities that share common limitations, e.g., radar and lidar simultaneously failing to recognize stationary objects (see [78]).
Components of identical type/similar algorithms	• Similar algorithms (e.g., perception or planning algorithms, including probabilistic calculations, machine learning approaches) with common performance insufficiencies are used in different channels.
Components of identical type/similar actuators	• Actuators sharing common performance limitations. For example: common limitations regarding environmental conditions (e.g., cold, damp) or load conditions.
Systematic coupling/common specification insufficiency	• Common inaccurate specifications (e.g., ODD conditions), common inaccurate acceptance criteria for performance requirements (e.g., false positive and false negative criteria for perception detection), among other kinds of insufficiency of specification, which is used for the development, including testing, of different channels.
Systematic coupling/common data during development	• The same data is used during the development of the perception, planning, or actuator algorithms in components of different channels that lead to the same functional insufficiency. • Examples of such data: insufficient scenarios to cover the ODD, inappropriate training data for ML-based perception approaches, inappropriate configuration data.

5.4.4 RESOLUTION STRATEGIES FOR IDENTIFIED PLAUSIBLE CAUSES OF DEPENDENT FAILURES AND OUTPUT INSUFFICIENCIES

Every identified plausible dependent failure (i.e., combination of a coupling factor and a dependent failure initiator) and every identified plausible dependent functional insufficiency (i.e., an identical insufficiency due to the same triggering condition or different insufficiencies with statistically dependent triggering conditions) needs to be resolved in order to achieve independence.

According to ISO 26262 a resolution of dependent failures includes “measures for preventing their root causes, or for controlling their effects or for diminishing the coupling factors”. For two redundant channels, whose independence is jeopardized, “controlling the effect” is not feasible without a third channel. Such a third channel can be considered as an external safety mechanism (i.e., external with respect to the two functional channels which are undergoing an analysis of dependent failures). A practical example of such an external safety mechanism is a brake system that performs a braking maneuver within the latest trajectory if both redundant channels of the ADI fail simultaneously and no new trajectories or related setpoints are provided anymore to the actuator systems. Such an approach is good to have as a “fallback from a fallback” but shall not reduce the rigor with which the first two functional channels are kept independent.

The examples of Functional Safety resolution strategies listed below in Table 27 either diminish the coupling factors or prevent/reduce the probability of the root causes. A clear distinction of the two approaches is not always possible (e.g., for a diverse configuration of identical SW components being used) but also of low relevance. What really matters is that the likelihood of a dependent failure is reduced by methods that can be reasonably argued.

All the examples of SOTIF resolution strategies listed below in Table 28 aim at diminishing the coupling factors because the occurrence of external triggering conditions can neither be avoided, nor their occurrence frequencies reduced. Diminishing coupling factors in the context of SOTIF means removing the common functional insufficiency (e.g., design weakness) itself and/or implementing safety mechanisms (e.g., design defenses) against the activated functional insufficiency, at least in one of the affected architecture elements. A potential alternative to such "system modification" measures are strategies related to "functional restrictions" [3], which aim to degrade or limit the intended functionality – for example, by restricting the ODD to exclude known scenarios containing the (common) activating triggering condition (DII).

Choosing the right architectural candidate is a first step towards avoiding coupling factors since asymmetric architectures will reduce susceptibility to CCFs significantly. Once the decision is taken, the most evident way to diminish a coupling factor between two independent FCUs during the implementation of an architectural candidate is to remove it by a design change (internal or external) of the architectural elements required to be independent. As an example, a single power supply for two independent channels constitutes a Functional Safety coupling factor of the class "shared resources". It can be removed by integrating two independent power supplies instead of a single one. As another example, the usage of two identical SW components in two independent channels constitutes a coupling factor of the class "components of identical type". It can be removed by replacing them with two diversely designed and implemented SW components but also interlocking or disabling of its features on one of the functional channels during ADS in active state). Other resolution strategies diminish the coupling to acceptable limits, rather than absolutely removing it on a design level. As an example, a suitable housing diminishes a coupling of the class "insufficient environmental immunity" to acceptable limits with compliance to an EMI standard.

After the resolution of every identified plausible dependent failure and every identified plausible dependent functional insufficiency, the two channels can be regarded as sufficiently independent.

A quantification of unidentified dependent failures during random hardware metric evaluation by a beta factor as suggested by safety standards like IEC 61508 or EN 62061 is not considered feasible by ISO 26262 "since no general and sufficiently reliable method exists for quantifying such failures." (ISO 26262-9:2018 clause 7.4.2).[64]

[64] The only quantification that can be reasonably performed is the one of a "shared HW resource" (coupling factor "shared resources"). The random hardware failures of such a shared HW resource can be considered in the hardware metric calculation as single point faults. Hint: The coupling "Shared information input" might also be related to random hardware faults metric considering HW faults that affect the "Shared Information Input".

Table 27: Functional Safety-related examples for resolution strategies.

Coupling factor class	Examples of resolution strategies
Shared information input	• HW: Diverse redundancy of inputs providing diverse sources of data (e.g., cameras, radar, LiDAR) and performing voting, use of predictive models to rely on prediction until valid data is restored. • HW+SW: provision of the shared input with the highest ASIL, this including the communication links (hint: only affective for systematic faults, not random hardware faults, which need to be considered as single point faults during PMHF evaluation)
Communication	• HW: Use of multiple, diverse, and independent communication channels with data integration checks with respect to failure modes according to ISO 26262-6:2018 clause D.2.4, to ensure that the data can be correctly received or transmitted even if one channel fails. The cascading of wrong values from one channel to the other channel can only be avoided by enforcing fail-silent behavior. • HW+SW: If information is shared between two channels that shall be independent, the information source and also the communication link shall be implemented in the highest ASIL.
Components of identical type	• HW: Using diversely designed components including logic inversion, timing delay and incorporating robust design and testing strategies or diverse technologies (e.g., FPGA versus MCU). • SW: Implementing diversely designed SW components for the same task to reduce the risk of common cause failures due to software bugs. • HW+SW: development of the component according to the highest ASIL (hint: intended functionality itself must be developed according to the highest ASIL, which might be impractical) • Diversification of homogeneous redundancy by diverse configuration, diverse inputs for SW or HW, diverse SW running on homogeneously redundant microcontrollers
Unintended interface	• HW: Physical separation by placing components in different locations to avoid simultaneous failure due to localized DFIs, e.g., for avoidance of crosstalk. • SW: Usage of memory protection unit for separate SW components of different ASIL that use the same physical memory, to ensure Freedom from Interference, i.e., prevent overwriting of each other's memory (note that interference needs to be blocked and not just detected and reacted to). Using a (safe and/or secure) certified hypervisor. Such hypervisors provide independent containers allowing different SW possibly accompanied by different operating systems to operate on the same hardware • HW+SW: Advanced System-on-Chip (SoC) components generally incorporate various CPU and accelerator configurations that share the same address space. CPU-centric mechanisms like memory protection units act as the initial defense against cascading software failures within a specific CPU. They may prove insufficient within complex SoCs that consist of multiple CPUs. Vendors have introduced supplementary hardware combined with proprietary firmware and hardware mechanisms to manage FFI in a multi-master configuration (e.g., multiple CPUs and DMAs sharing the same memory) within the SoC. Utilizing these mechanisms helps to enforce FFI between various combinations of DMAs and CPUs.

Coupling factor class	Examples of resolution strategies
Systematic coupling	• HW: Using diverse component suppliers (assuming that they have different production processes) by sourcing similar components from different suppliers to mitigate the risk of common manufacturing defects, physical separation/isolation, etc. In general: Usage of different production and supply processes • SW: Usage of different compilers or even programming languages, usage of certified compilers
Insufficient environmental immunity	• HW: Robust environmental protection by use of shielding and protective enclosures to guard against moisture, electromagnetic interference, temperature, physical separation/isolation, etc. Possibly, implement logic inversion, timing delays, physical and environmental separation to prevent environmental factors from affecting all inputs simultaneously. Utilizing PCB design techniques and guidelines to reduce EMI.
Shared resource	• HW + SW: Development of the Shared Resource according to the highest ASIL (hint for HW: only affective for systematic faults, not random hardware faults, which need to be considered as single point faults during PMHF, see above) • HW: Functional redundancies within shared elements itself, e.g., diverse redundant, and isolated power supply paths to ensure availability in case one path fails, ECC for shared memories, ability of identifying, isolating and dynamically reconfiguring a failing shared resource, etc. • SW: When being executed on the same target: use of interrupt priority schemes, memory and peripheral access protection.

In addition to the coupling-specific resolution strategies mentioned in the table above in a more general way models can serve as virtual sensors in some cases, replacing faulty or missing sensor inputs to maintain availability requirements. These models leverage historical data, system dynamics, and machine learning algorithms to estimate critical parameters, such as vehicle speed or object distance, when actual sensor data is compromised. By providing accurate and reliable estimations, these models help to prevent dependent failures ensuring redundant subsystems remain operational. This capability is vital in fail-operational designs, where maintaining functionality during sensor faults is essential for safety and availability.

Dynamic reconfiguration is an important strategy to address dependent failures while maintaining availability requirements. This approach involves real-time adaptation of system behavior in response to faults or degraded components, such as faulty sensors, ECUs, or communication links. For instance, if a sensor fails that provides shared input to ECUs, which shall be independent, the system can dynamically reconfigure to rely on alternate inputs, redundant components, or predictive models to maintain functionality.

Predictive maintenance is a proactive strategy that uses real-time data, analytics, and machine learning models to predict the health and remaining life of critical components. By continuously monitoring sensors, ECUs, and other system elements, predictive maintenance identifies early signs of wear, degradation, or failure before they lead to system downtime. In the context of dependent failures, the predictive maintenance technique can detect potential faults, such as sensor failures or ECU malfunctions being involved in couplings, and trigger maintenance or reconfiguration actions to prevent system-wide disruptions This by addressing potential faults

proactively, which ensures that components are serviced or replaced before their failure affects availability, supporting the fail-operational requirements of the system and maintaining overall safety and functionality.

Table 28: SOTIF-related examples for resolution strategies.

Coupling factor class	Examples of resolution strategies
Components of identical type/ similar sensors	• HW: Use of diverse sensor technologies, considering, e.g., increased resolution of sensor measurement, improved sensor disturbance detection. • SW: Application of sensor fusion algorithms.
Components of identical type/ similar algorithms	• Use of different algorithms for perception, e.g., statistical methods and ML-based approaches. • Use of different algorithms for planning, e.g., considering different approaches for situation analysis, prediction models.
Components of identical type/ similar actuators	• HW: Use of diverse actuator technologies, considering, e.g., increased accuracy, reduction of response times.
Systematic coupling/common specification insufficiency	• Application of state-of-the-art development processes.
Systematic coupling/common data during development	• Use of different sources of data, collected in real traffic situations (field tests) and generated by simulations. • Application of data validation methods, e.g., cross-validation, separation of training and test data.

5.4.5 INDEPENDENCE METRIC

Identifying the potential for plausible dependent failures and dependent functional insufficiencies and resolving them are unnegotiable stages on the track to Sufficient Independence. It is the stage of resolution that brings up the difficult question for the diligent engineers that is already addressed in the introduction: Is the achieved independence sufficient – in the sense that the item, into which the analyzed independent architectural elements are integrated, is safe enough?

The achievement of absolute independence is considered impossible, which is also true for absolute safety itself. Following the definition of Functional Safety in ISO 26262:2018, it is not the avoidance of any risk that needs to be ensured, but the avoidance of unreasonable risk – an insight which is the basis for its clauses. The room for interpretation when judging the sufficiency of independence might be significant, yet it can be narrowed with an objective methodological approach that supports the achievement of functional safety in multiple facets. The diligent engineer is supported by it and as a positive side-effect, design decisions are less prone to be influenced by non-technical aspects (e.g., approaching deadlines, estimated efforts exceeded, lacking expertise, ...).

The key question when setting up an objective methodology for judging sufficient independence is whether to go for a qualitative or quantitative approach. A qualitative approach tends

to go hand in hand with a binary judgement on whether dependencies are there at all in a given (architectural) design or not, which likely is a too theoretical approach because in real-world automotive designs not every dependency can be removed. That is why the challenge lies rather in identifying all possible dependencies and judging their plausibility by evaluating the rate of occurrence of related dependent failures and functional insufficiencies – thus the probability of a violation of the availability safety goal (considering the focus in this report on the independence requirement between channels – see introduction). For plausible dependent failures the mentioned probability needs to be reduced by the application of the outlined resolution strategies. This intent is very well in line with ISO 26262, which introduces the "sufficient independence" terminology (absence of reasonably foreseeable dependent failures). The combination of these signposts of ISO 26262 can be associated more with a quantitative evaluation ("credible or measurable rate of occurrence") than a mere qualitative decision of either an "independent" or a "dependent" status. ISO 21448 is not mentioned here because it does not elaborate the Independence topic in detail as ISO 26262 does. However, its statements can be easily transferred from the world of dependent failures to the world of dependent functional insufficiencies.

The refined approach proposed by the Safety & Architecture working group is a semi-quantitative one in the sense that the resolution strategy examples presented above shall be judged by their ability to reduce the probability of a violation of the availability safety goal by allocating them to "Independence Coverage" categories.

The proposed "Independence Coverage" categories are as follows:

HIGH:
Probability of CC fault or functional insufficiency or CF affecting the item availability within the EOTI reduced by 99%

MEDIUM:
Probability of CC fault or functional insufficiency or CF affecting the item availability within the EOTI reduced by 90%

LOW:
Probability of CC fault or functional insufficiency or CF affecting the item availability within the EOTI reduced by 60%

with

CC ... common cause

CF ... cascading fault

EOTI ... emergency operation time interval

Note 1: "... within the EOTI ..." refers to "Dependent failures can manifest themselves simultaneously, or within a sufficiently short time interval, to have the effect of simultaneous failures." (ISO 26262-1:2018 3.29)"

Note 2: Considering the ISO 26262 risk definition, the probability reduction can be understood as a risk reduction.

The allocation process for the Independence Coverage is again based on an engineering judgement. However there is a major difference with respect to individual engineering judgments being influenced by attitudes that were discourages in the beginning of the chapter: Independence Coverages (taking into account parameters like component complexity, configuration

space, etc.) are to be determined by an expert circle and shall be made available in an organization-internal guideline or even standardized (at least informatively and as a starting point of discussion with a need to take into account the pre- and border- conditions of a given context). Furthermore, the Independence Coverages are not situational because they are not decided in the heat of the moment within an automotive engineering project but deliberately and thus more objectively in calm times, which is a solid base for executing fail-degraded automotive projects.

Hint: This methodology resembles the ISO 26262 approach for judging the effectiveness of applied safety mechanisms for detecting and mitigating random hardware faults, i.e., their diagnostic coverage (see ISO 26262-5:2018 Annex D).

Sufficient independence is achieved if an Independence Coverage of 99% (i.e., a "HIGH" classification) is achieved for all identified potentials of plausible dependent failures and dependent functional insufficiencies. This can be achieved either by deciding for the application of a resolution strategy that is allocated to an Independence Coverage of "HIGH" right away, or alternatively by combining resolution strategies that have been allocated to a lower category. This approach offers the following combinations of categories with the described resulting outcomes for an Independence Coverage:

MEDIUM + MEDIUM = HIGH
rationale: equality of categories' risk reduction:
$(1-0.90) \times (1-0.90) = 0.01 = (1-0.99)$

LOW + LOW = MEDIUM
rationale: equality of categories' risk reduction:
$(1-0.60) \times (1-0.60) = 0.16 \sim (1-0.90)$

MEDIUM + LOW = HIGH
rationale: equality of categories' risk reduction:
$(1-0.60) \times (1-0.90) = 0.04 \sim (1-0.99)$

IMPORTANT: Be aware that the combinations above are based on the assumption of independence of applied Independence Measures. Whether this assumption holds true shall be investigated by the applier for the respective combination at least based on engineering judgement. As an example, the combination of two MEDIUM measures (diverse inputs and diverse configurations) for two homogeneous software components seems reasonable to achieve HIGH independence, but expert judgement is needed to rule out overlapping effects that render this evaluation too optimistic.

Example: Developing the combination of an SOC + platform SW that is deployed to both functional channels of an ADI according to ASIL D fault avoidance can be regarded as an Independence measure with a MEDIUM Independence Coverage – this taking the complexity of the SOC, the platform SW[65] and also their multitude of interactions into account. An additional Independence with at least an Independence Coverage LOW is needed. Diversifying the homogeneity of the combination SOC + platform SW by differently configuring the platform SW and also running different application SW on it can be argued as a resolution strategy with at least LOW Independence Coverage. Applying both these independent resolution strategies results in a HIGH Independence Coverage. This means that the potential for dependent failures due to the coupling "components of identical type" is sufficiently resolved.

[65] It is considered impossible to avoid all design faults in complex safety-related elements, especially for large, complex software [25], like a platform SW.

OUTLOOK

With the publication of the second edition of our report, the Safety & Architecture working group is entering a new phase. Building on the momentum and insights gained so far, we will continue our activities in a refreshed format that maintains a focused and collaborative exploration of key challenges in the field of automated driving (AD).

To date, our efforts have culminated in a comprehensive report that describes and evaluates conceptual system architectures suitable for an AD Intelligence. This achievement was made possible through the valuable contributions of a diverse group of collaborators from both academia and industry. Their input has helped ensure that the report reflects a broad spectrum of perspectives and expertise.

Looking ahead, we intend to use this report as a foundation for deep dives on specialized topics. These efforts will be carried out in smaller, interest-driven groups and will result in a series of white papers that build upon and extend the findings of the main report.

We have identified three particularly relevant topics for this next phase. These topics broaden the scope of our work to include systems adjacent to the AD Intelligence and address a critical implementation challenge where architectural measures can provide significant value:

- **Sensor System Redundancy:** Automated driving systems rely on a combination of sensor modalities—such as cameras, radar, and lidar—to perceive their environment. Redundancy is needed not only to achieve high availability, but also to compensate for each sensor's specific weaknesses. Ensuring sufficient independence between redundant sensors is essential for safety yet remains an open question. We aim to explore how such independence can be achieved and evaluated in practice.
- **Actuator System Redundancy:** Redundant actuators are equally vital for safe operation. The actuators in the vehicle (steering, braking, and powertrain) need to jointly ensure that a single, consistent vehicle trajectory is followed to achieve safe and stable behavior. Different architectural solutions exist to achieve such consistency between actuators or to detect inconsistencies/errors when they arise. We will investigate these approaches and their implications for system design and safety assurance.
- **Safe AI-Based Systems:** AI (or more specifically ML) plays a central role in many functional blocks of the AD stack, particularly in perception, but increasingly also in prediction and planning. Some organizations are exploring end-to-end (E2E) AI models for their AD systems. However, a holistic approach to ensuring the safety and regulatory compliance of AI in this context is still lacking. We propose to develop such an approach by integrating architectural measures, development processes, and monitoring measures.

We invite all interested parties to join us in these follow-up efforts. Your expertise and engagement will be crucial as we tackle these complex and evolving challenges together.

Finally, we would like to express our sincere gratitude to all current and past contributors, as well as to our external reviewers, for their dedication and valuable input throughout this journey.

TERMINOLOGY

TERMINOLOGY FROM STANDARDS AND LITERATURE

The "Safety & Architecture" Working Group makes use of the terminology laid out in different industry standards and literature. Please refer to the listed standards for all terms not specifically defined in the following.

A small number of terms have been added as new definitions to clarify the scope of the "Safety & Architecture" Working Group.

For terms related to systems, faults, and failures, we use the following (in order of preference):

- ISO 26262:2018 "Road vehicles – Functional safety" [2]
- IEC 61508:2010 "Functional safety of electrical/electronic/programmable safety-related systems" [30]
- ISO 21448:2022 "Road vehicles – Safety of the Intended Functionality" [3]
- Algirdas Avizienis, J-C. Laprie, Brian Randell, and Carl Landwehr. "Basic concepts and taxonomy of dependable and secure computing." IEEE transactions on dependable and secure computing 1, no. 1 (2004) [1]

For terms related to AD, we use the following (in order of preference):

- ISO/SAE PAS 22736 "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles" [79] (based on SAE J3016_202104 [7])
- BSI PAS 1883:2020 "Operational Design Domain (ODD) taxonomy for an automated driving system (ADS) - Specification" [80]

Please also refer to the relevant databases maintained by ISO and IEC:

- https://www.iso.org/obp/ui
- http://www.electropedia.org/

Term	Reference or definition	Notes
AD Intelligence	computational unit between the sensors and actuators	
Architecture	ISO 26262:2018-1	
Automated Driving System (ADS)	ISO/SAE PAS 22736	
Availability	ISO 26262-1:2018	
Cascading failures	ISO 26262-1:2018	In literature, one will often find the following differentiation: Failures of redundant systems due to systematic weaknesses of the architecture are caused by common cause initiators (CCI) and coupling faults (= ISO 26262 cascading faults). In many standards this differentiation is not done, e.g., IEC 61508-6:2010, Annex D: The term CCF is often used to cover all kinds of dependent failures as it is done in this annex. According to an Exida 2010 Safetronic paper, ISO 26262 is the first standard to distinguish between the two distinct phenomena.
Channel	subsystem, i.e., a separable building block of a system, composed of a perception element, and a planning element, i.e., the plan stage of the so-called "sense, plan, act" model of automated driving, in other words the end-to-end functionality from sensor input to trajectory output.	According to IEC 61508 a channel "implements independently an element safety function", which can be understood as a safety mechanism in ISO 26262:2018 terminology. In contrast, within the context of this report a channel implements a safety-related function (which includes the intended functionality) and is not necessarily independent from other channels. Independence is an additional requirement for a channel.
Common cause failure (CCF)	ISO 26262-1:2018	
Common mode failure (CMF)	ISO 26262-1:2018	
System-level Conceptual architecture	An abstract, high-level architecture that does not specify technical (e.g., HW, SW) components.	This is similar to the "Functional Safety Concept" required by ISO 26262:2018-3.
Controllability	ISO 26262-1:2018	
Coupling factors	ISO 26262-1:2018	
Dependability	Avizienis, TR 2004-47	• The paper defines this as encompassing the following attributes (quote): • availability: readiness for correct service [see also ISO 26262-1:2018] • reliability: continuity of correct service • safety: absence of catastrophic consequences on the user(s) and the environment [see also ISO 26262-1:2018] • integrity: absence of improper system alterations • maintainability: ability to undergo modifications
Dependent failures	ISO 26262-1:2018	
Dependent failure initiator (DFI)	ISO 26262-1:2018	

Term	Reference or definition	Notes
Diagnostic coverage (DC)	ISO 26262-1:2018	In the context of ISO 26262, this only covers HW faults. For our purposes, we use the same term to cover SW faults as well, which goes hand in hand with the decision to quantify (systematic) SW faults.
Diversity	ISO 26262-1:2018	
Dual-point failure	ISO 26262-1:2018	
Dual-point fault	ISO 26262-1:2018	
Dynamic Driving Task (DDT)	ISO/SAE PAS 22736	
DDT fallback	ISO/SAE PAS 22736	
Dynamic elements	BSI PAS 1883	
Ego vehicle	BSI PAS 1883	Used instead of "subject vehicle".
Element	ISO 26262-1:2018	
Emergent behavior		Behavior that cannot be attributed to one individual system alone, but arises in the interplay of various systems, components etc.
Environmental conditions	BSI PAS 1883	
Error	ISO 26262-1:2018	
Failure	ISO 26262-1:2018	
Fail-silent		In the case of a detected safety-related fault a transition into a safe state is initiated in which the intended functionality of an element is no longer provided (depending on the element this can mean a shut-off of communication to other elements or a shut-off of actuators)
Fault	ISO 26262-1:2018	
Fault-Containment Unit (FCU)	*subsystem with its own hardware and software, whose faults are prevented from propagating to its receivers.* ***Note 1**: Fault propagation is prevented by means of FCU-internal and/or external safety mechanisms, which are designed to ensure absence of cascading failures.* ***Note 2**: Faults with a potentially changed semantic (by an internal safety mechanism) propagate via FCU interfaces. Therefore, each interface of an FCU should be defined so that the system can react to such faults (i.e., the failure modes of the interfaces should be made known to its receivers).*	• On system level it will be a responsibility of the receivers to manage the failure modes of an FCU in a safe way. • For system-level conceptual architectures we assume that each FCU fails independently from other FCUs. This requires an absence of common cause failures that needs to be ensured by engineering measures. The assumption is interpreted to mean that potential dependent failures may exist between FCUs defined in the architecture, but must be mitigated by either reducing the probability of the root cause, reducing the coupling factors, or controlling their effects. • An arbitrary failure (including Byzantine failures) of an FCU must not lead to a failure of the complete ADI; ensuring this property is a key requirement for the system-level conceptual architectures.
Formal verification	ISO 26262-1:2018	

Term	Reference or definition	Notes
Functional insufficiency	ISO 21448:2022	Insufficiency of specification or performance insufficiency. Both terms are also defined in ISO 21448:2022.
Interface	Avizienis, TR 2004-47 [1]	The paper distinguishes between the "service interface" and the "use interface".
Item	ISO 26262-1:2018	
Hazard	ISO 26262-1:2018	
Malfunction	ISO 26262-1:2018	Following the approach of ISO 21448:2022, malfunctions in this report are associated as an unintended behavior of an item due to faults. This is in contrast to the output insufficiencies on element level and functional insufficiencies on system level of the intended functionality, which can lead to similar hazardous behavior on the vehicle level.
Mapping	The process of transforming a conceptual architecture into a technical HW and/or SW architecture.	• The same conceptual architecture can be mapped to many different HW/SW solutions. • Certain considerations need to be applied during the mapping to ensure properties of the conceptual architecture are not lost.
Minimal Risk Condition (MRC)	ISO/SAE PAS 22736	
Minimal Risk Maneuver (MRM)	BSI PAS 1883	In a highway ODD, there are multiple possible MRMs, e.g., reducing speed and continuing to the next rest stop, pulling over to the emergency lane, or coming to a controlled stop in the current lane. These differ by their inherent safety and the capability and timeframe necessary to execute them.
Misuse	ISO 21448:2022	An example of a direct misuse is the activation of a highway pilot in an urban setting. An example of an indirect misuse is a driver falling asleep and not monitoring an L2 system during operation.
Monitor	ISO/SAE PAS 22736	
Object and Event Detection and Response (OEDR)	ISO/SAE PAS 22736	
Operational Design Domain (ODD)	ISO/SAE PAS 22736	
Output insufficiency	ISO 21448:2022	
Passenger car	ISO 26262-1:2018	
Performance limitation	ISO 21448:2022	
Random hardware fault	ISO 26262-1:2018	
Request to intervene	ISO/SAE PAS 22736	
Routine/normal operation	ISO/SAE PAS 22736	
Safety	ISO 26262-1:2018	
Safety architecture	ISO 26262-1:2018	
Safety case	ISO 26262-1:2018	
Safety Element out of Context (SEooC)	ISO 26262-1:2018	

Term	Reference or definition	Notes
Safety goal	ISO 26262-1:2018	
Scenery	BSI PAS 1883	
Service	Avizienis, TR 2004-47	
Severity	ISO 26262-1:2018	
System	ISO 26262-1:2018	
Systematic fault	ISO 26262-1:2018	
Triggering condition	ISO 21448:2022	
Validation	ISO 26262-1:2018	
Verification	ISO 26262-1:2018	
Vulnerable Road User (VRU)	BSI PAS 1883	

REFERENCES

[1] A. Avizienis, J.-C. Laprie, B. Randell and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, 2004.
[2] ISO, "ISO 26262:2018 Road vehicles - Functional safety," 2018.
[3] ISO, "ISO 21448:2022 Road vehicles - Safety of the Intended Functionality," 2022.
[4] Aptiv; Audi; Baidu; BMW; Continental; Daimler; FCA; Here; Infineon; Intel; Volkswagen, "Safety First for Automated Driving," 2019.
[5] H.-P. Schoener and J. Antona-Makoshi, "Testing for Tactical Safety of Autonomous Vehicles," in 30th Aachen Colloquium Sustainable Mobility, Aachen, Germany, 2021.
[6] ISO, "ISO/SAE 21434:2021 Road vehicles - Cybersecurity engineering," 2021.
[7] SAE, "SAE J3016 Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems," 2021.
[8] Economic Commission for Europe - Inland Transport Committee, "Proposal for a new UN Regulation on uniform provisions concerning the approval of vehicles with regards to Automated Lane Keeping System," 2020.
[9] H. Egerth and S. Yantis, "Visual Attention: Control, Representation and Time Course," Annual Review of Psychology, vol. 48, pp. 269-297, 1997.
[10] D. Dvorak, "NASA Study on Flight Software Complexity," Jet Propulsion Laboratory, California Institute of Technology, 2009.
[11] J. McDermid and T. Kelly, "Software in safety critical systems: achievement and prediction," Nuclear Future, vol. 2, no. 3, pp. 140-146, 2006.
[12] J. Gray, "Why do computers fail and what can be done about it?," Tandem Computer Corporation, 1985.
[13] G. Li, S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer and S. Keckler, "Understanding error propagation in deep learning neural networks (DNN) accelerators and applications," in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2017.
[14] N. Kalra and S. Paddock, "Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?," Transportation Research Part A: Policy and Practice, vol. 94, pp. 182-193, 2016.
[15] R. Young, "Automated driving system safety: Miles for 95% confidence in "vision zero"," SAE International Journal of Advances and Current Practices in Mobility, vol. 2, no. 6, pp. 3454-3480, 2020.
[16] X. Zhang, J. Tao, M. Törngren, J. Gaspar Sanchez, M. Rusyadi Ramli, X. Tao, M. Gyllenhammar, F. Wotawa, N. Mohan, M. Nica and H. Felbinger, "Finding Critical Scenarios for Automated Driving Systems: A Systematic Mapping Study," IEEE Transactions on Software Engineering, vol. 49, no. 3, pp. 991-1026, 2023.
[17] Statistisches Bundesamt (Destatis), "Verkehrsunfälle 2018," 2020.
[18] P. Koushki and F. Balghunaim, "Determination and Analysis of Unreported Road Accidents in Riyadh, Saudi Arabia," Journal of King Saud University - Engineering Sciences, vol. 3, pp. 101-118, 1991.
[19] SAE, "SAE ARP4754A Guidelines for Development of Civil Aircraft and Systems," 2010.
[20] SAE, "SAE ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," 1996.
[21] Y. Fu, J. Seemann, C. Hanselaar, T. Beurskens, A. Terechko, E. Silvas and W. Heemels, "Characterization and Mitigation of Insufficiencies in Automated Driving Systems," in The 27th International Conference on the Enhanced Safety of Vehicles (ESV), Yokohama, Japan, 2023.
[22] UL, "ANSI/UL 4600 (Ed. 3) Evaluation of Autonomous Products," 2023.
[23] SAE, "SAE J3018 Safety-Relevant Guidance for On-Road Testing of Prototype Automated Driving System (ADS)-Operated Vehicles," 2020.
[24] S. Shalev-Shwartz, S. Shammah and A. Shashua, "On a Formal Model of Safe and Scala-

ble Self-driving Cars," Mobileye, 2017.
[25] H. Kopetz, "An Architecture for Driving Automation," 2020. [Online]. Available: http://www.the-autonomous.com.
[26] J. Lala, in First IFIP Workshop on Intelligent Vehicle Dependability & Security, 2021.
[27] Statistik Austria, "Straßenverkehrsunfälle mit Personenschaden, Jahresergebnisse 2018," 2019.
[28] P. Liu, R. Yang and Z. Xu, "How Safe Is Safe Enough for Self-Driving Vehicles?," Risk Analysis, vol. 39, no. 2, pp. 315-325, 2018.
[29] P. Koopman, "Safety Architecture Patterns," 2021. [Online]. Available: https://users.ece.cmu.edu/~koopman/lectures/ece642/Xtra_SafetyArchPatterns.pdf. [Accessed 30 01 2025].
[30] IEC, "IEC 61508:2010 Functional safety of electrical/electronic/programmable safety-related systems," 2010.
[31] C. Hanselaar, E. Silvas, A. Terechko and W. Heemels, "Detection and Mitigation of Functional Insufficiencies in Autonomous Vehicles: The Safety Shell," in IEEE 25th International Conference on Intelligent Transportation Systems (ITSC), 2022.
[32] A. Armoush, "Design patterns for safety-critical embedded systems," PhD thesis, RWTH Aachen University, 2010.
[33] M. L. Shooman, "Reliability of computer systems and networks: fault tolerance, analysis and design," in N-Modular Redundancy, Wiley-Interscience, 2002, pp. 145-201.
[34] J. von Neumann, "Probabilistic Logics," in Automata Studies, Princeton University Press, 1956.
[35] R. Lyons and W. Vanderkulk, "The Use of Triple-Modular Redundancy to Improve Computer Reliability," IBM Journal, pp. 200-209, 1962.
[36] U. Santoni, R. Mariani and J. Weast, "Independent safety monitoring of an automated driving system". United States Patent US20200017114A1, 23 09 2019.
[37] A. Terechko, Y. Fu, C. Hanselaar and K. Schuerman, "NXP Tech Blog," 3 2023. [Online]. Available: https://community.nxp.com/t5/NXP-Tech-Blog/Daruma-Design-Pattern-for-Safe-and-Continuous-Automated-Driving/ba-p/1609514. [Accessed 21 1 2025].
[38] C. Hanselaar, E. Silvas, A. Terechko and W. Heemels, "Detection and Mitigation of Functional Insufficiencies in Autonomous Vehicles: The Safety Shell," in 2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC), Macau, 2023.
[39] Y. Fu, J. Seemann, C. Hanselaar, T. Beurskens, A. Terechko, E. Silvas and M. Heemels, "Characterization and Mitigation of Insufficiencies in Automated Driving Systems," 15 4 2024. [Online]. Available: https://arxiv.org/abs/2404.09557. [Accessed 21 1 2025].
[40] C. Hanselaar, Y. Fu, A. Terechko, J. Seemann, T. Beurskens and E. Silvas, "Evaluation of the Safety Shell Architecture for Automated Driving in a Realistic Simulator," in 2024 IEEE Intelligent Vehicles Symposium (IV), Jeju Island, South Korea, 2024.
[41] H. Kopetz, "Method for controlling a technical device". Patent EP4009121B1, 07 12 2020.
[42] H. Kopetz, "Control system and method for the safe control of a technical System". Patent EP4455812A1, 25 04 2023.
[43] M. Wagner, J. Ray, A. Kane and P. Koopman, "A safety architecture for autonomous vehicles". Patent EP3400676B1, 2017.
[44] T. Bijlsma, A. Buriachevskyi, A. Frigerio, Y. Fu, K. Goossens, A. O. Örs, P. J. van der Perk, A. Terechko and B. Vermeulen, "A Distributed Safety Mechanism using Middleware and Hypervisors for Autonomous Vehicles," in 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, 2020.
[45] Y. Fu, A. Terechko, J. Groote and A. Saberi, "A formally verified fail-operational safety concept for automated driving," SAE Intl., pp. 7-21, 17 Jan 2022.
[46] Audi AG; BMW AG; Daimler AG; Porsche AG; VW AG, "Standardized E-GAS Monitoring Concept for Gasoline and Diesel Engine Control Units," 2013.
[47] B. Kaiser, B. Monajemi Nejad, D. Kusche and H. Schulte, "Systematic design and validation of degradation cascades for safety-relevant systems," in The 2nd International Conference on Engineering Sciences and Technologies, 2017.

[48] N. Mohan, P. Roos and J. Svahn, "System and Method for Controlling a Motor Vehicle to Drive Autonomously". Patent WO2019125269A1, 27 06 2019.
[49] N. Mohan, J. Svahn and P. Roos, "System and method for controlling a motor vehicle to drive autonomously". Patent WO2019125268A1, 27 06 2019.
[50] N. Mohan, M. Törngren and V. Izosimov, "Challenges in architecting fully automated driving; With an emphasis on heavy commercial vehicles," in Workshop on Automotive Systems/Software Architectures, 2016.
[51] M. Törngren, X. Zhang, N. Mohan, M. Becker, L. Svensson, X. Tao, D. Chen and J. Westman, "Architecting Safety Supervisors for High Levels of Automated Driving," in Proceedings of the 21st IEEE Int. Conf. on Intelligent Transportation Systems, 2018.
[52] N. Mohan and M. Törngren, "AD-EYE: A Co-Simulation Platform for Early Verification of Functional Safety Concepts," in WCX SAE World Congress Experience, Detroit, 2019.
[53] Audi, "zFAS the Brain of piloted Driving and Parking (nVIDIA GPU Technology Conference)," 2015. [Online]. Available: https://on-demand.gputechconf.com/gtc/2015/presentation/S5637-Matthias-Rudolph.pdf.
[54] udi, "Zentrales Fahrerassistenzsteuergerät," [Online]. Available: https://www.audi-mediacenter.com/de/fotos/detail/zentrales-fahrerassistenzsteuergeraet-52989. [Accessed 31 03 2025].
[55] Audi, "Audi A8 - Central driver assistance controller (zFAS)," 7 2017. [Online]. Available: https://www.audi-technology-portal.de/en/electrics-electronics/driver-assistant-systems/audi-a8-central-driver-assistance-controller-zfas. [Accessed 21 11 2023].
[56] Tesla Inc., "Tesla AI Day 2022," 01 10 2022. [Online]. Available: https://www.youtube.com/watch?v=ODSJsviD_SU&ab_channel=Tesla.
[57] AutoPilot Review, "Tesla Hardware 4 – Full Details and Latest News," 2023. [Online]. Available: https://www.autopilotreview.com/tesla-hardware-4-rolling-out-to-new-vehicles/.
[58] BMW Group, "Safety Assessment Report," 2020.
[59] J. Yoshida, "EE Times," 29 04 2020. [Online]. Available: https://www.eetimes.com/unveiled-bmws-scalable-av-architecture/.
[60] S. Shalev-Shwartz, M. Molnar, I. Granot, A. Shany and A. Shashua, "A Safety Architecture for Self-Driving Systems," 2024. [Online]. Available: https://static.mobileye.com/website/us/corporate/files/SDS_Safety_Architecture.pdf. [Accessed 28 01 2025].
[61] Mobileye, "True Redundancy," [Online]. Available: https://www.mobileye.com/technology/true-redundancy/. [Accessed 28 01 2025].
[62] S. Shalev-Shwartz, S. Shammah and A. Shashua, "On a Formal Model of Safe and Scalable Self-driving Cars," 2017. [Online]. Available: https://arxiv.org/pdf/1708.06374. [Accessed 28 01 2025].
[63] T. Wiltschko, "Mercedes-Benz DRIVE PILOT - A first step for automated driving and a giant leap for safety," in Keynote presented at Safetronic 2022, Stuttgart, Germany, 2022.
[64] M. Staron, Automotive Software Architectures: An Introduction, Springer Cham, 2021.
[65] ISO, "ISO/PAS 8926:2024 Road vehicles - Functional safety - Use of pre-existing software architectural elements," 2024.
[66] ELISA, "Enabling Linux in Safety Applications," [Online]. Available: https://elisa.tech/. [Accessed 08 04 2025].
[67] ISO, "ISO 24089 Road vehicles - Software update engineering," 2023.
[68] AUTOSAR, "Automotive Open System Architecture," [Online]. Available: https://www.autosar.org/. [Accessed 08 04 2025].
[69] UNECE, "UN Regulation No. 155 - Cyber security and cyber security management system," 2021.
[70] UNECE, "UN Regulation No. 156 - Software update and software update management system," 2021.
[71] ISO, "ISO/TS 5083 Road vehicles — Safety for automated driving systems — Design, verification and validation," 2025.
[72] ISO, "ISO/IEC TR 5469:2024 Artificial intelligence — Functional safety and AI systems," 2024.

[73] ISO, "ISO/PAS 8800:2024 Road vehicles — Safety and artificial intelligence," 2024.
[74] UNECE, "UN Regulation No. 157 - Automated Lane Keeping Systems (ALKS), Revision 1," 2023.
[75] UNECE, "UN Regulation No. 157 Amend.4," 03 March 2023. [Online]. Available: https://unece.org/transport/documents/2023/03/standards/un-regulation-no-157-amend4.
[76] ISO, "ISO/IEC 22989:2022 Information technology — Artificial intelligence — Artificial intelligence concepts and terminology," 2022.
[77] C. Hanselaar, M. Kumar, Y. Fu, A. Terechko, R. Prasad and E. Silvas, "Identification of Hazardous Driving Scenarios Using Cross-Channel Safety Performance Indicators," in 2025 Design, Automation & Test in Europe Conference (DATE), Lyon, France, 2025.
[78] M. Werling, R. Faller, W. Betz and D. Straub, Safety Integrity Framework for Automated Driving, arXiv preprint arXiv:2503.20544, 2025.
[79] ISO, "ISO/SAE PAS 22736:2021 Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," 2021.
[80] BSI, "BSI PAS 1883:2020 Operational Design Domain (ODD) taxonomy for an automated driving system (ADS) - Specification," 2020.
[81] Projekt Pegasus, "Projekt Pegasus," [Online]. Available: https://www.pegasusprojekt.de/en/pegasus-method.
[82] M. Scholtes, L. Westhofen, L. Turner, K. Lotto, M. Schuldes, H. Weber, N. Wagener, C. Neurohr, M. Bollmann, F. Körtke, J. Hiller, M. Hoss, J. Bock and L. Eckstein, "6-Layer Model for a Structured Description and Categorization of Urban Traffic and Environment," IEEE Access, vol. 9, pp. 59131 - 59147, 2021.
[83] B. Kaiser, B. Monajemi, D. Kusche and H. Schulte, "Systematic design and validation of degradation cascades for safety-relevant systems," in The 2nd International Conference on Engineering Sciences and Technologies, 2017.
[84] B. Frömel, "Fault Tolerance (lecture notes)," [Online]. Available: https://ti.tuwien.ac.at/cps/teaching/courses/cpsesfvo/slides-ws14/04_ft.pdf#:~:text=A%20Fault%20Containment%20Unit%20%28FCU%29%20is%20a%20set,and%20is%20assumed%20to%20fail%20independently%20from%20other.
[85] D. Powell, "Failure Mode Assumptions and Assumption Coverage," in Predictably Dependable Computing Systems, Springer, 1995, pp. 123-140.
[86] F. Cristian, "Understanding Fault-Tolerant Distributed Systems," Communications of the ACM, vol. 34, pp. 56-78, 1993.
[87] B. Littlewood and L. Strigini, "Validation of Ultrahigh Dependability for Software-Based Systems," Comm. ACM, vol. 36, pp. 69-80, 1993.
[88] H. Kopetz, Simplicity is Complex - Foundations of Cyber-Physical System Design, Springer Verlag, 2019.
[89] H. Kopetz, Real Time Systems - Design Principles for Distributed Embedded Applications, Springer Verlag, 2012.
[90] H. Kopetz, "Emergence in Cyber-Physical System-of-Systems," in Cyber-Physical System-of-Systems, Springer Verlag, 2016, pp. 73-96.
[91] A. Chou, J. Yang, B. Chelf, S. Hallem and D. Engler, "An Empirical Study of Operating System Errors," in Proceedings of the ACM SOPS 2001, 2001.
[92] Waymo, "Waymo Public Road Safety Performance Data," 2020.
[93] NHTSA, "Pre-Crash Scenario Typology for Crash Avoidance Research," 2007.
[94] American Psychological Association, "Multitasking Switching Cost," [Online]. Available: https://www.apa.org/research/action/multitask. [Accessed 22 May 2006].
[95] Automotive Electronics Council, "AEC-Q100:Rev-H Failure Mechanism Based Stress Test Qualification For Integrated Circuits," 2014.
[96] P. S. Shalev-Shwartz. [Online]. Available: https://www.youtube.com/watch?v=ViGL0z1BULs. [Accessed 22 08 2022].
[97] P. Koopman, "Simplified Proposal for Vehicle Automation Modes," 31 January 2022. [Online]. Available: https://safeautonomy.blogspot.com/2022/01/simplified-proposal-for-vehicle.html.

[98] California DMV, "Disengagement Reports," 2021. [Online]. Available: https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/disengagement-reports/. [Accessed 14 07 2023].
[99] Wikipedia, "Wikipedia entry on Heisenbug," [Online]. Available: https://en.wikipedia.org/wiki/Heisenbug. [Accessed 31 07 2023].
[100] S. Poledna, "Course: Dependable Computer Systems Part 5: Failure modes and models," 2007. [Online]. Available: https://ti.tuwien.ac.at/cps/teaching/courses/dependable_systems-ss08/dcs_slides/dcs-2007-p5.pdf. [Accessed 17 10 2023].
[101] N. Mohan, M. Törngren, V. Izosimov, V. Kaznov, P. Roos, J. Svahn, J. Gustavsson and D. Nesic, "Challenges in architecting fully automated driving; with an emphasis on heavy commercial vehicles," in Second International Workshop on Automotive Software Architectures (WASA), 2016.
[102] N. Mohan, P. Roos and J. Svahn, "System and Method for Controlling a Motor Vehicle to Drive Autonomously". Sweden Patent SE541390C2, 10 09 2019.
[103] N. Mohan, P. Roos and J. Svahn, "System and Method for Controlling a Motor Vehicle to Drive Autonomously". Germany Patent DE112018005794T5, 13 08 2020.
[104] N. Mohan, P. Roos and J. Svahn, "System and Method for Controlling a Motor Vehicle to Drive Autonomously". Worldwide Patent WO2019125268A1, 27 06 2019.
[105] Wikipedia, "Wikipedia entry on Daruma doll," [Online]. Available: https://en.wikipedia.org/wiki/Daruma_doll. [Accessed 24 07 2025].
[106] Y. Fu, J. Seemann, C. Hanselaar, T. Beurskens, A. Terechko, E. Silvas and W. P. M. H. Heemels, "Characterization and Mitigation of Insufficiencies In Automated Driving Systems," in 27th International Technical Conference on the Enhanced Safety of Vehicles (ESV), Yokohama, Japan, 2023.
[107] F. Brooks, "No Silver Bullet - Essence and Accident in Software Engineering," in Proceedings of the IFIP Tenth World Computing Conference, 1986.

LIST OF ABBREVIATIONS

Abbreviation	Meaning
ACM	Association for Computing Machinery
AD	Automated / Autonomous Driving
ADI	Automated Driving Intelligence
ADS	Automated Driving System
AES	Advanced Encryption Standard
AV	Automated Vehicle
BIST	Built-In Self-Test
CCDSS	Computer-Controlled Driving Sub-System (in channel-wise DCF architecture)
CCF	Common Cause Failure
CCI	Common Cause Initiator
CEHSS	Critical Event-Handling Sub-System (in channel-wise DCF architecture)
COTS	Commercial Off-The-Shelf
CSM	Controller Safety Mechanism (in DSM architecture)
DC	Diagnostic Coverage
DCF	Doer / Checker / Fallback
DDT	Dynamic Driving Task
DFA	Dependent Failure Analysis
DFI	Dependent Failure Initiator
DSM	Distributed Safety Mechanism
ECC	Elliptic Curve Cryptography
ECU	Electronic Control Unit
EOTI	Emergency Operation Time Interval
FCU	Fault-Containment Unit
FMEA	Failure Mode and Effects Analysis
FSM	Function Safety Monitor (in DSM architecture)
FTA	Fault Tree Analysis
FTDSS	Fault-Tolerant Decision Sub-System (in channel-wise DCF architecture)
FUN	Function (in the DSM architecture)
HARA	Hazard Analysis and Risk Assessment
HUD	Heads-Up Display
HW	Hardware
HWP	Highway Pilot
IEC	International Electrotechnical Commission

Abbreviation	Meaning
IEEE	Institute for Electrical and Electronics Engineers
IMU	Inertial measurement Unit
ISO	International Organization for Standardization
KPI	Key Performance Indicator
LOC	Lines of Code
MRC	Minimal Risk Condition
MRM	Minimal Risk Maneuver
MSS	Monitoring Sub-System (in channel-wise DCF architecture)
MTTF	Mean Time to Failure
NHTSA	National Highway Traffic Safety Administration
NIST	National Institute of Standards and Technology (US)
ODD	Operational Design Domain
OEDR	Object and Event Detection and Response
OEM	Original Equipment Manufacturer
OS	Operating System
PQC	Post-Quantum Cryptography
RSA	Rivest-Shamir-Adleman (cryptosystem)
SAE	Society of Automotive Engineers
SaRA	Safety-Related Availability
SEooC	Safety Element out of Context
SEU	Single-Event Upset
SoC	System-on-Chip
SOTIF	Safety of the Intended Functionality
SUV	Sports Utility Vehicle
SW	Software
TLS	Transport Layer Security
UI	User Interface
UNECE	United Nations Economic Commission for Europe
VRU	Vulnerable Road User
VSM	Vehicle Safety Mechanism (in the DSM architecture)
V2X	Vehicle-to-anything (vehicle, infrastructure)
WG	Working Group

APPENDICES

ODD OUTLINE OF REFERENCE AD USE CASE

FOLLOWED TAXONOMY

The HWP feature should only be active inside its defined Operational Design Domain (ODD), which is given by a set of conditions regarding its environment. We follow BSI PAS 1883:2020 [80], which mostly covers the same attributes as the formalism (6 layers) of Project Pegasus [81] [82]. The HWP feature should refuse to activate if these are not met and should, if these are no longer met, prompt the driver to take back control within a convenient time span and in the meantime ensure safety, e.g., by bringing the vehicle to a safe stop. The ability to execute an MRM must be maintained even outside the ODD.

SCENERY

Zones

Attribute	Sub-attribute (1)	Sub-attribute (2)	Capability
Zones	Geo-fenced areas		Yes, as designated by OEM
	Traffic management zones		No
	School zones		No
	Regions or states		Yes, as designated by OEM
	Interference zones	Dense foliage	Yes (not close to driving path)
		Tall buildings	Yes

DRIVABLE AREA

Attribute	Sub-attribute (1)	Sub-attribute (2)	Sub-attribute (3)	Capability
Drivable area	Drivable area type	Motorways (high-ways)		Yes, maximum 130 km/h
		Radial roads		No
		Distributor roads		No
		Minor roads		No
		Slip roads		No
		Parking		No
		Shared space		No
	Drivable area geometry	Horizontal plane	Straight roads	Yes
			Curves	Yes, maximum 1/100 m
		Transverse plane (cross-section)	Divided / undivi-ded	Divided
			Pavement	No
			Barrier on the edge	
			Types of lanes to-gether	
		Longitudinal pla-ne (vertical)	Up-slope	Yes, maximum +4%
			Down-slope	Yes, maximum -4%
			Level plane	Yes
	Lane specifica-tion	Lane dimensions		Minimum 3.5 m
		Lane marking		Yes, in good condition
		Lane type	Bus lane	No (may be present, but must not be used during nor-mal operation)
			Traffic lane	Yes
			Cycle lane	No
			Tram lane	No
			Emergency lane	No (may be present, but must not be used during nor-mal operation)
			Other special purpose lane	Yes, carpool lanes
		Number of lanes		Yes, minimum 2 lanes per di-rection
		Direction of travel	Right-hand traffic	Yes
			Left-hand traffic	No

Attribute	Sub-attribute (1)	Sub-attribute (2)	Sub-attribute (3)	Capability
Drivable area	Drivable area signs	Information	Variable	Yes, full-time and temporary
			Uniform	Yes, full-time and temporary
		Regulatory	Variable	Yes, full-time and temporary
			Uniform	Yes, full-time and temporary
		Warning	Variable	Yes, full-time and temporary
			Uniform	Yes, full-time and temporary
	Drivable area edge	Line markers		Yes
		Shoulder (paved or gravel)		Yes
		Shoulder (grass)		Yes
		Solid barriers		Yes, obligatory on left side
		Temporary line markers		No
		None		No
	Drivable area surface	Surface type	Asphalt	Yes
			Concrete	Yes
			Cobblestone	No
			Gravel	No
			Granite setts	No
		Surface features	Cracks	Yes, minor only
			Potholes	No, not in significant density
			Ruts or swells	Yes, minor only
			Damage caused by weather	Yes, minor only
			Damage caused by traffic	Yes, minor only
		Induced conditi-ons	Icy	No, not to a significant extent
			Flooded	No
			Mirage	Yes
			Snow	No
			Standing water	No
			Wet	Yes
			Contaminated	Yes, minor only

Additional assumptions:

- Changed road markings or reduced lane width are not supported.
- The speed limit is appropriate for the curve radius and slope of the road such that the entire stopping distance is visible without occlusions (in the absence of other vehicles).

JUNCTIONS

Attribute	Sub-attribute (1)	Sub-attribute (2)	Sub-attribute (3)	Capability
Junctions	Roundabout			No
	Intersection	T-junction		No
		Staggered		No
		Y-junction	On-ramp and off-ramp	No (except driving by)
			Other	No
		Crossroads		No
		Grade-separated	Interchange	No
			Other	No

ROAD STRUCTURES

Attribute	Sub-attribute (1)	Capability
Special structures	Automatic access control	No
	Bridges	Yes
	Pedestrian crossings	No
	Rail crossings	No
	Tunnels	Yes, with separate driving directions
	Toll plaza	No
Fixed road structures	Buildings	No
	Streetlights	Yes, but not required
	Street furniture	No
	Vegetation	No
Temporary road structures	Construction site detours	No
	Refuse collection	No
	Road works	No
	Road signage	No

ENVIRONMENTAL CONDITIONS

Attribute	Sub-attribute (1)	Sub-attribute (2)	Capability
Weather	Wind	Calm - fresh breeze (<10.7 m/s)	Yes
		Strong breeze (>10.7 m/s) - hurricane force	No
	Rainfall	Light rain (<2.5 mm/h)	Yes
		Moderate rain (>2.5 mm/h) - cloudburst	No
	Snowfall	Light snow (>1 km visibility)	Yes
		Moderate snow (<1 km visibility) - heavy snow	No
Particulates	Marine		No, not to significantly reduced visibility
	Mist and fog		No, not to significantly reduced visibility
	Sand and dust		No, not to significantly reduced visibility
	Smoke and pollution		No, not to significantly reduced visibility
	Volcanic ash		No, not to significantly reduced visibility
Illumination	Day		Yes
	Night or low-ambient		No, not to significantly reduced illumination
	Cloudiness	Clear - overcast	Yes
	Artificial illumination		Yes
Connectivity	Communication	V2V, V2I	Yes, at least intermittently
		Cellular	Yes, at least intermittently
		Satellite	No
		DSRC and ITS-G5	No
	Positioning	Galileo	Yes, at least intermittently
		GLONASS	Yes, at least intermittently
		GPS	Yes, at least intermittently

Additional assumptions:

- Not being warned of major road or traffic conditions is uncommon. We assume that the road layout is known ahead of time and that unexpectedly encountering challenging road or traffic conditions is uncommon as authorities oversee keeping the road in an acceptable state of repair and/or informing traffic participants (via signs, map data, and/or V2X) if this is not the case.
- HD maps are available for all supported highway segments.

DYNAMIC ELEMENTS

Attribute	Sub-attribute (1)	Sub-attribute (2)	Capability
Traffic	Density of agents	Dense traffic (including stop & go)	Yes
		Free-flow traffic (including no lead vehicle)	Yes
	Volume of traffic		
	Flow rate		
	Agent type	Cars	Yes
		Buses and trucks	Yes
		Motorbikes	Yes
		VRUs (pedestrians, bicyclists)	Yes, to a very limited degree
		Animals	Yes, to a very limited degree
		Minor static obstacles (lost load, debris, etc.)	Yes
		Major static obstacles (lost load, trees, rocks, etc.)	Yes, to a very limited degree
		Special vehicles	Yes
Subject vehicle (ego vehicle)	Behavior capabilities	Ego vehicle speed	0-130 km/h
		Lane change	Yes
		Lane merge	Yes
	Vehicle	All sensors and actuators in working condition	Yes
		Sensor or actuator non-operational	No, except during MRM
		Superficial body damage	Yes
		Moderate - major body damage	No
		Door or window open	No
		Low fuel or charge level	No
	Passengers	Driver not in driver seat	No
		Unbelted passenger	No
		Driver asleep or incapacitated	No

Additional assumptions:

- All human traffic participants are aware that the highway is a restricted environment and act accordingly (responsibly).

CLASSIFICATION OF TRAJECTORY CAPABILITY

Almost all conceptual system architectures discussed in this report have several subsystems capable of generating trajectories and controlling the vehicle. However, these subsystems can have vastly different capabilities, i.e., the generated trajectories can differ significantly in how well they avoid obstacles, ensure comfort, etc.

To judge whether using the outputs of a particular subsystem is acceptable in a certain scenario of faults and/or functional insufficiencies (similar to the concept of a degradation cascade [83]), we define six levels of trajectories. These are listed below (see also summary in Table 29) in descending order of capability and used as reference in the evaluation of the availability of the system (see section 2.3.1 and respective generic evaluations in section 4.2).

1. A nominal trajectory can only be generated by very complex subsystems. Such trajectories dynamically react to traffic and also ensure comfort and efficiency. To plan these, input from many different sensors and off-board information (e.g., HD maps, V2X, navigation data) are needed. These are updated continuously, e.g., every 40-50 ms. If there are no faults or functional insufficiencies in any subsystem, this is the expected output of the ADI. Such output should be considered generally safe.
2. A degraded trajectory resembles a nominal trajectory but applies some restrictions. Instead of continuing the mission, such trajectories either reduce speed (e.g., from 130 km/h to 90 km/h) or aim to come to a comfortable stop in a permanently safe location (e.g., at a rest stop); they are updated continuously, e.g., every 40-50 ms. In high-speed AD use cases, the ADI may only output degraded trajectories if there is a fault or functional insufficiency in at least one subsystem (also latent faults and/or sensor or actuator faults). If the issue can be resolved, it is beneficial to restore nominal functionality. Such output should be considered generally safe.
3. An MRM trajectory only aims to bring the vehicle to a controlled stop in an adequately safe location[66]. To plan such trajectories, input from an adequate sensor set is needed. The trajectories are updated continuously, e.g., every 40-50 ms. In high-speed AD use cases, the ADI may only output MRM trajectories if there is a fault or functional insufficiency in at least one subsystem. Once an MRM has been started, switching back to nominal or degraded mode is usually prevented. Such output should be considered generally safe.
4. A pre-planned MRM trajectory resembles an MRM trajectory but differs in planning horizon and may restrict lane changes (e.g., only to the emergency lane). Most trajectories are planned with a time horizon of at most a few seconds, but it may take dozens of seconds to come to a complete standstill in high-speed use cases. Pre-planned trajectories keep getting updated regularly and are buffered close to the actuators. In high-speed AD use cases, the ADI may only resort to a pre-planned (buffered) MRM if there are independent, simultaneous faults or functional insufficiencies in at least two subsystems. Such an MRM is executed without further dynamic updates ("blindly"), i.e., it cannot react dynamically to traffic anymore once the generating subsystem is lost. Such output should be considered potentially unsafe.
5. A blind braking trajectory is a basic residual reaction. It only aims to maintain the current curvature (or ideally the current lane) and apply a constant deceleration. To "plan" such trajectories, no sensor data or significant processing power is needed. In high-speed AD use cases, the AD system may only resort to blind braking if there are independent, simultaneous faults or functional insufficiencies in at least two subsystems. Such output should

[66] This depends on the considered AD use case. For a Valet Parking feature, it may be acceptable for stop anywhere. For a Highway Pilot feature, it may only be acceptable to pull over to the emergency (or right-most) lane; stopping in the current lane may only be acceptable under very rare circumstances.

be considered probably unsafe.

6. No output or a faulty trajectory represent the worst case (i.e., the acceptable residual risk). In high-speed AD use cases, the ADI may only remain silent or output faulty trajectories if there are independent, simultaneous faults or functional insufficiencies in at least two sub-systems. Such output should be considered generally unsafe.

Table 29: Levels of trajectories in descending order of capability.

Trajectory	Needed sensors	Dynamics and implied safety	Comfort, efficiency, and utility
Nominal	Yes (many)	High	High
Degraded	Yes (many)	High	Medium-High
MRM	Yes (few)	High	Medium
Pre-planned MRM	Yes (before execution starts) No (after execution starts)	Medium	Medium
Blind braking	No	Low	Low
None/faulty	No	None	None

For the evaluation, we use the following scheme:

- In the nominal case, i.e., without faults or functional insufficiencies, the AD system needs to execute a nominal trajectory (1). Lower capabilities (2-5) can be problematic, though not necessarily unsafe.
- In cases with a single fault or functional insufficiency, i.e., affecting a single FCU, the AD system needs to execute an MRM that is dynamically adapted to the traffic situation in real time (3). Of course, higher capability is "nice to have" (1-2). Lower capabilities (4-5) can be problematic, though not necessarily unsafe.
- In cases with two independent faults or functional insufficiencies, i.e., affecting two FCUs simultaneously, any remaining capability is acceptable (1-5).

SAMPLE ANALYSIS POINTS REGARDING DIFFERENT CONCEPTUAL ARCHITECTURE PATTERNS

This appendix provides additional details on the design principle D7: Mitigation of common cause hazards, section 1.5.2.

The dimensions introduced in Figure 6 can be exemplified by conceptually applying them to a selection of architecture patterns. Functional complexity is indicated by the depth of the slice, while the implemented capabilities' coverage of an operational domain is indicated by the surface area. Holes are therefore representative of an absence of capability.

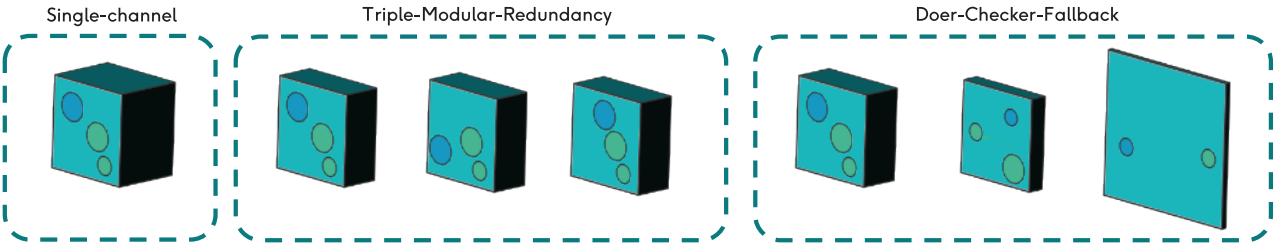


Figure 64: Channel dimensioning as per conceptual pattern of selected architecture candidates

Given the selected architecture patterns in Figure 64, the following relative observations can be made:

- Single-Channel architectures can be expected to have significantly complex implementations to meet the functional requirements of advanced use cases.
  - Errors and output insufficiency within the implementation is not complemented or offset by the capability of another channel; a fallback capability is not available.
- Triple Modular Redundancy (TMR) architectures typically offer redundancy of the same functionality, hence the equal depth of each slice and diverse location of errors to prevent unavailability of the function.
  - Nevertheless, the redundant functions may all contain the same output insufficiencies and therefore offer no prevention of common cause functional insufficiency hazards.
  - Even the diverse implementation of the functionality aiming to achieve non-common cause output insufficiencies could struggle with inexact agreement when handling the individual channel outputs.
  - A fallback capability beyond the nominal functionality within the intended ODD is not available, as indicated by the equal areas of each slice.
- Doer/Checker/Fallback architectures typically propose the diverse implementation of a complex performance channel and its complementary checker channel, hence the unequal depth of the slices and diverse location of holes.
  - A fallback capability beyond the nominal functionality is offered by a basic channel capable of offering minimum-risk-maneuver in conditions beyond the ODD; the area of which could be considered analogous to something like the Target Operational Domain (TOD).

THE | AUTONOMOUS