Project acronym:	*SafeAdapt*
Project title:	*Safe Adaptive Software for Fully Electric Vehicles*
Grant Agreement number:	608945
Coordinator:	*Dr.-Ing. Dirk Eilers*
Funding Scheme:	*FP7-2013-ICT-GC*

Deliverable 3.1

[Concept for Enforcing Safe Adaptation during Runtime]

Due date of deliverable:	31.12.2014
Actual submission Date:	06.03.2015
Lead beneficiary for this deliverable:	Fraunhofer ESK

Dissemination level		
PU	**Public**	**X**
PP	**Restricted to other programme participants (including the Commission Services)**	
RE	**Restricted to a group specified by the consortium (including the Commission Services)**	
CO	**Confidential, only for members of the consortium (including the Commission Services)**	

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013)

Document Information	
Title	Concept for Enforcing Safe Adaptation during Runtime
Creator	Fraunhofer: Philipp Schleiss, Christian Drabek, Gereon Weiss
Description	The document contains a detailed description of fundamental system properties, required design information, and runtime mechanisms needed to perform adaptation in order to operate electric vehicles safely and efficiently.
Publisher	Fraunhofer ESK
Contributors	CEA: Ansgar Rademacher, Önder Gürcan Delphi: Timo Feismann, Thorsten Rosenthal Duracar: Ken Lam Siemens: Cornel Klein Tecnalia: Alejandra Ruiz, Mª Carmen Palacios, Garazi Juez TTTech: Andreas Eckel
Language	en-GB
Creation date	13.8.2014
Version number	1.0
Version date	06.03.2015
Audience	☐ internal ☒ public ☐ restricted

Table of Contents

List of Figures

List of Tables

Glossary

Term	Definition
Aggregate	A sensor or actuator (may be directly connected to the Ethernet network or utilise an intermediary gateway node to gain access to the network)
Application	A single-threaded piece of software, contributing to a functionality of a vehicle (Applications participate in adaptation, thus being able to be instantiated on an arbitrary core node)
Core network	A set of nodes that communicated in the time-triggered traffic class
Core node	A platform used to host application that participate in adaptation
Feature	A combination application performing a certain functionality of a vehicle
Gateway node	Device to translate Ethernet traffic into a sensor-, actor-, or subsystem-specific protocol (gateways may implement rate-constrained or time-triggered traffic classes)
Node	Any device participating within the rate-constrained and time-triggered Ethernet network (e.g. gateways nodes, switches, core nodes, aggregates)
Platform	Combination of hardware and platform software (e.g. OS, drivers, diagnostics, …) without application software
Rate-constraint	Deterministic communication protocol (see time-triggered)
SAPC	Safe Adaptation Platform Core – A piece of software instantiated on every core node to control the local runtime reconfiguration
Task	The runtime instantiation of an application
Time-triggered	Highly deterministic communication protocol utilising virtual links to improve maximal latencies as compared to rate-constraint traffic
Virtual link	Reserved 1:n connection between nodes in the core network, guaranteeing a fixed payload to be transferred in within a predefined time-interval

Executive Summary

In light of the diminishing natural resources, innovation in the field of transportation systems is inevitable to safeguard the high standards of modern mobility. In this context, fully electric vehicles pose as a promising solution to enable sustainable forms of transportation while simultaneously improving the ecological footprint of the automotive industry. Despite this, new challenges with respect to safety and cost efficiency arise when pursuing to penetrate the market with fully electric vehicles. For instance, the replacement of mechanical solutions with electronic control systems demands for additional backup units in order to mitigate the risk of failure and prevent potentially disastrous situations. Considering the additional cost and weight of such dedicated backup units, the initially expected benefits of e-vehicles are however likely to be curtailed, thus questioning their economic feasibility.

To resolve this dilemma, this document provides a concept for safely redefining the responsibilities of control units during operation, thus enabling a resource efficient use of electronic hardware to handle failure situations. Building upon this failure handling concept, more advanced forms of adaptation are developed, e. g., to allow a vehicle's range to be extended through the deactivation of non-critical features.

In detail, a classification scheme is derived to capture and store the varying data dependencies, timing properties, and failover requirements of a vehicle's electronically controlled functionality. Subsequently, this formalised information is utilised to determine the feasibility of runtime configurations without the need for manual verification. Moreover, to ensure the correct operation of the runtime adaptation system, mandatory system properties are defined in which adaptation is guaranteed to function safely. For this, two deterministic and reconfigurable communication links, a detached and time-triggered computational model, unified system interfaces, and a generic fault classification scheme are specified as minimal requirements. Based on these functional requirements and architectural properties, a runtime concept is developed to enable the reconfiguration of safety-critical functionality at runtime, which is in turn grounded in a theoretical adaptation framework. More specially, a control unit's local and global fault detection techniques are unified to support the safe interaction and reconfiguration of dissimilar control units.

In essence, the document forms a sound conceptual blue print for the subsequent specification and implementation of an efficient and fault-tolerant runtime system to enable the economic and ecologically friendly advance of fully electronic vehicles.

1 Introduction

The interest in Fully Electric Vehicles (FEV) and autonomous driving has gained tremendous momentum in the last years. To benefit from their potential, electronically controlled vehicle features, such as Steer-by-Wire, are inevitable in order to maintain a manageable level of system complexity and achieve a reduced bill of materials. However, such electronically controlled features cannot always rely on mechanical fall-back solutions, and hence require the electronic control system to remain operational after a failure. For this, the following document contains a concept for performing adaptation during runtime as a mean to enforce safety of a vehicle's control system in a cost-efficient manner.

Adaptation during runtime can help to achieve the necessary level of fault-tolerance to ensure safe operations. In SafeAdapt, adaptation will be achieved with the help of the Safe Adaptation Platform Core (SAPC). The SAPC is software that is hosted on different computing platforms, which in turn are commonly referred to as core nodes. Each instance of the SAPC controls the adaptation for its local core node, but also guarantees to reach a correct system configuration with respect to all other instances of the SAPC. It is important that the SAPC follows safety standards and limits its impact on the effort for designing the resulting system. For this, each SAPC regularly sends information about the current state of its platform to the other SAPCs. When combining this volatile information with the predefined requirements of the vehicle, the SAPC is capable of calculating a configuration to address the newly occurred need for adaptation.

The goal of this document is to provide a concept for the implementation of the Safe Adaptation Platform Core based on the use cases and requirements defined in the previous deliverables 2.1 and 2.2. Building upon these conceptual results, deliverable 3.2 will provide a specification of how the SAPC is implemented. Moreover, this concept provides a target description that the modelling methods and tool of work package 4 must address. In the following chapter, the use cases and requirements are broken down to goals that the SAPC's implementation must fulfil. Thereafter, the third chapter defines the fundamental properties that a system must provide, if based on the SAPC concept. The fourth chapter gives an overview of the safety states and their corresponding runtime mechanisms from the perspective on an application. Following this, chapter 5 introduces the main strategy of how safety and optimisation are achieved during adaptation, thus resulting in chapter 6 describing this strategy through design and runtime concepts. The final chapter summarises the introduced concepts.

2 Goals

In order to guarantee predictable system-level behaviour in reconfiguration scenarios, techniques for controlling the self-adaptation of the system during runtime are needed. These techniques must preserve the design time specification defining the system's dynamic behaviour. For this, the runtime control system must respect predefined constraints concerning the allowed degree of adaptation to guarantee the quality of safety-relevant applications and prevent unwanted or uncontrolled behaviour. In turn, this allows the system to react on unforeseen situations and guarantee the predefined quality of safety-critical applications.

2.1 Safety

Safety is one of the most relevant issues for most automotive features and hence a major motivation for the SafeAdapt project. "ISO 26262 Road vehicles – Functional safety" is the standard that will lead the safety requirements for reconfiguration in SafeAdapt. The SAPC is designed to passivate less critical features in order to maintain the most critical ones in every possible scenario. The ISO 26262 provides an automotive-specific risk-based approach to determine integrity levels (Automotive Safety Integrity Levels – ASIL) of the features to be installed in a vehicle. These ASIL levels (D, C, B, A, QM) are used as the basis for determining the safety goals of each application as part of the hazard and risk analysis, which are then used to determine the reconfiguration requirements of each application. In addition, the proper operation of some features depends on correct interoperation between all applications of that feature. Therefore, the SAPC must also take the constraints of dependant applications into account when activating or passivating applications.

Moreover, fault-tolerance must be achieved by SAPC-internal or external safety mechanisms in order to control or mitigate faults. External mechanisms include all platform- and hardware-dependent methods to mask faults, whereas internal mechanisms utilise the SAPC's reconfiguration concept to cope with the occurred fault. For the latter, multiple safety use cases have been defined in deliverable 2.1 and refined in form of requirements in deliverable 2.2. The following use cases are further described from the perspective of an application's safety goals in chapter 4:

- Hot-standby (see use cases UC_116_01 & UC_111_01 in deliverable 2.1)
- Cold-standby (see use case UC_110_01 in deliverable 2.1)
- Degradation of application (see use case UC_411_01 in deliverable 2.1)
- Adherence to a feature's internal dependencies (see use case UC_114_01 in deliverable 2.1)

2.2 Energy Efficiency

In battery-driven FEVs, the energy consumption is even more critical than in vehicles with combustion engines, because the electric energy also powers the vehicle's drivetrain. As the SafeAdapt Platform Core is designed to be used in FEVs, the support for improving energy efficiency is one of the SAPC's major design elements. Furthermore, the need for energy efficiency

is motivated in use case UC_511_01 of deliverable 2.1, which applies adaptation for extending the vehicle's range. The concept of safe adaptation helps to make FEVs more energy efficient through a better resource utilisation. Safe adaptation allows using redundancy mechanisms other than additional hardware. By reducing the number of nodes, the weight of the vehicle is reduced, thus also lowering the overall energy required for reaching and maintaining momentum.

Regardless of energy concerns, the adaptation rules of the SAPC must treat safety properties of a feature as the most relevant description criteria when performing a reconfiguration. However, as long as safety requirements are appropriately addressed, energy efficiency optimisation criteria can be used in a second step to decide on how to perform reconfiguration. For example, keeping a required hot-standby instance operational should always precede switching it off for the sake of saving energy. Then again, when multiple configurations are available, the SAPC should consider using the configuration that is most energy efficient according to the data available. Furthermore, energy efficiency must be considered in cases when energy levels can produce a foreseeable safety hazard.

In addition, the SAPC should be designed so that it can also be used to support the smart use of available CPU resources. For instance, safe adaptation can facilitate disabling not needed features and enabling other system optimisations during runtime to save additional energy, for instance by shutting down a presently not needed node.

2.3 Maintainability, Extensibility & Easy Integration

Maintainability is an important factor when creating new software systems to guarantee proper operation of the system even after several years. It becomes even more challenging when different product lines and extensions of the system have to be considered. Adaptation can easily introduce more configuration complexity into any system as the number of configurations that have to be considered grows. However, adaptation mechanisms can enable the generic handling of failures, extensions, and updates, as, for instance, described in use case UC_211_01 and UC_311_01 of deliverable 2.1. One objective of SafeAdapt is to reduce the development and certification costs for software by up to 20%, as failure handling and extendibility requires considerable effort for systems with high safety requirements. In particular, failure handling will be simplified as it is handled in a generic way, which also enables easier reuse and migration of applications.

In essence, the SAPC addresses the problem of integrating multiple features through a modular approach allowing critical and non-critical applications to be executed on the same node. Often, systems depend on each other in cases of failure. For instance, a driver drowsiness detection system requires monitoring of the steering wheel activity. In case of a failure within the steering wheel system, a redundant fall-back system will still provide the needed functionality, but in turn requires the drowsiness detection to also adapt to the newly activated fall-back system. By providing a generic and interoperable failure handling mechanism, these direct dependencies between applications are respected. This significantly reduces development, testing, and certification costs. Furthermore, applications can be designed independently from platform-specific failure handling, which improves reusability.

In more detail, SafeAdapt will propose a re-designed electric/electronic (E/E) architecture for FEVs to provide efficient, safe, and cost effective robustness based on safe runtime adaptation. The architecture will reduce the complexity of the system and the interactions through communication channels by including several unrelated applications in a single computing platform. Consequently, this will lead to reduced costs, assembly, and maintainability efforts. The approach is based on existing multi-domain controllers, as developed in current research activities. As such, the two integrated hardware platforms RACE (Robust and Reliable Automotive Computing Environment for Future eCars) and TMDP (Trusted Multi Domain Platform) are used. Both platforms are designed to consolidate applications throughout different in-vehicle domains. To enable a standardised information exchange between these two platforms, Time-Triggered Ethernet (TTE) is used as a dependable and real-time inter-platform communication channel.

2.4 Relationship to other EU-Projects

SafeAdapt is well embedded in the context of related European projects. Within the FP 7 project SCARLETT, aerospace grade strategies were followed to allow a safe reconfiguration of an aerospace control system during operational conditions. The system under consideration used AFDX, CAN-bus, and FlexRay as a network topology, which was in part also used to transmit safety-critical data in the aircraft. The systems operating with safety-critical data relied on hot-standby, triple-redundant, dissimilarly designed architectures. The fault detection mechanisms were based on permanent voting between triple-redundantly calculated results. In case the voter detected that the active node controlling the application delivered results deviating from the result of the two redundantly running nodes, the active node was considered faulty and was immediately replaced by one of the two redundant nodes using configurations that were stored in a look-up table. SafeAdapt makes use of the knowledge and results gained in SCARLETT. However, the results in SCARLETT used aerospace grade designs which cannot directly be transferred to an automotive grade application. Even if such an application is highly safety-relevant, hot triple-redundant and dissimilar designed systems are far beyond realistic cost for the automotive industry.

Moreover, the FP 7 project DREAMS, which focusses on the integration of application with varying degrees of criticality onto one computing platform, also utilises the concept of look-up tables to achieve fault-tolerance. For this, application code is stored on potential backup nodes, thus allowing a rapid activation of the application's backup instance in case the node hosting the primary instance fails. In SafeAdapt, the complementary concept of an application repository is introduced, allowing an on-demand transfer of application code to the respective node. Based on this, the original application image or a degraded version thereof can be potentially activated on any node, which in turn requires a reconfiguration of the communication scheme. To determine the feasibility of this approach with respect to temporal failover limit (e.g. 10 ms), the following question should be answered:

- How do code size and memory types affect the time required to initialise an application from a software repository?
- Can reconfiguration by means of an application repository be performed deterministically?

3 Fundamental System Properties

The Safe Adaptation Platform Core (SAPC) is software that is designed to run with different operating systems on multiple types of hardware platforms. It is executed on all core platforms and calculates a new local configuration after the need for adaptation was detected. This local configuration is consistent with the local configurations that were determined by the other SAPC instances, thus forming a new configuration for the entire system. Moreover, the SAPC can only operate correctly as long as its hardware and software environment remains within the assumed design boundaries. Only then, it is possible for the SAPC to calculate and perform the required steps needed during its operation. In essence, the following chapter gives an overview of properties defining that design space.

3.1 System Architecture

The main goal of the SafeAdapt project is to develop a mechanism for safe adaptation and not to design a completely new hardware architecture. Therefore, the hardware architecture of the RACE car is reused where possible, or otherwise modified to support safe adaptation. Figure 1 shows the intended concept of the hardware architecture as far as it is relevant for the SAPC. The figure uses boxes for gateway nodes, round cornered boxes for core nodes, and circles for switches (see glossary). Gateway nodes provide access to aggregates, such as sensors and actuators, which are not designed to be directly connected to an Ethernet network. To ensure the required level of fault-tolerance, the power supply of the nodes has to be designed in a way that the loss of one power line still allows a safe operation. This is depicted through red and blue coloured borders denoting the two separate power circuits used in the RACE car. Moreover, the original RACE car architecture uses multiple overlapping rings to connect the nodes in the system. It is planned to replace the inner ring (core network) containing the main computing platforms (core nodes) of the RACE vehicle with a Time-Triggered Ethernet (TTE) solution. The wiring of the outer rings with the gateway nodes can therefore remain unchanged.

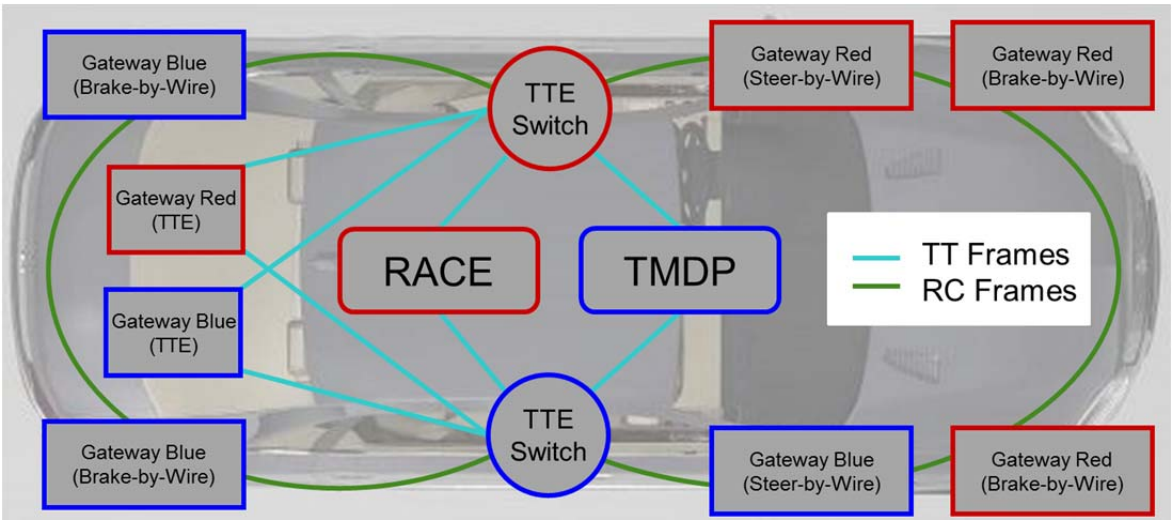

Figure 1: Overview of Hardware Architecture

In detail, the system consists of three kinds of nodes: core nodes participate in the safe adaptation of their software applications, whereas gateway nodes provide access to the sensors and actuators of the car. Both utilise switch nodes to interconnect and exchange information. More precisely, multiple aggregates are often represented by the same gateway node, as currently available automotive devices do not yet provide the needed Ethernet connectivity and therefore require an intermediary translation step performed by such gateways.

To prevent a fault in the communication infrastructure to cause a safety hazard, a redundant path between all safety-relevant nodes is required. For this, two switches are used in a double-star configuration. This ensures connectivity even if one switch or network link should fail. Less critical nodes that do not require redundancy can also be connected to only one switch. Regardless, all core nodes are directly connected to both switches using time-triggered traffic model (TT). To reuse the already existing infrastructure of the RACE vehicle, gateway nodes may be connected using time-triggered or rate constrained (RC) traffic (see Figure 1). Rate constraint traffic allows the reservation of bandwidth to ensure that communication requirements are always met. In comparison, time-triggered traffic improves this, by reducing the maximal communication latency, thus enabling highly responsive and predictable communication.

Moreover, the TMDP and RACE platform both build on different hardware, operating systems, and methods of fault detection. To benefit from a platform-independent SAPC, both platforms must implement a platform-specific extension to enable the communication with the unified interfaces of the SAPC. Similarly, to allow the execution of the same application on both platforms, AUTOSAR's RTE concept is utilised, thereby enabling the platform-independent development of applications. Furthermore, the operating system must provide real time capabilities to strictly ensure each feature's timing requirements.

3.2 Globally Synchronous Execution

The core network consists of core nodes and switches which communicate in a time-triggered manner. This allows for a synchronous communication between the nodes, as one node sends data exactly when the other node expects to receive the data, thus creating a globally synchronous execution pattern. In turn, this facilitates the use of a time-triggered scheduling paradigm on each core node. Applications can be scheduled to finish just in time when the communication slot is available. The globally synchronous execution and the thereby shared global time base are also helpful for adaptation. They guarantee that at the same point in time all core nodes will exchange their status information and perform the required measures. Moreover, to accommodate the use of rate-constraint traffic in the outer network rings of the vehicle, TTE switches are used to map this traffic onto predefined TT virtual-links.

3.3 Local Fault Detection

In general, adaptation is used to react to faults, mode change requests, and other environmental influences. To enable a failure recovery concept based on adaptation, the SAPC requires each platform to detect its local faults. Both the TMDP and the RACE platform support different kinds of fault detection mechanisms, such as, hardware- and software-based lock stepping, memory checks, and other sanity checks. The minimal fault detection capability of each platform will be subject to a fault-tolerance analysis, which is described in general in section 5.3 and will be conducted as part of deliverable 3.2.

From an abstract view, faults are typically associated with the time or the value domain, respectively meaning, that a computational event does not occur in the expected timeframe or that data is corrupted. In the time domain, faults can often be detected in software, as application specific timing requirements are used to identify deviations. Within the value domain, fundamental and context specific faults are considered. In general, fundamental faults in the value domain, such as bit-deviations caught by lockstep mechanisms, incorrect CRCs, ECC faults, overheating, power fluctuations, short circuits, or other limited physical damages can often already be identified in hardware. They are then propagated to the software through special purpose CPU-specific interrupts or otherwise directly mitigated within the hardware. However, context specific faults in the value domain are, similar to timing faults, often only detectable within the software layers of a device.

3.4 Generic Network Scheduling

When executing an application on a different core node the system must guarantee that the inputs and outputs of the application are still received and sent correctly. For this, all messages sent in the core network are broadcasted to all core nodes within the core network. This only guarantees the general accessibility, but not yet the timeliness in which messages are transferred. However, the core network provides time-triggered traffic between all core nodes. Specific time slots can be reserved to guarantee the delivery of messages at a certain point in time. Time-Triggered Ethernet (TTE) does not yet support dynamic reconfiguration during runtime, wherefore a different approach is used instead. The network resources are split into generic slots with a fixed network packet size. The slots are then distributed amongst all core nodes according to round robin algorithm during the design phase. Through this equal distribution of communication resources, every core node is capable of communicating a guaranteed amount of messages. This amount is dependent on the network's bandwidth, the chosen packet size, and the amount of core nodes, as described by the following formula:

$$\frac{Messages}{ms} = \left\lfloor \frac{Bandwidth\ in\ \frac{MB}{s} * 1000}{Core\ Nodes * Packet\ Size} \right\rfloor$$

For instance, choosing a packet size of 200 byte in a 100 Mbit/s (12 MB/s) network with 4 core nodes, each core node can send 15 messages of up to 200 bytes per millisecond. This is a more than sufficient value to allow the system to function correctly.

3.5 System Guarantees

The derivation of the correct fault cause from an observed effect is not always straight forward. For instance, in a purely fixed priority based system with no additional guarantees, an observed failure in the timing domain is often not caused by the application breaching its timing requirements, but caused by other tasks or by the accumulation of unexpected interrupts. Consequently, to be able to better map observations to causes, strict system design principles are applied to limit the amount of potential fault sources. More precisely, a set of properties was developed which the system must fulfil at any time to remain within the predefined laws of operation:

- The platforms **strict partitioning, bounding of interrupts and OS overhead, and global time synchronisation** guarantees that the application breaching its WCET is the actual cause of the delay (this does not apply to best effort tasks).
- **End-to-end data consistency protection** of the platform and communication network guarantees that false data was produced by the previous entity and was not corrupted during transportation.
- The **synchronous communication model** and **redundant communication infrastructure** guarantees that a missing message was caused by the passivation or failure of the remote core node.
- As **gateways** feed data into the core network in a time-triggered manner, it is guaranteed that no other data is simultaneously scheduled for transfer, as long as a **conflict-free and time-triggered schedule** was defined for every **gateway** during the design phase.
- The **Safety Element out of Context (SEooC)** concept is enforced through unbreachable time and memory borders, thus guaranteeing correct execution of an application, if it passes scheduling and allocation tests.
- The use of **generic communication slots** (see section 3.4) with packet sizes statically defined during design time and a message **broadcasting** system between all core nodes guarantees that all exchanged data is available on every core node and that each core node can transmit a minimal amount messages in a certain time frame.

4 Safety States of Applications

Beside a hazard analysis and risk assessment, the ISO 26262 requires a so called safety goal that has to be defined for each application. For this reason, the Safe Adaptation Platform Core has to know about the safety goals of the applications. Each application can have different options for degradation, a varying impact on safety, and different requirements on its availability and timeliness. These safety goals dictate how the application must be handled during adaptation. This chapter describes the different safety states (Fail-Safe, Fail Operation, and Fail-QM), which are used to achieve the safety goals, from the perspective of an application. More precisely, safety functions used to achieve each safety state are outlined in form of runtime mechanisms, which will be either implemented as part of the SafeAdapt vehicle or evaluated through simulation. Further, a hazard and risk analysis of these safety functions, which are represented by the SAPC's runtime adaptation mechanisms, will be performed in deliverable 3.3, to determine the safety goals of this form of adaptation.

4.1 Fail-Safe & Fail-Soft (Degradation)

A *Fail-Safe* state in the context of the SAPC is its capability to meet an application's safety goal by transferring it into a safe state. In more simple cases this may be achieved by passivating an application with a shutdown routine. However, the SAPC offers a more sophisticated measure to gracefully degrade applications and thereby uphold at least the basic functionality of the respective feature. This *Fail-Soft* state is a special form of the *Fail-Safe* state, allowing an application to produce less outputs or outputs of lower quality in order to reduce its required inputs or its resource use. For example, complex calculations can be replaced by linear approximations that take less time to calculate or less important input data can be supressed to achieve faster computations. Degradation is required if the peripheral hardware of a non-degraded application is not available, if not enough resources are available to run all required applications, or if the normal application is not available for all core nodes.

To determine which level of degradation shall be chosen for each application, the SAPC requires information on which applications it should prefer in case of not being able to schedule all applications. As the output of applications defines the quality of the corresponding feature, a partial ordering amongst all safety relevant features in the system is used to implicitly define the required application outputs. This partial order defines how the performance and the features of the vehicle shall degrade in case of a failure. As such, features with higher priority will be kept alive before features with a lower priority are considered. A feature and thereby all of its contributing applications may degrade and thus will be listed separately for every degradation level in the partial ordering. It is expected that a degraded feature will be listed with a higher priority, so it is considered first. Depending on its priority, the improved version of a feature is only included if enough resources are available. This does however not endanger safe operation of the vehicle, as the most important features are grouped and ranked with the highest level of importance.

4.2 Fail-Operational

Fail-Operational in the context of the SAPC is an application's need to be kept available. Certain applications are required for the safe operation of the vehicle and must always be run on at least one of the available core nodes, such as, the Steer-by-Wire feature in a FEV. These important features, in their highest level of degradation, form a minimal kernel of features that must always be active, thus being mapped to the highest degradation priority.

Moreover, features that need to complete within a certain period of time, will be represented as real time tasks to guarantee these requirements, whereas others will be executed as best effort tasks, hence not ensuring temporal determinism. To set up a new schedule after an adaptation, it is important to know the timing requirements, such as deadlines and failover times, of each real time feature. Especially in case of a time-triggered network, it is mandatory to process input data in a predefined period of time and to make the computing results available at the right point in time.

For this, the SAPC will support different redundancy strategies depending on how rapidly a backup instance must be available in case the original application fails. These strategies offer different compromises between their used resources and how quick a backup instance can be activated. As the start-up times are different for each combination of application and core node, the times must be compared with the required failover time of the application to determine a viable strategy.

Frozen-Standby

The most adaptive redundancy concept considered by the SAPC is a frozen-standby. A frozen-standby is stored on a software repository and its binary code is only loaded as required by supporting core nodes, as described in requirement TTT-009 of deliverable 2.2. This requires the core node and its operating system to support loading an application image and scheduling it at runtime. The advantage of this approach is that the core node running the backup instance does not need to be determined until an adaptation trigger occurs. However, this also implies that the application will not be available until the application image has been copied, loaded, and finally initialised by the target core node. By implementing this highly adaptive mechanism, the SAPC can be used to determine the general properties limiting a frozen-standby strategy for real time application, thus determining its feasibility for use in a fast fail-operation scenario.

Cold-Standby

In contrast to a frozen-standby approach, the application image can also be pre-installed on potential backup core nodes, thus providing a cold-standby option (see *reconfiguration of failed cruise control* use case UC_110_01 in deliverable 2.1). The backup instance can then be activated on one of the core nodes that were previously selected to host the pre-installed application image. The application must then only be loaded and initialised. In case of execute-in-place (XIP) memory the application can be initialised directly. Consequently, this approach provides a quicker failover time compared to the frozen-standby concept, but requires more flash memory. This can limit the number of core nodes that are potentially capable of hosting the backup application.

Hot-Standby

To further reduce the switchover time, a hot-standby application is run on a second core node. The hot-standby performs the same operations as the master, except that its outputs are supressed. In case of a failed master instance, the backup instance is triggered and instantly takes over the operation of the master. This strategy uses the most resources, as the application is run twice. Therefore, it should only be used where required by strict, small switchover times, for instance, as described by the Steer-by-Wire use case UC_111_01 in deliverable 2.1. The backup instance may also be a degraded version of the application, to reduce the used resources.

4.3 Fail-QM

Applications from 4.1 to 4.3 will be run as real time tasks. Fail-QM applications can run as a best effort task or even be skipped. For instance, comfort features like seat heating can be shut down at any time without affecting the ability to safely control the car.

The SAPC assumes that real time tasks are given enough calculation time to finish their calculations in time. No such guarantee exists for Fail-QM applications, which can only be monitored by quality of service (QoS) properties. To enforce this assumption of temporal isolation for independent real time applications, the RACE and TMDP platform both utilise an operating system capable of strict temporal partitioning.

5 Strategy for Safety & Optimisation

This chapter introduces the main strategy of the Safe Adaptation Platform Core (SAPC) which is used to assure safety and to achieve optimisation during adaptation. The next chapter will then enrich this strategy through an implementation concept, containing a detailed description from a design and runtime perspective.

5.1 Laws of Operation

After a fault occurs in a core node, the SAPC evaluates if any safety or mission-critical feature is affected. Such an impairment may manifest itself directly through non-functioning components or may only have an indirect effect, such as the lack of required redundancy. Consequently, the vehicle must adapt its operation mode to these new circumstances in any case. To ease the implementation of the SAPC, it is of benefit to formalise the global transitions from one operation mode to another. A fully functional vehicle remains in a *normal* law as long as no need for adaptation exists. From a theoretical standpoint, the SAPC's concept is capable to keep the vehicle operational as long as enough resources are available to service a minimal set of features. Figure 2 depicts all adaptation scenarios supported by this concept from the perspective of an entire vehicle:

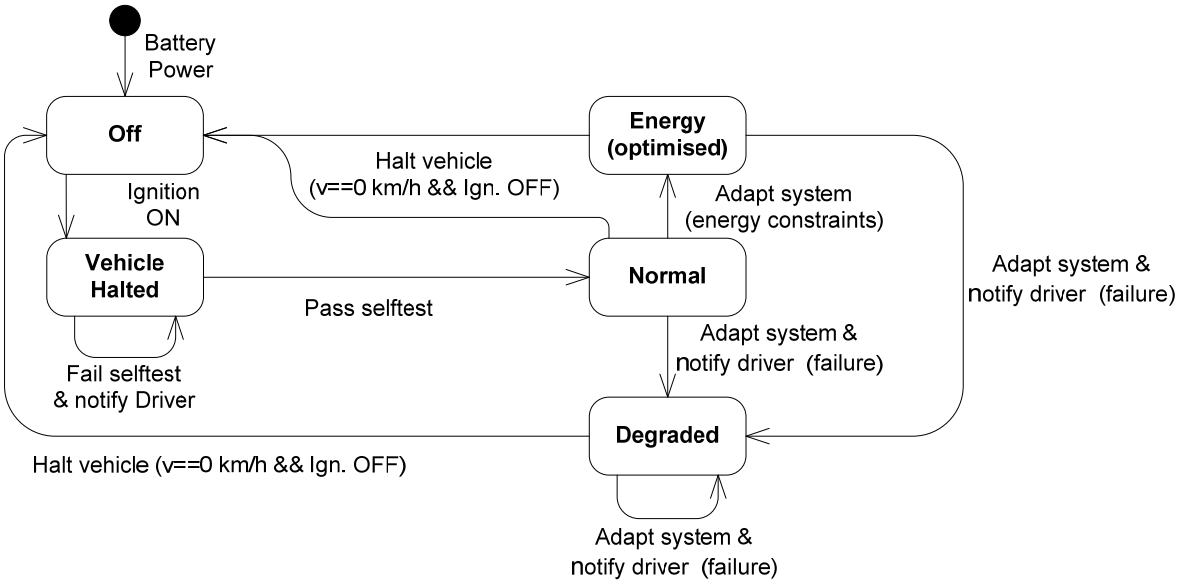

Figure 2: Laws of Operation (Vehicle)

In detail, as soon as a failure that requires adaptation occurs, the system transitions into a *degraded* law. This law only consists of a subset of features, thus compensating for the failures. Depending on the vehicle manufacturers' preferences, a *degraded* law may either force the driver to stop the vehicle as soon as possible or allow the driver either to continue driving in a limp home mode with limited torque and speed, or to continue operating the vehicle with a reduced set of features. However, this choice of a degradation strategy is limited by the hazard and risk analysis. For

instance, the operation of a vehicle is deemed unsafe, if crucial features cannot provide the required fault-tolerance mechanisms, thus forcing the driver to abort the mission. On the other hand, the analysis may also state that the vehicle can still operate in a limp home mode. In use cases where fault-tolerance cannot be completely recreated, this option can mitigate the risk of a hazard through the reduction of the vehicle's performance, thus allowing configurations that are not acceptable for operating the vehicle at higher speeds. Moreover, a *degraded* law may also maintain the full performance of the vehicle by providing the most important features with the required level of redundancy.

Conceptually, the degraded state may be maintained over multiple consecutive failures, as long as the SAPC can rely on certain fundamental properties of the system, such as a redundant power supply (see section 3.5 for details). Furthermore, the vehicle may also transition into an *energy saving* law, thereby also disabling or degrading functionality, for instance to increase the range of the vehicle.

SafeAdapt's prototype vehicle will demonstrate this law transition based on a typical automotive scenario where the first occurring failure will lead to a degradation state in which the mission can be safely aborted, but not continued. For this, each core node must implement a set of law transitions that are based on the vehicle's laws of operation. Figure 3 depicts the laws of a core node while also considering its initialisation phase:

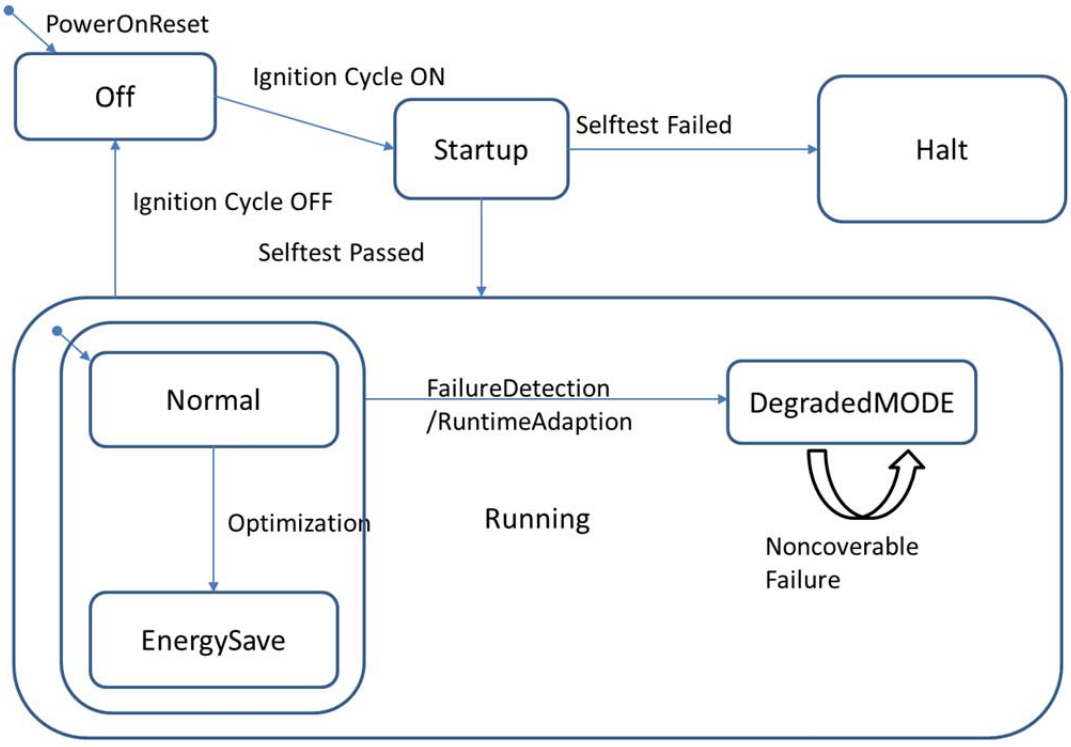

Figure 3: Laws of Operation (Core Node)

5.2 Theory of Adaptation

The SAPC's transition between laws of operation and the identification of the needs leading to such an adaptation are theoretically grounded within the concept of the M.A.P.E.-cycle, originating from the field of Autonomic Computing (see Figure 4). Therein, four phases in a cycle are defined to cover the entire adaptation process from monitoring of the system to execution of the adaptation based on common knowledge.

Figure 4: M.A.P.E.-Cycle

In every core node the local system state is *monitored* by collecting environmental information, hardware-specific diagnostics, and application states. This information is then compiled into a heartbeat in form of a health vector (see section 6.3.2 for details) and broadcasted to all other core nodes. Each node can now *analyse* the information and independently assess, if an adaptation is required based on the received information. This adaptation is *planned* in the next step. The *planning* phase makes heavy use of the *knowledge*. Depending on the needed adaptation and its requirement, this phase is performed in different ways. If an instant adaptation is required, the best known instant counter measure is chosen quickly. For example, available standby instances of failing applications are activated. If more time is available, e.g., in case the previous cycle already performed the instant adaptation, a plan is made for a new valid configuration that best matches the numerous adaptation constraints. If no adaptation is needed, the algorithm can use the reserved calculation time to increase its *knowledge*, e.g., create new instant adaptation plans for possible failures.

The results of the *planning* phase are then used in the *execution* phase to transition into an improved or safer state. Additionally, additional artefacts in form of instant adaptation plans created during the planning phase are added to the knowledge-base and can be used when a new need for adaptation was detected.

5.3 Fault Handling Strategies

The SAPC and its requirements should constitute a fault tolerant system which continues operating even if faults occur. Thus, such a system continues correct service provisioning in the presence of faults. Not only the hardware architecture, but also the software architecture needs to be designed in such a way that those faults are detected, handled, and recovered. SafeAdapt addresses this architectural challenge on the unit, component, and system level.

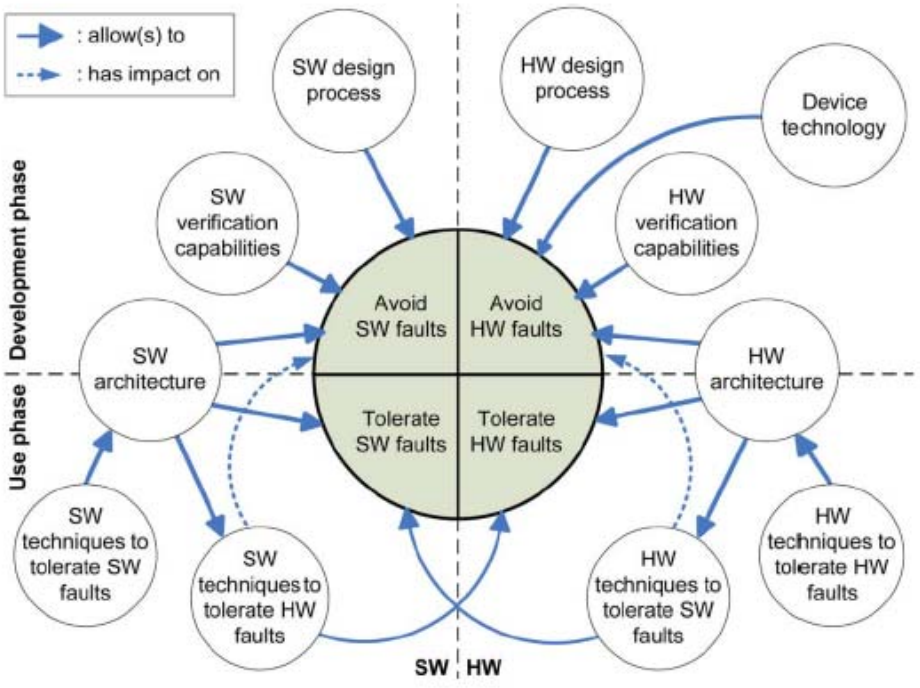

Figure 5: Fault Handling in Embedded Systems [Salewski et.al 2008] [1]

According to ISO 26262, it is most important to consider the following three types of failures:

a) Random HW failures can occur unpredictably during the lifetime of a hardware element and occur in accordance to a probabilistic distribution. Their persistence can be either permanent or transient depending on the fault's nature, such as, random defects inherent to a semiconductor, environmental conditions, usage, or handling.
b) Systematic failures are related in a deterministic way to a certain cause, which can only be eliminated by a change of the design, the manufacturing process, operational procedures, documentations, or the removal of software bugs and hardware design faults.
c) Common cause failures occur, when the failure of two or more elements of an item results from a single specific event or root cause.

[1] [Salewski et.al 2008] F. Salewski, S. Kowalewski, *Hardware/Software Design Considerations for Automotive Embedded Systems*, IEEE transactions on industrial informatics, vol. 4, no. 3, August 2008

There are many ways to categorise fault-tolerant design patterns. The following table depicts a possible approach to address different types of random hardware faults:

Fault-Tolerance Patterns				Addressed Type of Fault/Failure
Architectural	Independence			Common Cause, Cascading
	Redundancy	Spatial	Active	Random HW failures
			Passive	
		Temporal		Random HW failures, Systematic failures
		Informational		
	Diversity	Design (HW/SW)		Common Cause/Systematic failures
		Data		
Detection	Stateful	System Monitor (Logical/Temporal), Heartbeat, Watchdog, Acknowledgment		Random HW failures, Systematic failures
	Stateless	CRC, Hamming parity, voting, SW or HW based self-test algorithm, Read Back mechanism		Random HW failures, Systematic failures
Error Recovery	Quarantine	State indicator from voting unit		Random HW failures, Systematic failures
	Concentrated recovery	Minimise unavailability by focusing all resources on recovery activity		Random HW failures, Systematic failures
	Individual Decide Timing	Independent checkpoints		Random HW failures, Systematic failures
	Data Reset	Recover from an incorrigible data error by taking / computing initial values and approximate value		Random HW failures, Systematic failures
	Others	Hamming, memory partition reset		Random HW failures, Systematic failures

Table 1: Fault-Tolerance Patterns

Some of the mentioned techniques can be combined in such a way that different types of failures are addressed. In other words, techniques, such as, *Heterogenous Duplex Pattern* and *Diverse Redundancy Pattern* allow both random and systematic faults to be addressed at any level in the system design (from a complete system to a single component). This implementation is achieved in SafeAdapt by means of two diverse hardware architectural platforms developed from different suppliers together with the use of redundancy patterns (see section 4) at application level.

Furthermore, ISO 26262 standard states the following techniques as the way to address failures at the SW-architectural level:

Mechanisms for error detection at the software architectural level:

- Plausibility checks
- Detection of data errors
- External monitoring facility
- Control flow monitoring
- Diverse software design

Mechanisms for error handling at the software architectural level:

- Static recovery mechanism
- Graceful degradation
- Independent parallel redundancy
- Correcting codes for data

As part of specification of the SafeAdapt Platform Core in deliverable 3.2, a selection of these mechanisms will be chosen to satisfy the respective safety goals. For this specification, the following template will be used:

Fault Containment Region	**Failure Mode**	**Internal/ External**	**Internal/External Safety Mechanism**					**Fault Type**
			HW	***SW (Platform)***	***App***	***Fault Filter***	***SAPC***	
Core Node (TMDP)								
Core Node (RACE)								
Switch (TTE)								

Table 2: Template for the Specification of Safety Mechanisms

5.4 Forms of Adaptation

After a fault was detected and its impact and cause identified, the SAPC must choose from one or multiple adaptation options for every affected application, which are classified and explained from the view of an application in Table 3. The implementation of the SAPC will primarily focus on implementing the instantiation on a different core node, while passivating the defective node:

Form of Adaptation	Description
Different memory partition	An application is instantiated on different partition on the same core node. A partition is defined as a partially independent block of memory, which can be used if a fault can by confidently assigned to certain memory region and does not affect other memory regions.
Different core node	An application is instantiated on a different core node. In case the previous core node is defective, the node will also be passivated.
Degrade application (internal)	An application is provided with fewer resources during execution, such as optional input values or longer execution periods. The adaptation is based on different execution paths of the application. The SAPC must be aware of these options, to consider them during the adaptation's planning phase.
Degrade application (external)	A less resource intensive form of the application is instantiated. In comparison to an internal adaptation, external adaptation is based on instantiating a simpler version of the software.
Passivate application	An application is shut down.

Table 3: Forms of Adaptation

5.5 Failure Detection, Classification & Remediation

For safety purposes, failures within the system are detected in the analysis phase of the M.A.P.E-cycle, which is implemented as part of the fault filter (see section 6.3.1). A failure may be caused by various fault types ranging from single bit flips, over application and device design faults, to transient and permanent hardware failures. Each of the fault types may require different measures to recover. As such, this concept introduces a fault classification scheme to group them. This allows multiple different hardware platforms to internally map device specific faults onto this scheme, thereby guaranteeing a correct subsequent treatment by the SAPC. Further, the concept is designed to split the analysis phase from fault *observation* to *remediation* into four subphases. After the initial classification of faults, the SAPC must determine the *cause* of the observed faults to be able to determine its *impact* and finally derive the correct adaptation wish (see Figure 6). Furthermore, if

the cause cannot be derived unambiguously, the SAPC must consider all possible impacts and chose a remediation that comprises all of the non-excludable causes.

Figure 6: Fault Classification Scheme

More specifically, causes within the SAPC are grouped into abstract classes that can be directly mapped to a remediation strategy. The mapping between hardware-specific fault observations to SAPC fault classes is a platform-specific task, which is not handled within the SAPC. In detail, a preliminary mapping of SAPC fault classes to remediations is listed in Table 4. However, the table is subject to change, as the classification will be further refined during the implementation phase.

SAPC fault cause class	Remediation	Description
Maskable Transient Hardware Error	Remediation is directly performed in hardware or platform software. The SAPC remembers this occurrence in order to treat the error as permanent if it occurs more often.	Errors such as single bit flips in memory that are maskable by ECCs occur sporadically and are often not associated with a permanent hardware fault.
Hardware Partition Error	Passivate partition and move application to other partition or alternatively to other core node.	A hardware error that can be with certainty assigned to a region of the hardware, such as memory.
General Hardware Error	Passivate core node and reactivate applications on other core nodes.	Any hardware fault that cannot be assigned to a partition and thus makes the entire core node untrustable.
Application Design Fault	Passivate application (only QM)	Design faults may cause applications to overstep its WCET or produce incorrect values. This can however only occur for QM applications, as the rigorous application development, verification, and validation process eliminates the treat of design faults in safety-critical applications.

Table 4: Mapping of SAPC Fault Classes to Remediations

5.6 Energy & Resource Optimisation

In order to reduce the energy requirements of the system, the total computation load must be reduced. In the SAPC, this is performed by deactivating applications listed lowest in the priority-based adaptation rule system (see section 6.4.2). Based on this, the SAPC is capable of deactivating entire core nodes in order to save energy, or reduce CPU clock speeds to operate at a higher thermal efficiency. During design, similar to the failover concept, a planning tool can determine a highly optimised energy saving plan to utilise these two energy saving measures, which can then be effortlessly activated at runtime. Moreover, the planning tool can even determine multiple plans dependent on which set of features the driver wishes to keep active. The energy manager can use the same interfaces that are used to propagate adaptation wishes introduced by failures to the SAPC.

On the other hand, to reduce the amount of required core nodes, a dense system utilisation is required to not waste resources through idle CPU cycles. With respect to the rigid synchronous execution model of the SAPC in combination with pessimistic WCETs, a schedule is created that by design has a high idle rate to mitigate the seldom occurring risk of applications using the full WCET. To circumvent this inefficiency, the SAPC pads out these idle times with best effort tasks, thus enabling core nodes to be fully utilised while guaranteeing all deadlines.

6 Design & Runtime Concept

After defining the general adaptation strategy, a more detailed view on the runtime mechanisms, interfaces, and communication concepts is necessary to implement the system. For this, the following chapter describes the workflow of the SAPC and defines the required data needed to plan and perform an adaptation.

6.1 General Adaptation Workflow

The SAPC on every core node relies on three fundamental sources of information to derive an adaptation plan: a fault filter, cyclical information of other cores (health vector), and a local database with adaptation plans (see section 6.3 for details). Foremost, the platform-dependent fault filter collects and groups together detected faults to enable further interaction with the SAPC over one well-defined software interface. The faults can be triggered by the underlying hardware, detected by the local diagnostics modules, or directly propagated by applications. Moreover, the platform-independent SAPC cyclically receives information from all other core nodes including information about applications and planned adaptations. Finally, this variable information is analysed with adaptation rules and plans that are stored in a local database to determine the needed adaptation. As soon as a plan is found, the SAPC's platform-dependent complex device driver (CDD) is instructed to perform the adaptation which in turn uses a platform-specific method to transpose these device-independent instructions into commands that are understood by the underlying real-time operating system. In addition, the other core nodes are informed of this adaptation through the health vector. This control flow is depicted in Figure 7 and its artefacts are described in detail in chapter 6.3.

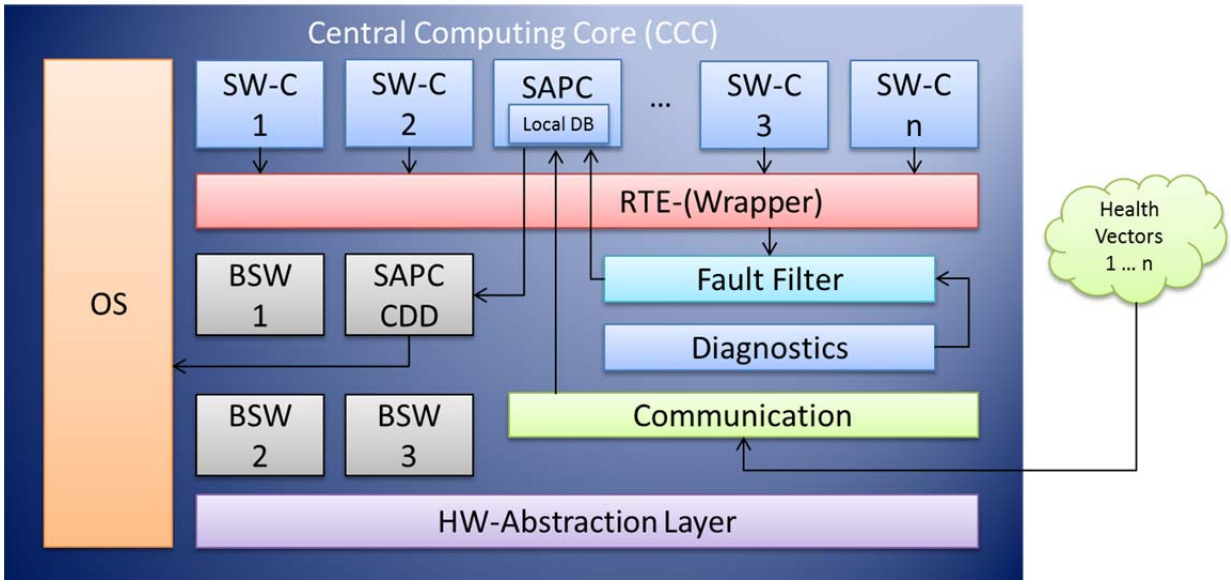

Figure 7: Adaptation Interfaces

6.2 Two-Phased Failover

To enable an almost seamless transition into an degraded law, the SAPC maintains a look-up table in form of *instant adaptation plans* (see section 6.4.1) to instantaneously choose a predefined configuration for at least the most general SAPC failure cause class (see section 5.5), as for instance covering a failover of each core node. As such, a switch to a hot- or cold-standby instance is simply performed by activating the application and possibly deactivating another predefined application currently occupying the needed computational resources, thus also potentially adjusting the scheduling of the entire system. With rising combinatory possibilities after every new law transition, this form of predefined configurations however turns more troublesome. For instance, the storage required for maintaining these configurations increases. This makes an on demand and situation dependent calculation of a new configuration more beneficial. After eliminating an eminent threat in the first failover phase, the SAPC is now capable of searching for new configurations in a second phase that is not bounded by such a strict temporal interval. This optional planning capability is useful for scenarios in which a driving mission should not be aborted after the first failure of a core node.

6.3 Information Sources of the SAPC

6.3.1 Local Fault Filter

The SAPC implements a universal interface to group fault causes in accordance with the previously discussed classification scheme (see section 5.5). However, as fault types vary throughout the computing platform types, a platform specific filter is individually developed to map the extensive amount of faults onto the reduced set of generic SAPC fault types. Furthermore, application specific fault detection in the time and value domain may be outsourced to generic system services or directly implemented in the application. For instance, simple comparison of target and actual values or the detection of non-received messages are prime candidates for being implemented as system services through their well-defined nature. On the contrary, complex plausibility checks are better situated in the application itself due to their context specific nature. Therefore, the application is offered an interface to propagate such anomalies. As this is also a platform- and application-specific design decision, the fault filter service is also applied to this scenario to offer the application a platform-specific channel to propagate such anomalies and further coincide with the universal SAPC interface.

6.3.2 Health Vector

Every SAPC is cyclically informed about the health of all core nodes. This is done in form of a health vector, which enhances the heartbeat concept by adding finer grained runtime information, as listed in Table 5 and depicted in Figure 8. As every health vector is mapped to a fixed time slot within the time-triggered network, a receiving core node is also able to detect non-received health vectors and is thereby capable of inferring the other core node's health status.

Property	Possible Values	Description
cnID	Any positive integer	Unique computing unit identifier
cnState	PASSIVE	This meta-state covers all sub-states of a core node in which it is not yet able to meet its functional and non-functional requirements. In this state the core node does not send out any application data. From a receiver's perspective it is not required to have a more detailed view onto the state within PASSIVE. The PASSIVE state includes the sub-state ISOLATED. An ISOLATED core node will remain in this state for the rest of the active driving mission. Through restarting, repair, and self-tests the isolation may be removed again.
	ACTIVE	The core node is able to execute the assigned set of applications.
	READY2SLEEP	The core node is de-initialised, all applications have been stopped and the core node is ready for sleep. The state READY2SLEEP will be maintained until all other core nodes are informed of this new state, before the core node finally transitions into the SLEEP state. To return into an ACTIVE state again, the PASSIVE state is used to indicate a future reactivation of the node.
appConfig	appStatus (struct)	Deployment, application standby information, and adaptation plan per application

Table 5: Attributes of Health Vector

The *appStatus* structure further contains the properties listed in Table 6:

Property	Possible Values	Description
appID	Any positive integer	Identifier for the application, unique within the vehicle.
runningAsMaster	True & false	Application is running as master.
runningAsNewMaster	True & false	Application is planned to run as master. This is required to ensure that the decentralised computation of the new configuration reaches an identical result on all nodes.
appState	PASSIVE	The application is passive.
	INIT_INITIAL	This state signals that the application is initialising during a normal start-up, wherefore execution-time may be longer than the WCET of NORMAL.
	INIT_ADAPTATION	This state signals that the application is initialising after an adaptation, wherefore execution-time may be longer than the WCET of NORMAL.
	NORMAL	The application is functioning as intended.
	ISOLATED	The application was isolated.
performanceLevel	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	Calculated performance of application (lower values represent better performance)

Table 6: Attributes of Health Vector's AppStatus

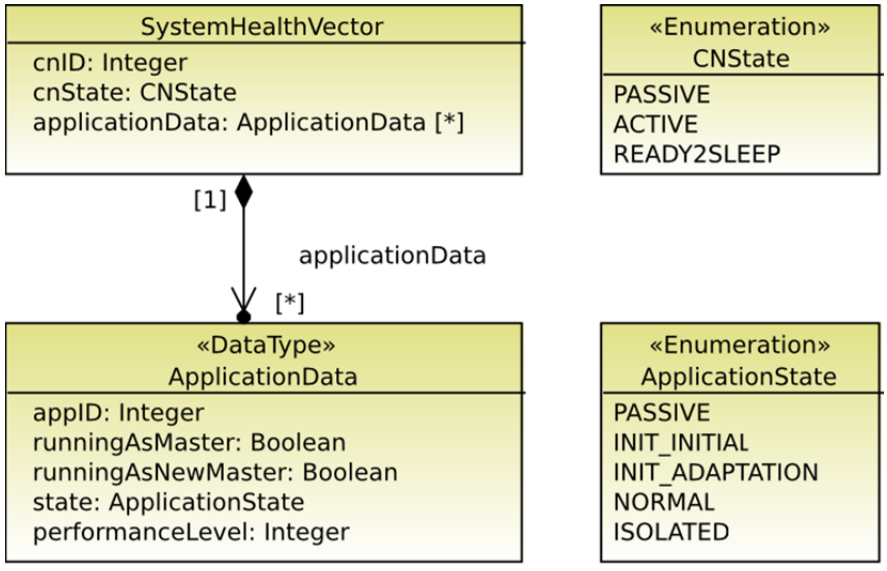

Figure 8: UML-Model Representing the Health Vector

6.3.3 Local Database

Next to local and remote runtime information, which is respectively provided by the fault filter and health vector, the SAPC accesses adaptation design decisions through a local database to calculate the intended adaptation behaviour for every possible single event failure scenario. More precisely, the database is conceptually split into two sections. One section for *instant adaptation plans*, which precisely define new schedules for every predicted failure and optimisation scenario. The other section defines *rules of adaptation* which the SAPC or an offline planning tool can apply in a self-determined manner to find a new valid configuration or calculate new instant adaptation plans. The rationale behind this duality within the configuration method is based upon the need to guarantee quick failover responses for applications that require short failover cycles, as previously described in the two-phased failover concept (see section 6.2). When searching for such a new configuration, it is desirable to not only find a valid setting but to also apply as little degradation as possible while maintaining a high level of operational efficiency. In addition, activating a standby instance on another core node may well cause all core nodes to swap to an entire different schedule to guarantee inter-application timing dependencies. However, in many adaptation scenarios, this poses an overly disruptive action, wherefore the planning algorithm must be capable of finding instant adaptation plans with limited variance in scheduling changes to remain within valid bounds. For instance, the transition from one valid schedule to another must consider that the failover periods of all applications are not affected by changes in the scheduling offsets. The definition of these bounds is however platform specific and will be derived during the implementation phase.

As the computational complexity of this optimisation problem rapidly increases with the amount of deployed applications, finding the optimal configuration may not be feasible within an acceptable timeframe when using the vehicles computational resources. Therefore, the local database's adaptation rule system is designed to be expressive enough to find non-optimal configurations that meet a minimal quality threshold through heuristics to reduce the time required for finding a valid setting. In contrast, an offline planning system, which is used during the design phase of the vehicle, can calculate optimised solutions that can then be stored within the vehicle as predefined configurations. Allowing this coexistence of offline and in-vehicle planning provides the most flexible design approach, as vehicle manufactures can freely make use of both approaches depending on the used hardware and application loads. The first adaptation step for every failure type and energy optimised configuration are prime examples for configurations that can be stored in a prepared format, as the number of initial failures and energy saving options remains within manageable bounds. In contrast, secondary adaptation steps are more likely to benefit from the in-vehicle planning system, as the combinatorial possibilities increase exponentially with every subsequent failure.

Moreover, the schema of the local database does not vary between offline and online planning. This allows both online and offline calculations to rely on the same data and algorithms, thus emphasising the development efficiency of this hybrid planning system. In addition, the *system guarantees* (see section 3.5) ensure that the defined information is sufficient to always find a valid configuration that also exhibits the intended behaviour without requiring supplementary expert knowledge, thus furthermore highlighting the design simplicity of this approach. Similarly, the

conjunction of strict system guarantees with the local database enables the automatic generation of the local database from modelling artefacts occurring in earlier steps of the design phase. The information contained within the local database is described in detail in section 6.4.

In SafeAdapt, the instant adaptation is planned to be shown as part of the demonstrator, whereas the feasibility of online and offline planning algorithms will be determined through simulation or other appropriate methods.

6.4 Content of the Local Database

The Local Database (LDB) of a SAPC consists of two main sections*: instant adaptation plans* and *rules of adaptation*, which a described in detail in the remainder of this subsection. *Instant adaptation plans* precisely define new schedules for every predicted failure and optimisation scenario. On the other hand, *rules of adaptation* define the information required by the SAPC or an offline planning tool to find a valid configuration or calculate new instant adaptation plans. This relationship is depicted in Figure 9 and described below in detail:

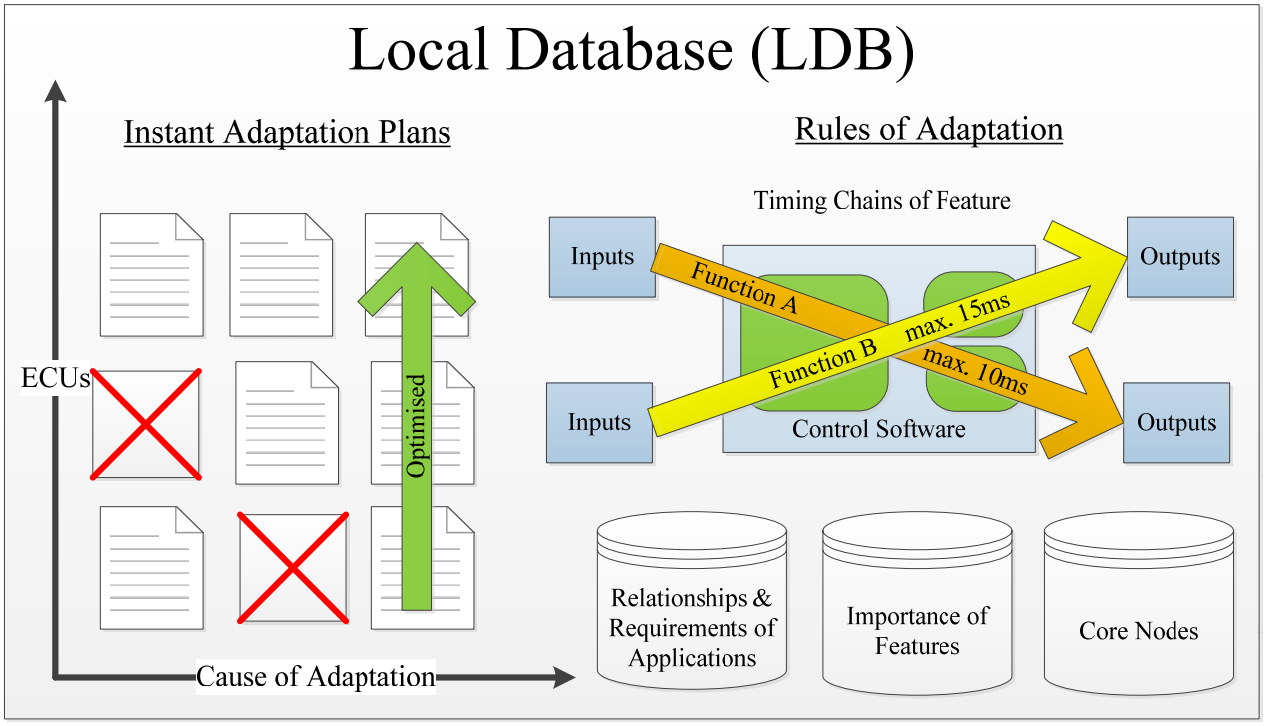

Figure 9: Structure of Local Database (LDB)

6.4.1 Instant Adaptation Plans

In detail, *instant adaptation plans* always map a specific failure or any other adaptation trigger to a new configuration for the entire system. This is inevitable, because the dynamic activation of a task on a new core node may negatively affect unrelated timing chains, as inter core node execution offsets of applications have to be changed to accommodate for the scheduling requirements of the new application. Therefore, satisfying all these dependencies can lead to cyclical timing dependencies that are not always computable within the limited timeframe available for adaptation.

Consequently, such a new configuration contains the cause of failure and scheduling information for every task on all remaining core nodes. In addition, the deviation from configuration to another are known, thereby allowing the configuration to be stored in a reduced format on the core nodes, as only relevant information and not the entire new schedule needs to be stored.

6.4.2 Rules of Adaptation

Furthermore, the *rules of adaptation section* of the local database defines several attributes to capture crucial hardware properties and the requirements of every feature in a well-defined format:

Timing Chains

Features, such as Steer-by-Wire or cruise control, are each modelled as a simple chain consisting of only input, one abstract computation, and output elements, which are then further annotated with timing requirements of each feature. More specifically, the defined timing requirements only apply to the software within the core infrastructure and are derived from the actual physical requirements during the design phase (see Figure 10). For instance, aging of data within external sensors during sampling periods or latencies of data transiting from the secondary networks beyond the core gateways into the core system are therefore not included in this total timing chain figure. Moreover, the time-triggered nature of the gateways dictates when input values are written to the core network and when output signals are expected to leave the core network again. These fixed timing requirements provide the only reference point to the systems execution periods and hence influence the SAPC's scheduling decisions as the execution of all application is calculated in relation thereto. In detail, every input and output signal is not only described by a period and offset, but also by its occurrences in a hyper-period. A hyper-period is defined by the least common multiple of all periods, hence also including the periods of all applications. The hyper-period is determined during the design phase, as all periods of applications, inputs, and outputs are already known during this phase. Thereafter, the planning algorithm places the execution of all applications on this pre-calculated hyper-period. Since applications with executions times lasting longer than their periods would lead to incorrectly planned schedules, this form of execution is prohibited and the system designer is constrained to expressing execution overlaps within one application through the definition of multiple applications.

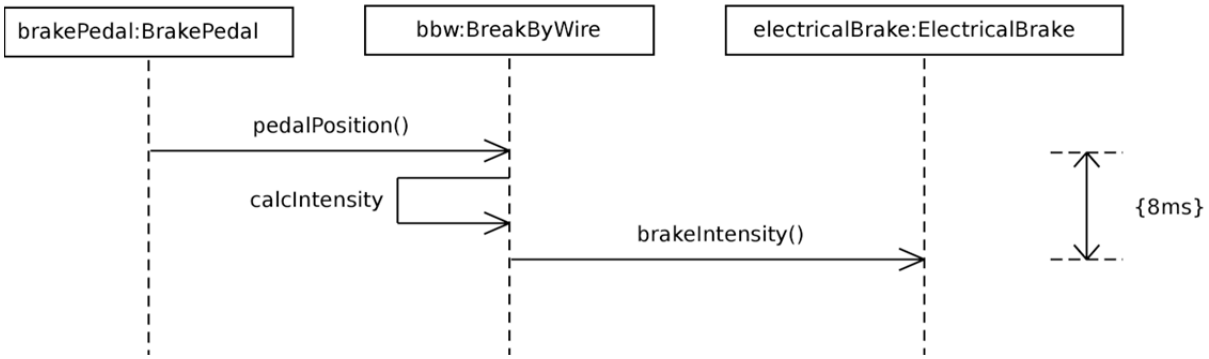

Figure 10: UML-Model of End-to-End Deadlines

However, not all applications require such a rigid schedule. Applications can either be defined by multiple degraded application configurations with longer periods or deadlines to guarantee a minimal servicing interval. This approach is beneficial for important applications that only suffer from non-safety critical effects, if scheduled less often. Therefore, they can subsist in a system with reduced resources. On the other hand, there are applications that do not pose a hazard if not executed. They can be executed in the slack times of rigid applications and are only monitored by an application specific runtime Quality of Service (QoS) management system. This system can in turn trigger the deactivation or the reallocation of this application without affecting the primary system schedule as the application is only executed in slack time. Furthermore, the creation of a fully utilised system is infeasible as execution time is reserved by overestimated WCETs, which is then only partially used during normal operation, hence allowing this concept to improve resource utilisation. Additionally, input-output (I/O) intensive tasks benefit from being executed as often as possible when resources are available, instead of only being executed according to a strict cycle. To define this best effort behaviour a simple omission of the WCET value is sufficient for signalling the SAPC that this application shall only be executed during idle times. If required, additional execution time can be reserved for QM applications to at least account for their standard execution times.

Relationships & Requirements of Applications

The previous described *timing chains* are broken down into single applications that are required for the correct operation of the driving function within the *relationships & requirements of applications* subsection, thus spanning a function graph of all features. This section of the local database lists all applications of the vehicle and links them by specifying their predecessors and successors, which can be other applications or final input and output elements. As the final output elements are also listed within the *timing chain* section, the planning algorithm is capable of deriving the complete mesh of applications required to support a feature and allows applications to exist in multiple timing chains without further configuration. Obviously, this representation is capable of letting multiple features depend on the same application effortlessly. In addition, the concept prohibits the occurrence of cycles within application dependencies to circumvent infinite worst case execution times of entire timing chains.

Furthermore, each application needs a different time to execute on a different CPU type, wherefore the worst case execution time of an application is stored for every core node. By not only defining the WCET per CPU type but per core node the encoding scheme implicitly acts as a general indicator if an application is capable of executing on a certain core node, as for instance local hardware related requirements or non-existent application code may make the execution on a different core node infeasible. Then again, WCETs are commonly determined in absence of pre-emption, thus further information would be required to account for effects such as context switch induced cache invalidations that in turn prolong the execution of the task. To keep the concept simple, WCETs will first be defined by considering the effects of pre-emption thus determining the WCETs in absence of caching benefits, but without including CPU and OS overhead, as this information is already directly considered within the planning algorithm. This concept may

however be further refined during the implementation phase through bounding the occurrence of pre-emption, if efficiency is negatively affected by these overly pessimistic WCETs.

Moreover, applications may bear different failover requirements (see section 4) that are also relevant for determining new allocations. Further, these failover types implicitly determine if two applications can be co-hosted on the same core node, thus making an explicit configuration superfluous. For example, a hot-standby instance should not be executed on the same core node as the primary instance, as a loss of the core node would render the standby instance useless. Similarly, core nodes that are only wired to one and the same power supply are also incapable of hosting a set of primary and active standby instances.

Importance of Driving Functions

In case of system degradation, the SAPC determines which features must be deactivated and which features should only be degraded internally to comply with the restricted set of computational resources. Therefore, features are prioritised by a number indicating if a feature is indispensable or how favourable its operability is in comparison to other features. Hereby, multiple applications can share a priority, thus indicating that all applications within that priority must be scheduled to fulfil the degradation level. Furthermore, features can either degrade by utilising less inputs and serving fewer outputs, or by being activated less often. To differentiate these modes of degradation in the local database, a feature that changes its input and output requirements, or its interval of execution (period) is modelled as two distinct features. This form of degradation leads to a diverging selection of applications that, for instance, may require less inputs and thus perform less resource intensive operations, but in turn produce less precise or less optimised control values.

Similarly to the degradation of features, every application may provide multiple degradation variants which may either be due to application-internal changes or the replacement of an entire application with a simplified version. From a configuration perspective, a new application is modelled as soon as the resource requirements or the amount of input and output ports changes, even if these degradation variants of the application are based on the same source code. The reason behind this separate modelling of an application’s degradation levels is based on the need to capture the deviating resource requirements, such as the worst case execution time. In turn, the SAPC can calculate the resource savings of scheduling a degraded application, thus allowing other application belonging to features of a higher priority to be scheduled. In addition, the SAPC must prevent outputs from being serviced twice, which is potentially possible through defining a normal and degraded version of a feature. However, through assigning every output with an output type this hazard can simply be eliminated by the SAPC as it is implemented to only service every output type once.

Core Nodes

To determine if a configuration is feasible, the SAPC or an offline planning tool must also know the number of core nodes and their interdependencies, such as shared power supply lines, to create configurations that prevent catastrophic outcomes from failures within one power system. Further, to determine the maximal amount of pre-emption that can influence the execution time of an application, the SAPC must also factor in the temporal overhead caused by hardware and the

operating system. The temporal overhead is encoded in a single number indicating the duration of one context switch for every core node. During the implementation phase, this value may also be added to the platform-specific WCET of an application to simplify this data structure. Further all core nodes are numbered strictly monotone to provide a universal tie breaker for resolving conflicts that may occur in the decentralised planning system. Without a universal tie breaker the SAPCs on different core nodes could otherwise calculate conflicting configurations. Additionally, every core node must be aware of the network packet sizes and the assigned generic communication slots in relation to the system's hyper-period so that the planning algorithm is capable of aligning application execution to transmission windows (see section 3.4). Annotating every core node with these network-specific values, enables later optimisations through assigning unevenly distributed network slots to core nodes, when certain applications are considerably more communication intensive.

7 Summary

In order to add a substantial value to the future development of fully electric vehicles, this document defines several goals in the categories of safety, energy efficiency, maintainability, and extendibility. The Safe Adaptation Platform Core (SAPC) is designed to attain the goals by adapting to failure scenarios and optimisation needs through predefined and dynamically planned reconfiguration. For this, the systems concept is designed to address safety in accordance to the ISO 26262, while enabling energy optimisation through the deactivation or degradation non-vital features. Furthermore, SafeAdapt continuously aims at benefiting from related project, such as the EU FP 7 projects DREAMS and SCARLETT.

In detail, the SAPC represents a software module that is executed on every TMDP and RACE platform to perform the required local adaptation. It builds upon a system architecture with redundant time-triggered communication paths, a synchronous execution model, a communication scheme based on broadcasting, strict temporal partitioning, and an AUTOSAR interface for enabling the execution of applications on multiple platforms. Based on this, the SAPC is developed in a platform-independent manner to enable the integration on the TMDP and RACE platform. To interact with the SAPC through its specified interface, each platform maps its own platform-specific fault types to one of the unified fault types of the SAPC. Besides the detection of local faults, the SAPC interacts with all other SAPCs in form of cyclically exchanged status information to reach a consistent view on the global system state.

Next to these volatile sources of information, the SAPC utilises a local database that contains a formalised description of the network, the computing platforms, and all application related requirements that are required to determine a valid system configuration. In addition, this local database also stores predefined system configurations for every adaptation scenario, which are used by the SAPC to perform a rapid failover or activate an energy saving mode. Moreover, the SAPC may optionally calculate a new situation-dependent system configuration in a secondary adaptation step to enable the driver to continue the driving mission after a first failure occurred. Furthermore, these adaptation options are formalised through operational laws describing the state of the vehicle’s control infrastructure.

From a safety perspective, the SAPC offers multiple safety states, such as, Fail-QM, Fail-Safe, Fail-Soft, or Fast Fail-Operation, to meet the application’s temporal failover requirements. These safety states are further mapped to runtime reconfiguration mechanisms, thus providing options for hot-standby, cold-standby, or a repository-based failover. In addition, a failure handling framework is presented, describing a complete collection of methods to mitigate the risks associated with different types of faults. The framework will be used as part of the future specification of the SAPC to select appropriate methods to systematically address all potential fault causes and illustrate the respective platform’s and SAPC’s capability to mitigate these risks.

List of Abbreviations

Abbreviation	Definition
APP	Application
ASIL	Automotive Safety Integrity Level
CCC	Central Computing Core
CDD	Complex Device Driver
CN	Core Node
E/E	Electric/Electronic
FEV	Fully Electric Vehicle
HW	Hardware
RACE	Robust and Reliable Automotive Computing Environment
RC	Rate Constraint
QoS	Quality of Service
SAPC	Safe Adaptation Platform Core
SEooC	Safety Element out of Context
SW	Software
TMDP	Trusted Multi Domain Platform
TT	Time-Triggered
TTE	Time-Triggered Ethernet
XIP	Execute in Place