A safe generic adaptation mechanism for smart cars

Alejandra Ruiz, Garazi Juez
ICT- European Software Institute Division
Tecnalia
Derio, Spain
{firstname.lastname}@tecnalia.com

Philipp Schleiss, Gereon Weiss
Fraunhofer ESK
Hansastr. 32, 80686 Munich
Munich, Germany
{firstname.lastname}@esk.fraunhofer.de

***Abstract*—Today's vehicles are evolving towards smart cars, which will be able to drive autonomously and adapt to changing contexts. Incorporating self-adaptation in these cyber-physical systems (CPS) promises great benefits, like cheaper software-based redundancy or optimised resource utilisation. As promising as these advantages are, a respective proportion of a vehicle's functionality poses as safety hazards when confronted with fault and failure situations. Consequently, a system's safety has to be ensured with respect to the availability of multiple software applications, thus often resulting in redundant hardware resources, such as dedicated backup control units. To benefit from self-adaptation by means of creating efficient and safe systems, this work introduces a safety concept in form of a generic adaptation mechanism (GAM). In detail, this generic adaptation mechanism is introduced and analysed with respect to generally known and newly created safety hazards, in order to determine a minimal set of system properties and architectural limitations required to safely perform adaptation. Moreover, the approach is applied to the ICT architecture of a smart e-car, thereby highlighting the soundness, general applicability, and advantages of this safety concept and forming the foundation for the currently ongoing implementation of the GAM within a real prototype vehicle.**

Keywords—self-adaptive systems; ISO 26262; fail-operational

I. INTRODUCTION

Cyber-physical systems are evolving towards smart appliances with self-X capabilities, like self-healing or self-optimisation. Such self-adaptation in CPSs must however still meet safety requirements. On the path towards fully self-adaptive systems, there are several intermediate steps to be taken. A major evolutionary step could presently be imminent in the automotive domain. Modern cars are more and more becoming smart cars which are foreseen to drive autonomously and fully electric. Both trends have great impact on the system architecture of the vehicle. With an electric car many mechanical parts, like the control gear, vanish from the car and are replaced by electronic parts. As also the backup properties of mechanical parts are removed, additional measures must be taken to ensure the same safety, e.g., for a steer-by-wire system, which allows steering a car electronically without a mechanical connection to the steering wheel. With the emerging progression towards automated driving functions, vehicle systems have to ensure a higher degree of availability. The driver is not expected to monitor the car, as long as there is no unforeseen event. In case of the latter, the car requests the driver to take over the control within a certain time frame. Thus, when a subsystem fails, critical functionality has to remain operational.

This work was partially funded by the European Commission within the Seventh Framework Programme as part of the SafeAdapt project under grant number 608945.

This is particularly challenging as present automotive systems are able to reach their safe state by shutting down in case of a malfunction. For example, a failure of an ECU (Electronic Control Unit) in a traditional car can be handled by turning of this unit without losing control of the driving behaviour. In contrast, an ECU hosting a brake-by-wire application cannot be shut down without losing the ability to brake, as long as there is no costly mechanical backup installed. This shift from fail-silent to fail-operational systems poses a great challenge for future automotive systems, since fail-operational behaviour is up to now only implemented in other embedded systems domains with different constraints, such as the cost per unit in avionic systems [1].

Current automotive approaches rarely support fail-operational applications [2]. Despite this, current research is heading into the direction of creating functional safety concepts that provide fail-operational behaviour at system level for individual functions. Self-adaptation poses as a vital option to overcome critical failures and allow for the further operation of a smart car. For instance, in case of the breakdown of an ECU, another ECU can be adapted to take over the execution of the critical functionality. Such redundancy mechanisms are presently only planned for complete system parts in many embedded systems domains, like in a car with three identical redundant ECUs for a steer-by-wire application. Another example is adapting the system to changing contexts, such as different driving situations [3]. Self-adaptation provides efficient solutions to these problems but lacks the same inherent safety.

In order to provide additional concepts for safe self-adaptation, this paper presents a novel generic adaptation mechanism for smart cars. The implementation of self-adaptation in a per application fashion is not cost-efficient since it only permits a limited reuse of developed safety artefacts. Therefore, a generic adaptation approach is introduced, which provides system-wide adaptation in an uncoupled and safe way. Moreover, special focus must be laid on hazards that are newly introduced through the use of adaptivity.

In the next section, related work is discussed. Thereafter, safety requirements are described in Section III, which then lead to the definition of a generic adaptation mechanism in Section IV. Based on this, the main hazards of such an adaptation are derived in Section V, thus leading to a unified safety concept in Section VI. In order to evaluate this concept, its applicability is demonstrated in an automotive case study of a smart car in Section VII, followed by a review of used evaluation methods. Finally the findings are discusses and concluded.

II. RELATED WORK

Self-adaptation has been identified as a promising solution to many upcoming challenges of distributed software systems [4]. For distributed embedded systems much research has been carried out for considerations of incorporating self-adaptation.

Exploiting the concepts of an Integrated Modular Architecture (IMA), reconfiguration of avionic systems has been pursued [5]. It focuses to enhance present avionic application software standard ARINC 653 for reconfiguration. However, the approach is strongly based on its application in the avionics domain, e.g., standardised IMA infrastructure or higher costs for single products than within the automotive mass market.

With [6] an approach has been presented which utilises an agent-oriented paradigm to self-adapt industrial automation systems enabling self-healing. A safety concept is not explicitly considered and the approach targets a different domain which allows application of agent-technology. In the area of telecommunications, a context-aware self-adaptive system for mobile devices has been presented [7], which exploits semantic web services for composing only non-safety-critical functions. The above approaches outline selected examples for self-adaptation in different embedded systems domains, but they do rely on domain-specific characteristics that prohibit the application to smart cars, e.g., no considerations of safety in mobile devices.

For automotive systems, an early attempt towards enabling self-adaptation has been the EvoArch project [8], in which a car's functionality is managed via a market-oriented paradigm. Therewith, required and provided services are adaptively orchestrated. As it has been an initial concept, no details on safety aspects have been researched. In [9] a service-oriented approach exploiting a Java-based middleware is introduced, enabling self-configuration of a vehicle's functionality. Safety is not explicitly considered and single points-of-failure, like a centralised configuration manager, are accepted for simpler system designs. The DySCAS project [10] aimed at flexible self-configuration of an automotive system by introducing a novel middleware. Self-adaptation is handled with policies, which can be updated in distributed variation points in the system architecture. However, no safety concept for the self-configurable vehicle system is provided. All these projects implement self-adaptation of a car's system through diverse mechanisms but do not consider safety aspects. With [11] an approach for data-flow oriented design of self-healing automotive systems is presented, which allows considering redundant components as a safety mechanism. Nevertheless, safety aspects beyond designing this redundancy are not considered.

Safety aspects of a new car ICT architecture are presented in [12] focusing on providing the control platform as a compositional safety element according to ISO 26262. The thereby introduced analysis is specific to the proposed novel system architecture. Heckemann et al. [13] propose the concept of a "safety cage" which is an independent safety mechanism. They define a software safety cage as: "a piece of software that monitors the behaviour (outputs) of the original function and takes appropriate actions if a malfunction is detected". This approach proposes a reaction at the application level to contain the failures and mitigate its effects on the whole system. The objective is to detect the failure and trigger a graceful degradation of the application.

Another view for safety applied on adaptive systems is the one presented in [14] that utilises modular conditional certificates. ConSerts are post-certification artefacts (i.e., certification has been conducted in the traditional way) equipped with variations points bound to formalised external dependencies that are meant to be resolved at runtime. They focus on verification of the guarantees and needs from the application point of view, whether an application needs inputs that are offered by other applications. [15] identifies challenges in developing self-adaptive systems and managing uncertainty. Whittle [15] mentioned that "While a few techniques have been developed to support the monitoring and analysis of requirements for adaptive systems, limited attention has been paid to the actual creation and specification of requirements of self-adaptive systems. As a result, self-adaptivity is often constructed in an ad-hoc manner". This has ended up producing architectures, in which safety is not the main concern. The systems manage to handle the adaptation, but how the system is developed in order not to introduce new hazards is not taken into account. Rushby [16] on the other hand discusses in particular the certification issues of self-adaptive systems. It is stated that "safety-critical functions that operate adaptively all the time seem especially challenging to certify, so there is likely to be a discrete switch to adaptive mode and this will be employed only when conventional controls are unable to cope. The trustworthiness of the mode switch from normal to adaptive behavior is therefore particularly critical. One attractive idea is for the mode switch to be triggered by monitoring the runtime behavior of the system against its safety case". He moreover highlights, as the adaptation mechanism itself is implemented in software, that the software also becomes the part which inherits the issues with regards to assuring safety. One of the main concerns for certifying self-adaptive systems is that the operation behaviours are not completely determined at design time. Therefore, [17] proposes using model checking for verification self-adaptive systems as an adequate technique. As they mentioned "these techniques are applicable under the assumption that vital system characteristics, as for example system configurations, are known at design time and, more importantly, continue to hold at runtime".

Diverse research activities in the area of self-adaptation have been carried out. However, either they deal with novel architectures or mechanisms, not taking safety-criticality sufficiently into account, or approaches solely focusing on assuring safety of self-adaptive systems in general. On this account, this work introduces a novel approach covering a generic adaptation mechanism for enhanced architectures of smart cars, as representative example for a self-adaptive CPS, and the corresponding safety concept.

III. SAFETY REQUIREMENTS

In distributed control systems, safety hazards may occur through both hardware faults and systematic software errors. In order to reliably identify software errors during runtime, it is inevitable to rule out any hardware deficiency as the root cause of incorrect system behaviour. As such, this work focuses on detecting and managing hardware faults in an unambiguous

and reliable manner through the use of strong diagnostic capability. This consequently allows to infer software error through the absence of hardware faults at runtime.

From a hardware fault perspective, the envisioned system must provide means to handle faults in a hierarchical manner and assign a certain fault observation to a specific hardware element. For this, the following list summarises the types of failures that must be handled by the presented adaptation mechanism:

- *Platform failure*: A random hardware failure that affects an entire computing platform. The whole platform is considered as a fault containment region.
- *Memory failure*: Permanent memory cell, bank, and area failures are classified as memory failures.
- *Clock failure*: A failure of an internal clock leading to an incorrect notion of passed time.
- *Power supply failure*: A permanent fault or a transient power supply faults, like a crank pulse or other short power drops.
- *Sensor failure*: A value of a sensor is no longer available or the provided data is incorrect or inconsistent.
- *Network failure*: Problems on the availability of the network elements, i.e., the path to reach those elements is blocked or the communication is corrupted or not carried out in the expected time slots.

IV. Generic Adaptation Mechanism

A. Redundancy Management

To enable systems to reconfigure in a situation-dependent manner, knowledge of a new system configuration is required. Based on such knowledge, a runtime system can then transition into this new configuration through the assistance of a redundancy management scheme. In fully self-adaptive systems, both the planning and transitioning phase are envisioned to correctly function at runtime. To attain such high aims, this work exclusively focuses on the subtopic of dynamic reconfiguration, which forms a cornerstone on the path to comprehensive self-adaptation. As such, all reactions are predetermined during design-time following an extensive hazard analysis. Consequently, each anticipated failure is mapped to a specific system configuration that is capable of mitigating the respective hazard.

More precisely, the concept of a *general adaptation mechanism (GAM)* is introduced to manage system-wide and predetermined reconfiguration plans at runtime within a dedicated module. For this, each processing unit within a managed system is equipped with an instance of the GAM to detect adaptation events, such as the failure of another device, and depending on the event, instruct the underlying operating system to transition into a new configuration. As such, the GAM is a reusable, generic, and platform-independent software artefact that operates in a distributed manner in order to provide a globally consistent system state. Here, special attention must be paid in order to ensure its consistency, which is enforced through a temporally well-defined exchange of status messages between GAM instances operating on different computing units.

In comparison to traditional safety functions, which are characterised by their simplicity, the GAM concept combines redundancy and other needs for adaptation within a single logical software artefact. Considering this and the distributed nature of this generic adaptation management concept, it is inevitable to also derive safety goals for the GAM in order to ensure the correct functioning of this safety function. Here, the hardware architecture is of special interest, as it lays the foundation for creating a line of argumentation to prove certain guarantees and behavioural characteristics of the generic adaptation mechanism.

In detail, reconfiguration relies heavily on the cyclic exchange of heartbeats between processing units. As such, the system must consist of at least two processing units that are connected through two physically independent channels and powered by two independent power sources, thus excluding communication link and power failures from a fault cause analysis. Moreover, each computing platform must meet minimal diagnostic capabilities in form of lockstepped processing units to detect deviations in the calculations and thereby infer a local hardware fault.

B. General Software Architecture

Based on these properties, a globally consistent state is provided, which is enforced through an adaptation mechanism on every device. More specifically, the software architecture of this adaptation mechanism is composed from multiple software artefacts, each providing a distinct functionality. At the architecture's core, an *adaptation logic* module is responsible for reacting to local hardware faults and changes within the system's global state. For this, a *communication* module is responsible for cyclically receiving heartbeats from all other control units. As such, the adaptation logic can infer the failure of another control unit through the absence of a heartbeat. Moreover, a simple lookup table is utilised to determine the required configuration for a new system state, such as compensating for the failure of another processing unit. This information is stored within a *local database* based on analysis performed during design time, to determine a safe behaviour of the system for any single failure within this control architecture.

In case of unsalvageable local faults, the adaptation logic is further capable of discontinuing operations through a fail-silent mechanism, thus preventing incorrect system behaviour from occurring. Subsequently, another predetermined control unit will detect this failure of a control unit, and therefore adapt accordingly through the assistance of the *adaptation logic* module. As this module is independent from a specific platform implementation, an additional level of abstraction in form of a *diagnostics* and *fault filter* module is required to determine if a specific local hardware fault leads to an entire control unit being considered unreliable, or if the fault may be masked or mitigated by platform-internal mechanisms. Similarly, a new system configuration must be translated into a sequence of platform-specific commands, in form of modifica-

tions to the operating system's schedule. To accomplish this task, a *complex device driver* module is utilised, thus allowing the *adaptation logic* module to trigger reconfiguration without having to consider the specific scheduling strategy of the underlying platform. This relationship is depicted in Figure 1.

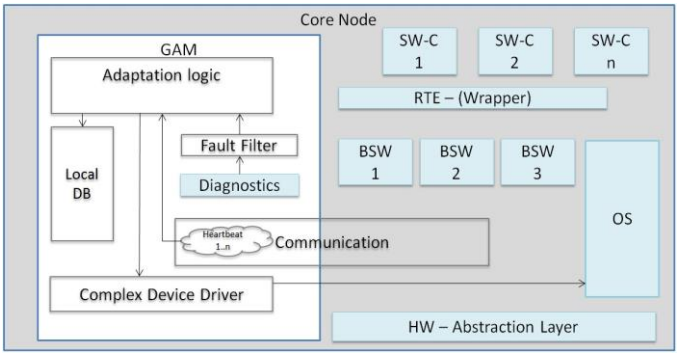

Fig. 1: Adaptation Interfaces

V. ADAPTATION HAZARDS

In the context of ISO 26262 [18], every item without internal safety mechanisms shall be evaluated as part of a hazard analysis and risk assessment process. As such, each vehicle function should be analysed in order to determine if the risk of a specific hazardous situation can be mitigated. Despite this, relying on a single concept for managing the availability of multiple independent functionalities can lead to the occurrence of new types of hazards due to its systematic nature. Consequently, hazards that were not considered during an initial functionality-based hazard analysis must also be accounted for when utilising the GAM concept. For this, a closer look at the different forms of adaptation is required, to derive a set of hazards that must additionally be addressed.

In general, the adaptation mechanism utilises different methods to adapt to a new situation. Foremost, after detecting an unmaskable local fault, a platform is *passivated* by self-isolating itself through the disabling of all communication links in order to prevent further error propagation. Similarly, individual non-critical applications can be passivated in order to free enough resources for scheduling critical tasks after failure situation, as part of a *graceful degradation* strategy. Moreover, to complete a reconfiguration process, required tasks must once be *activated* on another platform by transitioning from a non-operation into a fully operational state, thus providing at least a minimal set of functionality to operate a vehicle safely.

During this reconfiguration process, it is inevitable to prevent certain system states from occurring. More specifically, a design fault within the interpretation of local hardware diagnostics may falsely trigger a local passivation as part of the item's safety function. As a systematic occurrence of such false trips would however endanger the availability and safety of the entire system, additional design measures must be taken to prevent such situations from occurring. On the contrary, an incorrect detection of a remote platform's failure can cause a functionality to be provided by two software instances, thus potentially leading to an unspecified control behaviour. Therefore, a method is required to ensure that a platform is either entirely isolated or alternatively accessible from all other relevant platforms. In addition, the time required to perform such a reconfiguration must be bounded during the system's design phase to ensure that any situation triggering an adaptation is handled in an acceptable time frame, thus not endangering the real-time properties of affected applications. For this, a *fault tolerant time interval (FTTI)* is introduced for each application to describe the maximal acceptable interval in which the functionality may remain in an uncontrolled state. Moreover, any platform participating as part of the proposed system must be developed to the highest safety level amongst all applications hosted within that system. This typically leads to the general requirement that all platforms should be developed in accordance to the requirements of the ASIL D. As such, the prevention of design and hardware faults through appropriate isolation measures is an essential prerequisite for utilising the general adaptation mechanism.

VI. FUNCTIONAL SAFETY CONCEPT

To ensure that the previously introduced requirements, with respect to mitigating the risks of newly introduced adaptation hazards, are appropriately addressed, this section introduces an innovative and refined functional safety concept. In detail, this functional safety concept and its mechanisms for attaining fault containment and mitigation of potential hazards are described from a software, a hardware architecture, and a communications perspective to provide legibly evidence of why a single fault will not lead to a violation of any of the system's safety goals.

Form of Adaptation	Description
Platform Failover	An application is instantiated on a different computing platform, after the primary platform was categorises as defective.
Degrade Application	An application operates with fewer resources requirements, such as discarding optional input values or permitting longer execution periods. This form of adaptation is based on different execution paths of the application, thus the GAM must be aware of these options.
Passivate Application	An application is disabled either as part of a degradation strategy or as part of the passivation of an entire platform.
N-Version Failover	A different version of an application is activated to accommodate potential software errors. This may include diversely implemented software or the use of simpler control laws.

Table 1: Forms of Adaptation

The previously mentioned fail-silent behaviour is implemented at platform level whereas the selection of different fault tolerant and adaptation strategies occurs at system level to provide fail-operational behaviour. Here, the executed automotive function plays an important role in the selected adaptation strategy. This includes different redundancy strategies

i.e., hot-, warm-, and cold-standby or graceful degradation. Moreover, in order to react to software design errors, adaptation may also include a mechanism for activating a diversely developed application, thus allowing to react to unspecific behaviour of an application in absence of any hardware faults. In essence, Table 1 summarises the possible adaptations covered by GAM.

A. Hardware Architecture

The hardware architecture of a system based on the GAM concept requires at least two fail-silent computing platforms with diverse hardware, platform software, and embedded safety mechanisms in place. Based on this, the required level of fault tolerance is ensured through different strategies, which are highlighted in green in Fig. 2 and further discussed in the ensuing paragraphs.

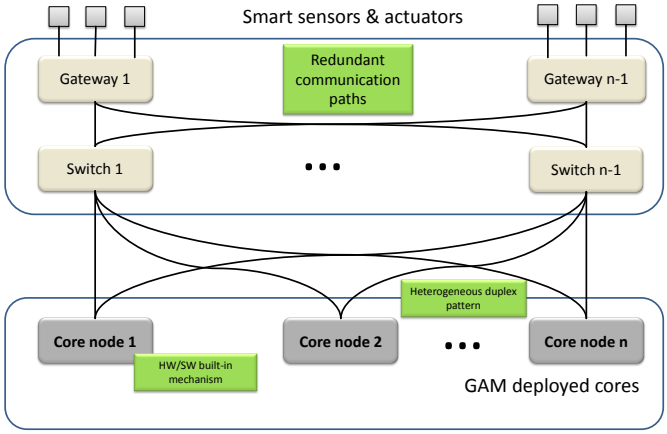

Fig. 2: Safety Concept on an Architectural Level

1) Network Topology

Failures within a communication medium are generally detectable and mitigatable through redundant and independent communication channels. In case of a switched communication medium, a double-star configuration may be used to attain redundant communication paths. Further, to achieve the required level of fault tolerance, sensors and actuators need to be replicated. More specifically, critical sensor data is validated with a 2-out-of-3 (2oo3) voting strategy to ensure the correct perception of the environment. This widely known hardware architectural spatial redundancy pattern [19] allows high protection against random hardware faults. The voter module compares the outputs from all sources of information and uses techniques such as majority voting to get a reliable value.

2) Diverse Redundancy Patterns

For several years great effort has been devoted to the study of fault-tolerance design patterns [20]. One of the preceding redundancy patterns is formally defined as *Heterogeneous Duplex Pattern*. It targets the management of random and systematic faults to increase both the reliability and the safety of a system. This pattern incorporates two independent and diverse hardware channels (pair of core node) designed and implemented by different teams and with different technologies. Especially, the combination of different hardware technologies, such as the use of different microcontroller or FPGAs provides an effective building block for preventing systematic and common cause failures. In sum, this solution has a high random and systematic failure rate, thus being a very effective solution for applications requiring high safety integrity levels.

Based on this, a backup core node will work in standby mode and only take over the critical functions of the primary core node after that core node fails. As a matter of fact, each core node must hold a set of hardware and software safety mechanisms to support ASIL D applications and to guarantee fail-silent behaviour at component level. One of the main underlying benefits is that the implemented hardware based safety mechanisms extensively reduce software based error detection time. This allows different error detection and correction procedures inside the electronic control unit depending on the nature of the failure. Concerning covered fault types, transient faults are directly covered at ECU level whereas not recoverable ECU level permanent faults, such as permanent clock failures, are reported to the *Fault Filter* so that a global adaptation process can be started.

3) HW & SW Fault Management Mechanisms

In the same way as redundancy architectural patterns are deployed at system level, lockstep architectures [18] [21] are a guarantee at core node level to be able to execute the highest ASIL applications. In lockstep mode, operations run almost in parallel and the results are compared by an independent comparator. If a mismatch is detected in the output of the two processors, a flag is activated. The generated trap usually leads to either reinitialising or switching to safe mode in case of a not recoverable fault. This decision depends on the hardware and the safety concept of the ECU. Nevertheless, a reoccurring fault detected by a lockstep mechanism typically leads to the conclusion that the platform is not trustable anymore. This means that the whole core node would fail silently. Consequently, no critical applications will be hosted on that core anymore.

Since the two channel concept of the system can be built through the use of diverse hardware elements, the utilised fault tolerance pattern can deal with systematic, random, and common cause hardware failures. Hence, it is very effective for transient errors, Arithmetic Logic Unit (ALU) type failures, and direct current faults, e.g., stuck-at and bridging faults. As claimed, hardware diversity (design, layout) and isolation provide effective coverage for common cause failures as well as systematic failures. The drawback of this approach is that it could be extensively complex to prove the diagnostic coverage. CPU failure modes are also covered by software implemented build-in self-test (BIST) tests and the inclusion of watchdogs. The latter are capable of detecting scheduling and timing errors. Regarding memory protection, all memory is protected by hardware Error Correction Codes (ECC) and a Memory Protection Units (MPU) that stretch over the whole address space (including peripheral registers) and enable a simple separation of software, thus guaranteeing the freedom of interference. Freedom of interference is assured between different ASIL and QM software in terms of timing and execution, and exchange of information. Further, software-based integrity checks may also be implemented.

To finish with some of the most relevant safety mechanisms at platform level, external and internal monitoring mechanisms are also a guarantee to cover different failure modes. They offer a solution for the detection of clock (clock monitors) and voltage errors (under-voltage and overvoltage detection). In sum, all hardware faults, detection mechanisms, and respective counter measures that are enforced by the GAM concept are listed in Table 2.

Fault Region	Detection mechanism	Fault Containment	System reaction
Core Node failure	HW lockstep	System	Failover
Memory failure	Recoverable by MPU	ECU	No need for failover
	Not recoverable by MPU	System	Failover
Clock failure	SoC internal WD	System	Failover
	ECU WD	System	Failover
Power supply failure	Fail-silent	System	Failover
Sensor failure	Input loss	ECU	Redundant paths
	Input comparison	System	Degrade application
Network failure	Input comparison	ECU/System	Redundant paths
	Recoverable by CRC	ECU	Depending on frequency of occurrence
	Not Recoverable CRC	System	Redundant paths & Failover

Table 2: Hardware Failure Management

B. Communication Perspective

1) Globally Consistent State

Another key element of the architecture is the Extended Heartbeat (EH). This periodically transmitted signal poses as an innovative error handling solution to monitor the status of the core nodes by means of a time-triggered transmission between them. It contains a platform specific status, including information on the currently running applications. It is periodically sent from one core node to all other nodes in a predefined time slot through a communication medium that can guarantee the transmission within this time slot. It is absolutely essential that the system does not differ from the temporal behaviour defined at design time. To simplify the temporal determinism and partitioning of the system, all computing platforms must communicate over a synchronous communication medium, such as a time-triggered network.

In case of a not recoverable core node level failure, such as a power failure, the extended heartbeat is not transmitted and the other core nodes must take over the lost functionality. Moreover, an extended heartbeat also holds information about its origin in form of a core node identifier and the state of each application instance. Such states may include if an application is operational, passivated, or in a standby mode.

2) Data Integrity

An extended heartbeat must further include two different mechanisms to ensure information correctness and integrity:

- Cyclic redundant check
- Rolling counter

The Data Validation (Integrity Check) is responsible to provide checks on the input data and the system itself during the execution of the derived algorithm. A cyclic redundancy check (CRC) is included for that purpose. Range checks or correctness checks are carried on basis of functioning parity or CRC checks. Likewise, a *rolling counter* is added to guarantee that the current message has been updated since the last computation cycle. This is used to detect stale and omitted transitions.

C. Software Perspective

After the functional safety concept has been described at architectural level, the focus is in the following laid on the software mechanisms implemented by the GAM. These mechanisms help to complete a safe adaptation process by ensuring the correct functioning of the GAM. The different safety mechanisms used to shield the system from the violation of a safety goal are depicted in Fig. 3 and where required further explained in the following.

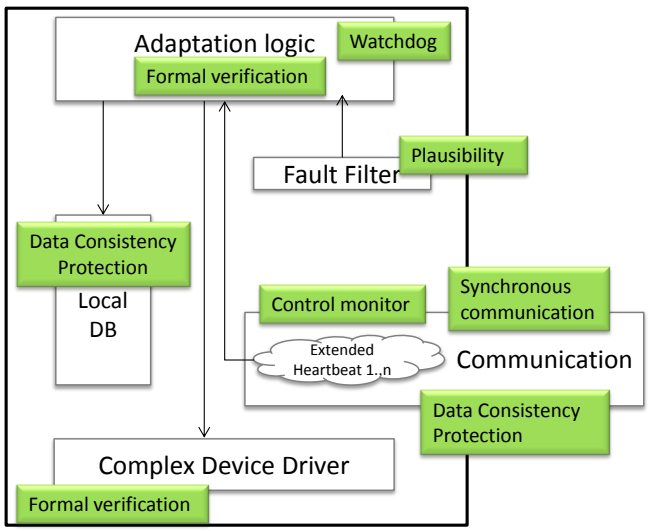

Fig. 3: Safety Concept of Adaptation Mechanism

1) Plausibility Checks

Among the different software based fault tolerance patterns, the ISO 26262 highly recommends plausibility checks as error detection mechanisms at software architectural level to reach the highest safety level for applications. They check the integrity of any signal by means of a specified reference model of the desired behaviour, assertion checks, or comparing signals from different sources. In other words, some predicates are defined in a set of variables to determine their validity at runtime. This is used to filter the set of failures, which the GAM can handle, and performance ranges of the core.

2) Data Consistency Protection

The previously mentioned CRC strategies are not only used within the extended heartbeat. In the same way, data consistency protection should be ensured between the different platforms' local databases. The content of both of them must be equal and neither incoherencies nor inconsistencies will be found between them. Even if this feature is guaranteed during the design time, the local databases include information redundancy mechanisms such as parity bits or CRCs.

3) Formal Verification

Likewise, fault tolerance is achieved through different mechanisms to detect or correct random hardware faults, systematic ones must be either avoided or removed during design time. Jean-Claude Laprie [22] argued that techniques such as formal verification can be applied to ensure fault-free designs. This is carried out by performing model checking to find possible design errors of the Adaptation Logic and Complex Device Driver modules. For the GAM concept, model checking is performed to verify whether the component model meets a given specification.

VII. Automotive Case Study

In this section, the applicability of the safety concept of the generalised adaptation mechanism is demonstrated by means of an automotive case study. For this, a smart car is currently under development to incorporate the safety architecture described in Section VI. Based on this vehicle, three representative adaptation scenarios are selected to demonstrate the capabilities of the safety concept, which is further evaluated and discussed.

A. Smart Car Setup

For final testing of the GAM concept a real electric car will be used within the presently ongoing SafeAdapt research project [23]. This smart car includes novel electronic system architecture with centralised hardware platforms implementing the GAM. As in modern cars many functions are controlled purely electronically, redundancy mechanisms are required to ensure their availability, even in the case of a failure. For now, the approach is limited to single failure in the system so that a safe state can be reached. After this, the mission has to be aborted safely, i.e., through halting the vehicle.

For the evaluation, the focus is laid on three different applications with varying types of criticality with respect to their assigned safety level. A Steer-By-Wire (SBW) function enables the electric steering of the vehicle, by sensing the driving and steering wheel angles, calculating the intended wheel angles, and actuating the change of direction via motors to the front axis. An Adaptive Cruise Control (ACC) function enables the vehicle to automatically keep its distance to an in front driving or stopping car. For this, it perceives the surrounding environment through radars and cameras, calculates the appropriate throttle value, and actuates the engine control. Both are critical vehicle functions. As the SBW functionally must re-establish an operation state within milliseconds after a failure, is it implemented as a hot-standby application on a backup node. As the failover times of the ACC functionality is less critical, it in contrast can be implemented as a more resource-efficient cold-standby application in cases where the availability of the ACC is desirable after a failure. As a third exemplary application, the control of the air condition (A/C) was selected, which has no requirements with respect to availability.

The currently developed smart car will include two different and powerful computing platforms, which execute the SWB, ACC, and A/C functions. In this example case study, Platform 1 (P1) hosts the SBW and ACC, whereas platform 2 (P2) executes a hot-standby version of SBW (see Fig. 4) and can potentially activate a new ACC instance. In order to demonstrate use cases with more than two platforms, the following scenarios will consider the existence of additional platforms (Platform n).

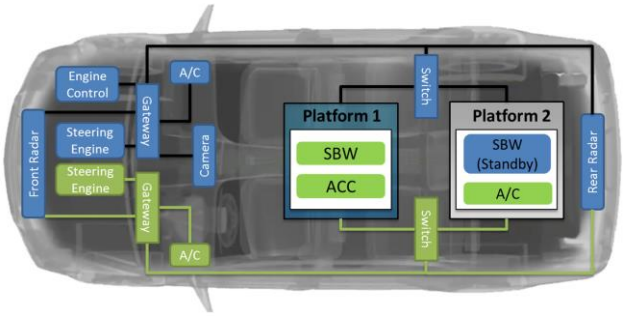

Fig. 4: Exemplary System Architecture of a Smart Car

B. Platform Failover

In this scenario, the smart car is expected to be in driving operation on a road, when a defect, such as short circuit, leads to an immediate failure of one of a platform (here, without the loss of generality, P1). P1 has been executing the SBW and ACC applications (see Fig. 5). This situation generates the need for an adaptation in the form of a failover to the remaining platform P2.

Here, the failure of P1 and the accompanying transitions into a fail-silent state was so abrupt that no extended heartbeat (EH) could be sent. Consequently, the adaptation logic block (AL) of platform two detects the lack of a new EH from P1 and thus starts a predefined adaptation process to mitigate the present hazard. For this, the AL block will fetch a new system configuration from the local database, which is implemented as a look-up table. To enable an almost seamless transition into a degraded state, this look-up table is maintained in form of instant adaptation plans. As such, a switch to a hot- or cold-standby instance is simply performed by activating the respec-

tive applications through adjusting the schedule of the entire system. After eliminating an eminent threat in the first failover phase, the AL block 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 in general useful for scenarios in which a driving mission should not be aborted after the first failure of a platform.

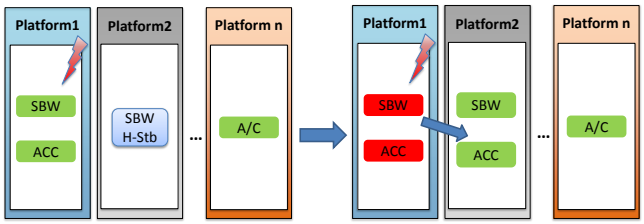

Fig. 5: Adaptation Scenario 1- Platform Failover

In this scenario, the first action is to activate the hot-standby instance of the SBW application on the unaffected platform. This is the quickest and most resource-consuming form of adaptation. Next to this, an instance of the ACC application will also be activated on an unaffected platform. As this form of adaptation was specified as cold-standby during the design phase, the newly activated instance has to first attain a valid perception of the system, before transition into an operational state. Consequently, this initialisation of the ACC cannot be performed as rapid as the SBW reactivation, which is however acceptable for its defined temporal criticality of the application. In essence, both applications will be running again, before the driver notices that the vehicle has been in a not fully functioning state. To define a maximal acceptable failover time, design time simulations have been carried out with respect to the performance of the vehicle's dynamics in the most time-critical driving scenarios, as part of a preliminary hazard and risk assessment. Regardless of this, the system will notify the driver by means of a warning message after a failure occurred. At this point, the driver has to stop the vehicle to prevent a hazardous situation.

C. *Degradation of Application*

In a second scenario, adaptation is not triggered by a platform failure, but through false sensors data. More specifically, input data of one of the sensors is identified to be no longer trustworthy.

Just as on the previous example, there are two critical applications running, the SBW and the ACC. In this example, the SBW application executing as a hot standby instance on the second platform is a more basic application (e.g., with respect to maximal driving speed) than the primary instance. In this case, the standby application does not require data from the faulty sensors.

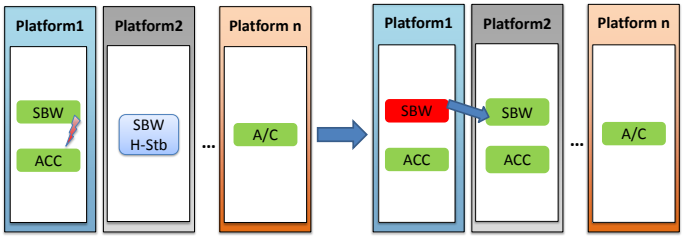

Fig. 6: Adaptation Scenario 2 – Degrade Application

When the failure of a sensor is detected on the primary platform, the fault filter block from this platform will inform the AL. As the degraded version of the affected SBW application is hosted on another device, the platform will passivate the primary SBW application. Simultaneously, the second platform will activate the degraded SBW instance, which does not require data from the optional and untrustworthy sensors. Arguably, this type of adaptation would also be performed within the application of the primary platform. As the backup instance is however required for other failure scenarios, it may also be reused for these types of failures (see Fig. 6).

D. *Energy-Efficiency*

In the third scenario, the adaptation is not triggered by a failure, but by the need to reduce the energy consumption in order to increase the expected range. The battery management system will detect that the battery charge of the vehicle has decreased below a predefined threshold, indicating a low battery status. This value is passed as warning to the system through the Fault Filter. The latter informs the AL about the low energy state and triggers the switch to a low energy configuration through passivation. This form of adaptation does not have the same time requirement for carrying out reconfiguration as compared to the other introduced scenarios. The objective here is not to isolate a fault and reactivate an operational instance, but to reduce the energy consumption. Thus, the time window for choosing the most adequate configuration is not as tight.

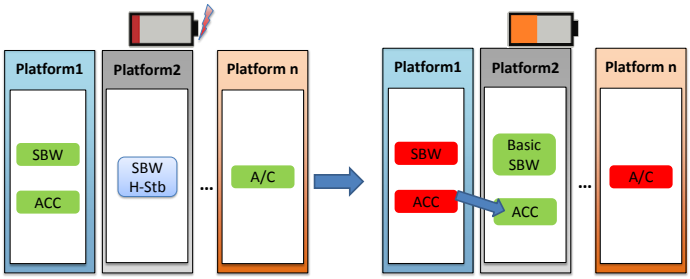

Fig. 7: Adaptation Scenario 3 – Energy-Efficient Configuration

Nevertheless, the new configuration plan has to take into account the energy consumption of the applications and their criticality. For example, in this scenario the less critical application with the highest energy consumption is the A/C, which is therefore going to be passivated. In this example, the adaptation is able to switch off the A/C, which is one of the most battery consuming applications (see Fig. 7). Further, the control unit hosting this application may transition is a passive sleep state, as no essential applications are hosted on that device.

VIII. VERIFICATION, VALIDATION & TEST PROCESS

The presented scenarios of the smart car highlight how the system with the GAM is capable of handling expected failures. Moreover, the system's safety concept has to be validated and it must be ensured, that all identified hazards have been avoided or mitigated to a tolerable level of risk. For this, a safety case development method was used in order to support the claims about the fulfilment of all safety goals. This has also provided useful means of identifying the type of evidences

requirement to support the claims. The results of different verifications at different development phases are the ones used in the safety case to support the adaptation safety goal fulfilment. Fig. 8 shows an excerpt of this safety case regarding the fulfilment of the goal that the adaptation has to be triggered correctly. In the following, the different validation strategies are discussed in detail.

Foremost, the GAM itself is developed using model based design. This provides the opportunity to apply formal verification, so that fault avoidance can be assured. At design time, a state machine was defined to ensure that the algorithm is executing only the necessary logic for the adaptation phases, in which it currently resides. To apply the concept of a state machine onto a distributed control system, it was necessary to ensure a synchronised execution of applications on all platforms. This form of synchronisation allows to define state transitions of an individual platform within the context of the entire system. In detail, the execution of each application is assigned to a fixed execution window during design time, further allowing a simple verification of the systems temporal correctness. Based on this, the system's reaction to a failure in form of a transition into a new configuration is easily described through explicit transitions at each execution point of the GAM.

In addition, the behaviour for each type of adaptation covered by the design must be tested. The examples presented in this paper have been used to define test cases. For defining the vehicle applications that will require adaptation, care must be taken to identify all highly critical applications with respect to safety and availability.

Within the project, two validation and test platforms have been used to ensure the safety of the GAM concept. Primarily, a vehicle dynamics simulation software [24] is used to determine the maximal acceptable failover times for individual software components. This simulation environment will focus on testing the perceived vehicle dynamics that define the maximum time for the GAM to execute a specific adaptation plan, before control of the vehicle is lost. Thus it helps to evaluate and validate the expected fault tolerant time interval at vehicle level. This is especially important for defining the controllability of the vehicle during the adaptation. Additionally, Hardware in the Loop (HiL) tests will be executed on a test bench to verify the GAM behaviour in the diverse use cases.

Moreover, a real car [25] is utilised to test the final system. Here, the objective is to demonstrate the software behaviour and system architecture in a real working car, while at the same time testing the proposed functional safety concept. The test setup of the vehicle's hardware architecture consists of two platforms that were developed diversely. For this, a platform developed by Delphi [26] and another developed by Siemens [25] are used as core nodes within the vehicle. As such, the GAM software component is deployed on both platforms. These two platforms have already been developed to fulfil the minimal diagnostic coverage for unambiguously detecting hardware faults in accordance to the requirements of the GAM concept. Moreover, the platforms are enhanced with software to map hardware-specific faults onto the generic GAM fault model.

To further evaluate the failure management behaviour in early development stages, models have been designed on basis of the EAST-ADL [27] architecture description language for automotive embedded systems. Moreover, the Ernest tool [28] is being utilised to test the system configuration for unanticipated failure cases on basis of a simulation.

IX. Discussion

With respect to the introduced safety concept, it should be noted that the increase of complexity in automotive application functions requires more sophisticated functional safety concepts. For instance, autonomous driving will introduce complex control algorithms, which in turn will exhibit a higher risk of containing implementation or design errors. Consequently, strategies, such as switching to a simpler control law, will have to be applied to the automotive domain in order to ensure the safe and autonomous operation of a vehicle. Moreover, strong hardware diagnostics are required to unambiguously assign an observed fault pattern to a specific software error. Therefore, it is inevitable to provide strong diagnostic in order to rule out any other hardware-related cause, as this may eminently lead to wrong or even hazardous forms of adaptation.

Moreover, self-adaptive systems are focused on adaptation functionality. Despite this, no harmonised verification process exists for these types of systems, as for instance, the functional safety standard ISO 26262 does not provide any guidelines on how the safety of adaptive systems should be assured. Regardless, this work has tried to apply the objectives of ISO 26262 on a self-adaptive system.

Even though systematic software errors are considered for control applications, the GAM must be free of systematic errors, as it represents a central critical element to ensure the correct functioning of a control system, and thus further cannot provide a safe state. As such, this work focuses on applying both formal techniques to verify the design and implementation correctness, as well as conducting additional tests through the use of simulation frameworks. In sum, this conservative approach provides an even higher level of confidence with respect to the correct configuration and functioning of a general adaptation mechanism.

X. Conclusion

Smart cars which are capable of self-adaptation hold great potential with respect to enhanced features like extensibility or provisioning of fail-operational behaviour. This, however, raises challenges for the safety concept of such vehicles. The presented approach introduces a generic adaptation mechanism, which enables a safe adaptation of a car's functionality. Based on the hazards introduced with adaptation a functional safety concept was derived. Its soundness and advantages are shown by the application to automotive use cases. Thereby, it is highlighted that this generic adaptation mechanism is able to handle common types of adaptation scenarios in a safe way.

Future work is ongoing with implementing the concepts in a real electric vehicle, thus for instance ensuring the safe operation and highly available of a steer-by-wire functionality within the course of the SafeAdapt project.

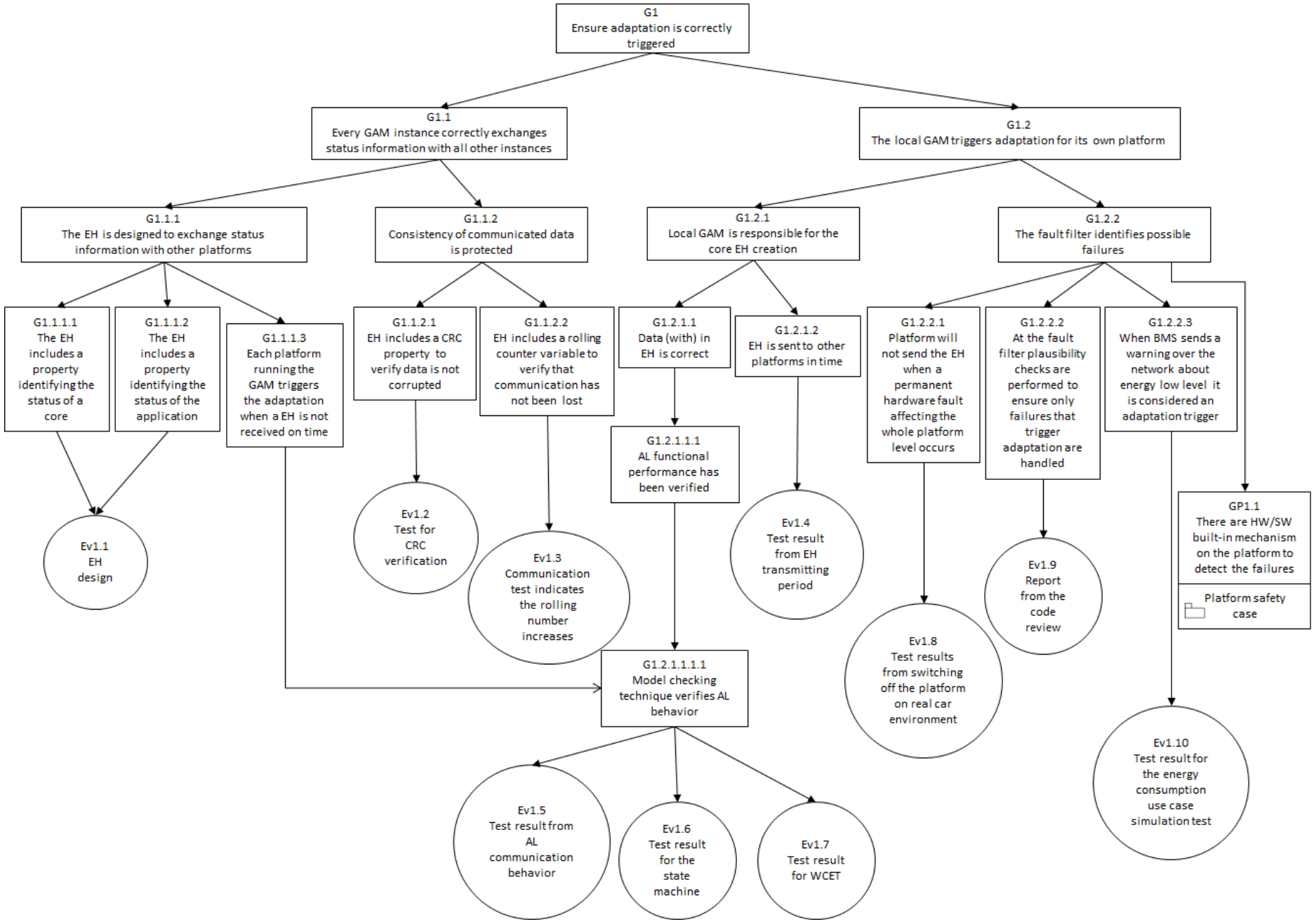

Fig. 8: Excerpt of Safety Case

REFERENCES

[1] P. Bieber, E. Noulard, C. Pagetti, T. Planche, and F. Vialard, “Preliminary design of future reconfigurable IMA platforms”, ACM SIGBED Rev. 6, 3, Article 7, October 2009.

[2] P. Cuenot, “Deliverable D3.3.2 – Requirement specification for a multi-class ECU concept”, SAFE project, 2014, http://www.safe-project.eu/SAFE-Publications/SAFE_D3.3.2.pdf.

[3] G. Weiss, F. Grigoleit, and P. Struss, "Context modeling for dynamic configuration of automotive functions," 16th International IEEE Conference on Intelligent Transportation Systems - (ITSC), pp. 839-844, 2013.

[4] R. De Lemos, H. Giese, H. A. Muller, M. Shaw, J. Andersson et al., “Software Engineering for Self-Adaptive Systems: A Second Research Roadmap,” Software Engineering for Self-Adaptive Systems, 7475, Springer, pp.1-26, 2013.

[5] V. Lopez-Jaquero, F. Montero, E. Navarro, A. Esparcia, and J.A. Catal'n, "Supporting ARINC 653-based Dynamic Reconfiguration," Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), 2012, vol., no., pp.11,20, 20-24 Aug. 2012.

[6] H.Mubarak, and P. Gohner, "An agent-oriented approach for self-management of industrial automation systems," Industrial Informatics (INDIN), 8th IEEE International Conference, pp. 721-726, 2010.

[7] J. Floch, C. Fra, R. Fricke, K. Geihs, M. Wagner et al., “Playing MUSIC - building context-aware and self-adaptive mobile applications”. Software: Practice and Experience 43(3), pp. 359-388, March 2013.

[8] P. Hofmann, and S. Leboch, “Evolutionäre Elektronikarchitektur für Kraftfahrzeuge (Evolutionary Electronic Systems for Automobiles)“, it-Information Technology, pp. 212–219, 2005.

[9] M. Dinkel and U. Baumgarten. "Self-Configuration of Vehicle Systems - Algorithms and Simulation”, in WIT '07: Proceedings of the 4th International Workshop on Intelligent Transportation 2007, pp. 85-91.

[10] R. Anthony, A. Rettberg, D.-J. Chen, I. Jahnich, G. de Boer et al., “Towards a Dynamically Reconfigurable Automotive Control System Architecture”, IESS, volume 231 of IFIP. Springer, pp.71-84, 2007.

[11] H. Seebach, F. Nafz, J. Holtmann, J Meyer, M. Tichy et al., “Designing Self-Healing in Automotive Systems”, Proceedings of the 7th International Conference on Autonomic and Trusted Computing (ATC), 2010.

[12] K. Höfig, M. Armbruster, and R. Schmid, “A vehicle control platform as safety element out of context”, HiPEAC Computing Systems Week, May 2014.

[13] K. Heckemann, M. Gesell, T. Pfister, K. Berns, K. Schneider et al., “Safe automotive software”, in Knowledge-Based and Intelligent Information and Engineering Systems, Springer, pp. 167–176, 2011.

[14] D. Schneider and M. Trapp, "Conditional Safety Certification of Open Adaptive Systems" ACM Trans. Auton. Adapt. Syst. 8, p. 8, 2013.

[15] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, “Relax: Incorporating uncertainty into the specification of self-adaptive systems”, 17th IEEE International Requirements Engineering Conference. RE, pp. 79–88, 2009.

[16] J. Rushby, “A safety-case approach for certifying adaptive systems”, AIAA Infotech@ Aerospace Conference, pp. 1992, 2009.

[17] S. Getir, S. Gerasimou, B. Eberhardinger, and T. Vogel, “Assurances for Self-Adaptive Software Systems”, Report from the GI Dagstuhl Seminar 14433: Software Engineering for Self-Adaptive Systems, 2015.

[18] International Organization for Standardization (ISO), “ISO/DIS 26262: Road vehicles - functional safety”, 2011.

[19] R.S. Hanmer, “Patterns for Fault Tolerant Software”. Wiley series in software design patterns, ISBN-13: 978-0470319796, 2007.

[20] A. Armoush, “Design Patterns for Safety-Critical Embedded Systems”, PhD thesis, Aachen University, 2010.

[21] R. Mariani, P. Fuhrmann, and B. Vittorelli, “Cost-effective Approach to Error Detection for an Embedded Automotive Platform”, SAE World Congress & Exhibition, April 2006.

[22] J.C. Laprie, “Dependability: Basic Concepts and Terminology Springer-Verlag”, ISBN 0-387-82296-8, 1992.

[23] SafeAdapt project, http://www.safeadapt.eu.

[24] A. Pena, I. Iglesias, J. Valera, and A. Martin, “Development and validation of Dynacar RT software, a new integrated solution for design of electric and hybrid vehicles”, Technical article on integrated solutions for new vehicles designing, Presented at EVS 26, Los Angeles (USA), 2012.

[25] S. Sommer, A. Camek, K. Becker, C. Buckl, A. Zirkler et al., “RACE: A Centralized Platform Computer Based Architecture for Automotive Applications” IEEE International Electric Vehicle Conference (IEVC), 2013.

[26] C. Stellwag, T. Rosenthal, and S. Gandhi, “Isolation of Cores - Reduce costs of mixed-critical systems by using a divide-and-conquer startegy on core level”, WICERT workshop, 2013.

[27] EAST-ADL Association, http://east-adl.info/.

[28] B. Kamphausen, A. Stante, M. Zeller, and G. Weiss, “ERNEST - framework for the early verification and validation of networked embedded systems”, Embedded World 2013. Exhibition & Conference, 2013.