Deployment Calculation and Analysis for a Fail-Operational Automotive Platform

Klaus Becker, Bernhard Schätz, Christian Buckl
fortiss GmbH
Guerickestr. 25, 80805 Munich, Germany
Email: {becker, schaetz, buckl}@fortiss.org

Michael Armbruster
Siemens AG, Corporate Technology
Otto-Hahn-Ring 6, 81739 Munich, Germany
Email: michael.armbruster@siemens.com

***Abstract*—In domains like automotive, safety-critical features are increasingly realized by software. Some features might even require fail-operational behavior, so that they must be provided even in the presence of random hardware failures. A new fault-tolerant SW/HW architecture for electric vehicles provides inherent safety capabilities that enable fail-operational features.**

In this paper, we introduce a formal model of this architecture and an approach to calculate valid deployments of mixed-critical software-components to the execution nodes, while ensuring fail-operational behavior of certain components. Calculated redeployments cover the cases in which faulty execution nodes have to be isolated. This allows to formally analyze which set of features can be provided under decreasing available execution resources.

Keywords*—*Fault-Tolerance; Fail-Operational; Deployment;

I. Introduction and Motivation

Embedded systems are often operated in safety-critical environments, in which unhandled faults could cause harmful system failures. Hence, safety-critical systems have to react on faults properly. Many current safety-critical systems for mass-markets, like vehicles, handle faults by invalidating faulty data and avoiding harm by going into a fail-safe state. However, this may cause the loss of provided features. This is not acceptable for features that require fail-operational behavior.

To increase their dependability, systems must be able to resume affected features without any service interruption. If system resources get lost due to hardware failures, runtime-reconfiguration can be applied to efficiently use the remaining resources. As the remaining resources may become insufficient to provide the full set of features, the explicit deactivation of some features would allow to keep alive the subset of features with the highest demand with respect to safety, availability and reliability. We use these terms as defined in [1].

However, in current automotive E/E architectures, reconfiguration is substantially restricted by Electronic Control Units (ECUs) tailored to their provided features, and inflexible communication buses. This heterogeneity prevents a system-wide mechanism to increase the feature reliability by resuming relevant software on other ECUs after hardware failures. Additionally, the integration of new features becomes more and more complicated due to increasing feature interactions. These and more challenges are for instance discussed in [2]. It is stated that a substantial revision of the vehicles HW/SW architecture can reduce its complexity to an adequate level.

We propose a new centralized HW/SW platform for vehicles, capable to overcome the mentioned shortcomings. The platform provides inherent safety properties and supports fail-operational features without requiring mechanical fallbacks. To avoid harm, faulty hardware is isolated from the remaining system. Affected software is resumed on intact hardware. Extensions to the feature-set after sale are supported in a Plug-and-Play manner.

In this paper, we address the calculation and analysis of the deployment of software components to the execution nodes inside the proposed architecture. To provide fail-operational features, software components are deployed redundantly. However, with a rising number of software and hardware units, this configuration becomes more and more complex and hard to manage manually. We therefore provide an automated configuration support for deployment decisions, ranging from a semi-automated to a fully-automated approach. Our approach is based on a formal system model and a set of formal constraints that describe the validity of deployments with respect to the safety-concept. Model and constraints characterize an arithmetic problem that can be solved by SMT-solvers.

The main contribution is an approach to calculate and analyze different reconfigurations of the deployment to become active after execution nodes become isolated. The set of active software components – and thus also the set of provided features – is automatically reduced when the remaining system resources become insufficient to provide the initial set of components. Components are deactivated based on their priorities, which can either be assigned manually or derived automatically. Our approach allows to formally analyze at design-time if the desired system and feature properties can be fulfilled, like which set of features can still be provided after one or multiple isolations. Analyzing the deactivations of single features allows to analyze the entire system degradation.

In section II we present the basic concepts of the proposed platform. Section III shows the main contribution of this paper, which is a formal model and a constraint-based approach to calculate valid deployments and to analyze which features can be provided after isolations of execution nodes. The applicability is shown by a little example from the automotive domain. Related work is discussed in section IV and the conclusion and future work is given in section V.

II. System Architecture and Safety Concept

A. System Architecture

In this paper, we discuss the deployment calculation for a scalable and uniform platform that was developed with the aim to reduce the complexity of automotive HW/SW architectures. The main platform characteristics are:

- A simplified hardware & network structure with a scalable set of execution nodes.
- A homogeneous communication system for the highly available transfer of critical real-time data based on industrial standards (e.g., Ethernet).
- Integrated mechatronics components (smart actuators and sensors), such as wheel hub motors with integrated steering, braking and damping.
- A runtime environment (RTE) that can execute both highly available, safety-critical software as well as non-safety-critical functions side-by-side.
- Plug-and-play capability to update or extend the vehicles safety-critical features by retrofit new software and modern sensors/actuators after purchase.

The vehicles hardware architecture is composed by a scalable set of central execution nodes (also called *Duplex Control Computers* (DCCs)) and a set of peripheral execution nodes providing the physical sensing and actuating (also called Smart-Aggregates). The DCCs are connected to each other and to the Smart-Aggregates by redundant switched Ethernet-Links. The DCCs assemble the *Central Platform Computer* (CPC). We assume homogeneous DCCs for flexibility in the deployment. Fig. 1 shows an example system architecture.

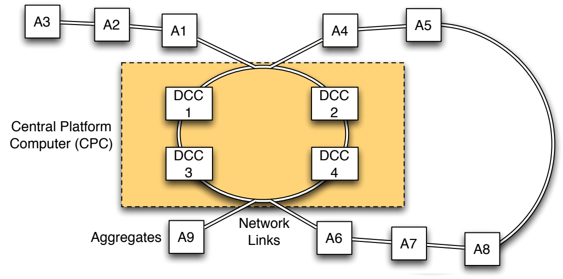

Fig. 1. Example instance of the proposed hardware architecture

The proposed system has two different power supplies *red* and *blue*. Each execution node is supplied by either the red or the blue one. Hence, if one power-supply gets lost, only a subset of the execution nodes gets lost and the residual nodes can continue the operation. These basic properties have also been described in [3]. As scheduling policy, we follow the concept of logical execution times [4], meaning that the software components are executed within *cycles*. Each execution node provides a certain budget of time per cycle that can be used to execute application software components. In this paper, we assume a simplified model in which all software components are scheduled with the same rate in each cycle.

B. Fault-Model

According to ISO 26262 [5], we use the following terms. 1) *Fault*: abnormal condition that can cause an element or an item to fail, 2) *Error*: discrepancy between a computed, observed or measured value or condition, and the true, specified, or theoretically correct value or condition, 3) *Failure*: termination of the ability of an element to perform a function as required, 4) *Fault-Model*: representation of failure modes resulting from faults, and 5) *Random Hardware Failure*: failure that can occur unpredictably during the lifetime of a HW element and that follows a probability distribution.

In this paper, we consider random hardware failures that lead to isolations of execution nodes. According to e.g. [6], we further consider Fault-Containment Regions (FCR) as the set of subsystems that share one or more common resources and that can be affected by a single fault.

We consider the vehicle as a set of FCRs. All FCRs have precisely specified linking interfaces in the domains of time and value including a link-specific fault-model. We describe the link-specific fault-models per FCR as far as it is helpful to understand the deployment model, presented in section III. The relevant FCRs are 1) execution nodes including its communication-links to data sinks, 2) application software components including its communication-links to data sinks, and 3) power supplies.

We define for each FCR a fault-model with the states *correct* z_c, *faulty* z_{ft}, *faulty but passivated* z_{fp} and *faulty out of control* z_{ooc}. A random hardware failure with a failure-rate λ leads to the transition from z_c to z_{ft}. The failure detection and passivation mechanisms within the faulty FCR but also within the receiving FCR lead to a transition from z_{ft} to z_{fp}. A passivated FCR does no longer harm the system operation. Anyhow, the functionality of the passivated FCR will no more be available. Only in case that the failure cannot be detected neither by the faulty FCR itself nor by the receiving FCR, we assume that this faulty FCR behaves fully out of control and thus, a correct system-operation can no longer be ensured. Out of control z_{ooc} means that the faulty FCR can neither be passivated nor controlled. With regard to ISO26262 this is equivalent to the fact that any safety-goal can no more be reached.

In this paper, we assume a state-transition time of $0s$ and a sufficient failure detection coverage. Sufficient means that the probability of any FCR to be in state z_{ooc} will be acceptably low to meet the quantitative safety-requirements of the ISO26262 [5]. Only if these assumptions are true, the deployment-considerations shown later in section III can be applied in a reasonable manner.

C. Safety & Redundancy Concept

Fault-tolerance is the ability of a system to maintain control objectives, despite the occurrence of a fault [7]. To achieve this, we deploy multiple instances of application software components in a redundant manner to the execution nodes. This enables the system to absorb loss of execution nodes and results in features being fail-operational, meaning that features can continue operation in the presence of a limited number of random hardware failures.

In the safety concept of the proposed platform, application software components (ASWCs) are grouped to so called *ASWC-Clusters*. These clusters get deployed to the execution nodes of the system. Those ASWCs belong to the same Cluster that have the same *Automotive Safety Integrity Level* (ASIL) and the same requirements to behave *fail-operational*.

Each ASWC has multiple safety goals, while each safety goal has an assigned *fault-tolerance time* (FTT). The smallest of these $FTTs$ is the so called $minFTT$ of an ASWC. Likewise, the $minFTT$ of an ASWC-Cluster is the smallest $minFTT$ of the ASWCs that are mapped to this cluster.

Each cluster has at least one deployed instance that is active as a so called *master*. If the cluster is required to be fail-operational, a second instance is deployed as a hot-standby or cold-standby *slave* (also known as hot/cold spare). The decision to create a hot- or a cold standby slave depends on the $minFTT$ of the cluster compared to the *fault-recovery time* (FRT) of the proposed platform.

We neglect here the time that is required to switch a cold-standby slave to become a master. With the proposed platform, a maximum switchover time can be verifed. We actually aim on a switchover-time of max 50ms. In this paper we assume the FRT to be a defined constant as it can be shown that a max FRT can be proven. Due to asynchronities and different fault-detection times depending on the faulty FCR, the actual FRT can be less than the maximum value we assume herein.

Depending on the required level of fail-operationality, additional inactive instances of a cluster are deployed, meaning that they are only in memory but not executed. Hence, we differ between *activations* (active deployments, ASWCs are executed) and *allocations* (inactive deployments, ASWCs are not executed, only in memory). An inactive allocation may become active if this is required after isolations.

Different constraints have to be fulfilled by a deployment to be valid. For instance, if an ASWC-Cluster has a master and a hot-standby slave, master and slave have to be deployed onto two execution nodes with different power-supplies to avoid that both instances get lost simultaneously when a power-supply fails. If the execution node of the master gets isolated, the slave becomes the new master and if required, a passive instance becomes the new hot-standby slave.

III. Deployment Calculation and Analysis

We define the system properties and the deployment problem as shown in the following sections.

A. Formal System and Deployment Model

Definition 1 *A Vehicle* $\mathbb{V} = \langle F, S^A, H^A, \Phi\rangle$ *comprises a set of* Functional Features F*, an* Application Software Architecture S^A*, an* Execution Hardware Architecture H^A *and a* Configuration Φ.

Definition 2 *An Application Software Architecture* $S^A = \langle S, SC\rangle$ *is composed by a set* $S = \{s_1, ..., s_n\}$ *of Application Software Components (ASWCs) and a set* $SC = \{sc_1, ..., sc_q\}$ *of ASWC-Clusters with* $sc_i \subseteq S$ *while* $\forall i, j : sc_i \cap sc_j = \emptyset$ *and* $\bigcup_{i=1}^{q} sc_i = S$*. We describe the mapping of* $s \in S$ *to* $sc \in SC$ *with* $\alpha(s) \rightarrow \{sc_i \in SC \mid sc_i \textit{ contains } s\}$ *and* $\alpha(sc) \rightarrow \{s_i \in S \mid s_i \textit{ is mapped to } sc\}$.

Definition 3 *The set of functional features* $F = \{f_1, ..., f_m\}$ *contains the features of the vehicle that can be recognized by the user. A feature is realized by one or more ASWCs and the involved Sensors and Actuators, while each ASWC contributes to realize one or more features. For* $s \in S$ *and* $f \in F$*, we define this relationship as* $\chi(s) \rightarrow \{f_i \in F \mid s \textit{ contributes to realize } f_i\}$ *and* $\chi(f) \rightarrow \{s_i \in S \mid f \textit{ is partly realized by } s_i\}$.

Definition 4 *An Execution Hardware Architecture* $H^A = \langle E, L\rangle$ *comprises execution nodes* E *and communication links* $L = E \times E$ *between these nodes. The set of execution nodes* $E = E^C \cup E^A$ *is composed by a set of central execution nodes* $E^C = \{e_1, ..., e_k\}$ *and a set of peripheral Smart-Aggregate nodes* $E^A = \{e_{k+1}, ..., e_l\}$ *with attached physical Sensors and Actuators. The set* E^C *is also called the* Central Platform Computer *(CPC).*

Definition 5 *The Configuration* $\Phi = \langle\delta_P(SC), \delta_A(SC), \delta(SC)\rangle$ *defines how ASWC-Clusters* SC *are deployed to execution nodes* E*, either passively (*δ_P*) or actively (*δ_A*). For* $sc \in SC$*, we define* $\delta_P(sc) \rightarrow \{e_i \in E \mid sc \textit{ is in memory of } e_i\textit{, but not executed on } e_i\}$*,* $\delta_A(sc) \rightarrow \{e_i \in E \mid sc \textit{ is in memory of } e_i \textit{ and executed on } e_i\}$ *and* $\delta(sc) = \delta_A(sc) \cup \delta_P(sc)$.

Our deployment approach can either be applied to ASWCs or to ASWC-Clusters. The motivation to think in Clusters and not in single ASWCs is that the definition of Clusters reduces the complexity with regard to the amount of combinations to be considered for deployment and master-slave switchovers. Furthermore, the ASWCs within a Cluster have a kind of stronger binding to each other. Thus, we aim on a deployment of ASWCs which are bound to one cluster within the same ECU. An example for a binding quality is data-transport delay.

ASWCs might contain invisible sub-components and internal communication channels. We don't model external communication channels between ASWCs in this paper for simplicity.

B. Fixed Properties of the Deployment Model

Each ASWC $s_i \in S$ is defined by several properties. Property $wcet(S) \rightarrow \mathbb{N}^+$ defines the *Worst-Case Execution Time*. Property $asil(S) \rightarrow \{0..4\}$ defines the *Automotive Safety Integrity Level* (ASIL) of an ASWC [0: Quality-Management (QM), 1: ASIL-A, 2: ASIL-B, 3: ASIL-C, 4: ASIL-D]. Property $failOp(S) \rightarrow \mathbb{N}_0$ defines the fail-operational level [0: non fail-operational, n: s_i has to be provided after n isolations]. The minimum of the fault-tolerance times of an ASWC for its different safety goals is defined by $minFTT(S) \rightarrow \mathbb{N}^+$.

As defined in section II-C, the vehicle property $frt(\mathbb{V}) \rightarrow \mathbb{N}^+$ defines the *fault-recovery time* of the vehicle $\mathbb{V}$. The frt has influence on if the slaves are deployed as hot or as cold-slaves, depending on their $minFTT$.

For execution nodes $e \in E$, the following properties are defined. The property $totalTimeBudget(E) \rightarrow \mathbb{N}^+$ defines the budget of time that is provided in each cycle to execute the ASWCs. We assume here that ASWCs are executed in every cycle. The property $powerSupply(E) \rightarrow \{0, 1\}$ defines the power supply of the execution node [0: Blue, 1: Red]. Finally, the property $isolated(E) \rightarrow \{0, 1\}$ defines if the execution node $e_i \in E$ is isolated in the current solution instance. We do not model the amounts of required and provided volatile and non-volatile memory here for simplicity. These are handled in a similar manner than the WCET and the time-budget.

C. Solution Properties of the Model

In this section we describe the model-properties that represent the solution of the deployment problem.

The properties of ASWC-Clusters $sc \in SC$ depend on the mapped ASWCs. Properties $asil(SC) \rightarrow \{0..4\}$ and $failOp(SC) \rightarrow \mathbb{N}_0$ define the ASIL and the fail-operational level of a cluster. It's ensured by constraints that $\forall s_i \in \alpha(sc) : asil(sc) = asil(s_i)$ and $failOp(sc) = failOp(s_i)$. Property $minFTT(SC) \rightarrow \mathbb{N}^+$ is the minimum of all the $minFTT(s_i)$ for $s_i \in \alpha(sc)$. The property $sumWcets(SC)$ is defined to be equal to $\sum_{s_i \in \alpha(sc)} wcet(s_i)$.

For execution nodes $e \in E$, $usedTimeBudget(E) \rightarrow \mathbb{N}_0$ is defined to be equal to $\sum_{sc_j \in SC \mid e \in \delta_A(sc_j)} sumWcets(sc_j)$, which is the sum of the $wcet(s)$ of those ASWCs that are active in the schedule on node e. A constraint ensures that $\forall e \in E : usedTimeBudget(e) \leq totalTimeBudget(e)$.

Notice that the decision if an ASWC-Cluster instance becomes a master or a hot-standby slave is done at runtime by a Platform-Management component of the RTE of the proposed vehicle platform. This is, because there are also other reasons beside node-isolations that may lead to the deactivation of a master. Hence, the calculated master/slave deployments as shown in this paper are not used as predefined runtime-configuration, but at design-time to statically analyze the fail-operational runtime-behavior. It can be analyzed under which circumstances it is possible at runtime to keep a master resp. a slave alive in the presence of faults that lead to node-isolations.

D. Deployment Constraints and Problem Solving

To define the set of valid deployments, we setup an arithmetic model of the system properties and the deployment constraints. We implemented the deployment calculation and analysis by defining the model with arithmetic calculations and simple functions like *Sums*, *Implications* and *if/then/else* relations. We do not list the detailed constraints in this paper for space reasons. The model can be solved for instance by an SMT-Solver like Z3 [8].

E. Example of an Initial Deployment

In this section we show an application of our approach on a simplified example from the automotive domain. Consider the following functional features and ASWCs:

Feature f_i	**ASWCs** s_i of $\chi(f_i)$	**asil**(s_i) / **failOp**(s_i) / **wcet**(s_i) in ms
f_1 : Infotainment	s_1 : Infotainment	QM / 0 / 2
f_2 : Energy-Management	s_2 : RemainingDrive-RangeEstimation	A / 0 / 0.7
	s_3 : EnergyEfficiency-Assistant	A / 0 / 0.3
f_3 : ADAS-A	s_4 : AdasSwc1	C / 0 / 1.7
	s_5 : AdasSwc2	D / 1 / 1
f_4 : ADAS-B	s_5 : AdasSwc2	D / 1 / 1
f_5 : Manual Driving	s_6 : ManualAccelerate	D / 3 / 1
	s_7 : ManuelBrake	D / 3 / 1
	s_8 : ManualSteer	D / 3 / 0.5

The features f_3 and f_4 are placeholders for some Advanced Driver Assistance Systems (ADAS), like an ACC or automatic parking. Notice that feature f_3 is realized by two ASWCs with different properties and ASWC s_5 contributes to realize two features f_3 and f_4. Feature f_3 is non fail-operational, as not all $s_i \in \chi(f_3)$ are fail-operational, but f_4 is fail-operational.

In this example, five ASWC-Clusters $\{sc_1, ..., sc_5\}$ are established. Due to the constellation of the properties $asil(s_i)$ and $failOp(s_i)$, the ASWC-Clusters are: $\alpha(sc_1) = \{s_1\}$, $\alpha(sc_2) = \{s_2, s_3\}$, $\alpha(sc_3) = \{s_4\}$, $\alpha(sc_4) = \{s_5\}$ and $\alpha(sc_5) = \{s_6, s_7, s_8\}$. Notice that ASWC s_5 is only in one cluster, although it contributes to two features.

Considering a CPC with 4 execution nodes (DCCs) as shown in Fig. 1, a valid initial deployment for the example is shown in Fig. 2. The colors (red/blue) of the execution nodes denote their attached power-supply. We assume here that $minFTT(s_i) < frt(\mathbb{V})$ for all fail-operational ASWCs. Hence, hot-standby slaves are required. As provided execution time of the execution nodes per cycle, we assume $totalTimeBudget(e_i) = 4ms$. As visible in Fig. 2 at the values of $usedTimeBudget(e_i)$, no time-budget of any execution node is exceeded.

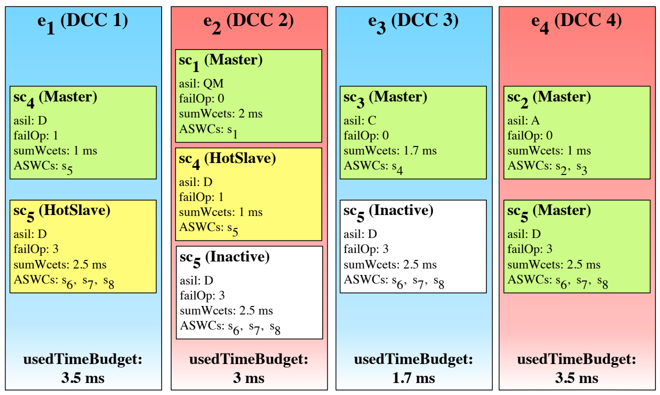

Fig. 2. Initial deployment for the example

The Z3 SMT-Solver [8] calculated the initial deployment in 125ms on a 2 GHz Core i7. This is the time for the check()-operation, not the time for setting up the model and constraints.

F. Reconfigurations after Isolations

Let $E^C_f \subset E^C$ be the set of isolated execution nodes. For all $e_i \in E^C_f$, we set $isolated(e_i) = 1$. It is ensured by constraints that no ASWC-Cluster is activated anymore on one of the isolated execution nodes.

Definition 6 *A Platform-Availability-Graph (PAG) is a directed acyclic graph $G = (V, E)$. Each vertex V represents a set of alive central execution nodes $E^C_a = E^C \setminus E^C_f$. The edges E describe a transition between two vertexes, meaning that some $e_i \in E^C$ move from E^C_a to E^C_f. A transition happens due to an isolation or if a power-supply disappears.*

Fig. 3 shows an example CPC containing 4 central execution nodes (DCCs) and the two power-supplies (red and blue).

When considering only one fault, the PAG looks like shown in Fig. 4. The vertexes are labeled with the Ids i of the alive nodes $e_i \in E^C_a$. The edges are labeled with the Id i of that $e_i \in E^C_f$ which has recently been isolated resp. with the power-supply (R, B) that has recently been broken down.

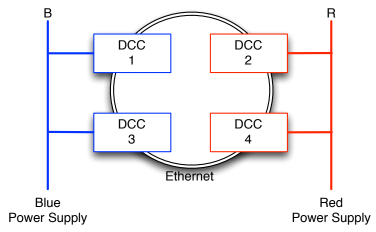

Fig. 3. An example Central Platform Computer (CPC) with 4 DCCs

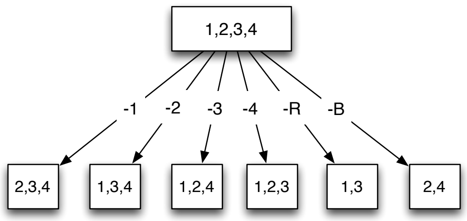

Fig. 4. Example PAG considering only one fault

The validity of deployments after a transition in the PAG is ensured by arithmetic constraints. For instance, these constraints ensure that the mapping of ASWCs to clusters is not changed during a PAG-transition. Furthermore, it is ensured that the previous allocations or activations of ASWC-Clusters to execution nodes are not changed unnecessarily during a PAG-transition. To do this, some solution properties of the former deployment are used as fixed properties for the follow-up deployment, when a PAG-transition is calculated.

To cover deactivation scenarios that might be required after isolations of central execution nodes, each $sc \in SC$ has additionally the following properties:

- $hotStandbySlaveReq(SC) \rightarrow \{0,1\}$: indicates if a hot-standby *slave* is required. The valuation is derived by considering $minFTT(sc)$ and $frt(\mathbb{V})$
- $hotStandbySlavePresent(SC) \rightarrow \{0,1\}$: indicates if a required hot-standby *slave* can be established
- $masterPresent(SC) \rightarrow \{0,1\}$: indicates if the *master* can be established or not

In order to decide about the deactivation order for the ASWC-Clusters, each ASWC-Cluster has assigned the properties $prioPointsMaster(SC) \rightarrow \mathbb{N}^{+}$ and $prioPointsHotSlave(SC) \rightarrow \mathbb{N}^{+}$ storing priorities of actively deployed instances of the cluster.

We calculate the sum of the priorities of all active instances of ASWC-Clusters and use these priorities and their sum to construct an order in which the instances of the clusters should be deactivated in case system resources become insufficient. We do this by maximizing the sum of the priorities. The clusters with the lowest priority get deactivated first. We derive the priorities depending on $asil(SC)$ and $failOp(SC)$. We set $prioPointsMaster(sc) = asil(sc) + failOp(sc) + 2$ and $prioPointsHotSlave(sc) = asil(sc) + failOp(sc) + 1$. Hence, clusters with lowest ASIL will get deactivated first. However, the priorities could also be set differently.

G. Example of a Deployment after an Isolation

Fig. 5 shows the follow-up deployment for the case that DCC1 becomes isolated in the initial deployment (cf. Fig. 2).

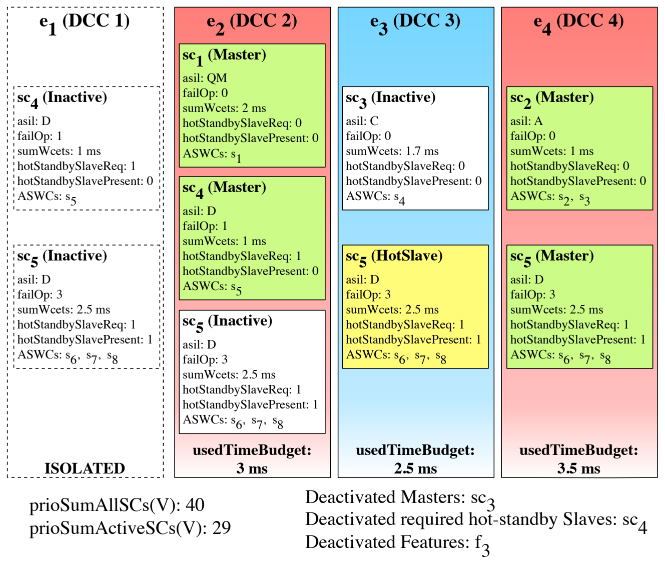

Fig. 5. Follow-up deployment after DCC1 has been isolated

While in the initial deployment all clusters can be deployed as required, after the isolation of e_1 (= DCC 1) the master of cluster sc_4 gets lost and its slave on e_2 becomes the new master. As $failOp(sc_4) = 1$, no new slave is created because it's not required that sc_4 is still present after the next isolation. Furthermore, the slave of cluster sc_5 get lost. As $failOp(sc_5) = 3$, an inactive instance of sc_5 must be activated to serve as new slave to prepare for the next isolation. The new slave of sc_5 can only be activated on e_3, not on e_2, because it is not allowed that master and slave depend on the same power-supply. However, to be able to execute sc_5 on e_3, sc_3 has to be deactivated as the sum of the WCETs of sc_3 and sc_5 would exceed the time-budget of e_3. The deactivation of sc_3 forces the deactivation of feature f_3, as $\alpha(sc_3) = \{s_4\} \subseteq \chi(f_3)$.

The sum of priority points in the initial solution was 40. The loss of the master of sc_3 and the slave of sc_4 forces a loss of 11 priority points ($prioPointsMaster(sc_3) = 5$, $prioPointsHotSlave(sc_4) = 6$). Hence, without DCC1 only 29 priority points can be provided by the system (cf. Fig. 5).

When this procedure is continued by isolating more DCCs in arbitrary order, the cluster sc_5 always has a master instance, even if only one DCC is left. This is important as $failOp(sc_5) = 3$.

To calculate the follow-up deployment, 1.1s were spent for checking 11 models that are unsatisfiable for priority point sums 40 to 30, plus 90ms for checking the valid solution.

The designer can analyze the system's fail-operational behavior by considering the set of deactivated features for each situation. This allows to formally analyze if all desired system and feature properties can be fulfilled, without executing the system. The initial deployment can also be changed manually in order to analyze the systems feature availability depending on different initial deployments.

IV. Related Work

In this section, we discuss related work of deployment approaches with focus on safety and fail-operationality.

In [9], the authors show an approach to analyze graceful degradation. They use a utility function to measure the set of active features. This can be seen as quite similar to our sums of priorities. To reduce complexity, they group components by defining subsystems based on the interfaces of components. We group components by their dependability requirements. This allows separation of mixed-critical components. The main differences are that they consider a fail-silent fault-model, while we consider fail-operational behavior of features. Furthermore, we focus more explicitly on deployment constraints that ensure fail-operational behavior. Another difference is that we consider the explicit deactivation of components to be able to keep alive other components that are required to behave fail-operational. They consider a fixed hardware configuration, while we consider a HW-Architecture whose provided resources decrease after random hardware failures due to execution node isolations.

In [10], the authors introduce a design methodology for safety-critical systems, called SCRAPE (Safety-Critical Real-Time APlications Exploration). In addition the *fault-tolerant data flow* (FTDF) model of computation is introduced. The SCRAPE design flow has 6 main steps. Our work is mainly related to steps 4 and 5, namely the specification of *Fault Behavior and Mapping Constraints* as well as the calculation of a *Fault-Tolerant Embedded Software Deployment*. However, with SCRAPE fail-silent execution platforms are addressed, while we focus on ensuring fail-operational features. Additionally, we address the analysis about which feature-set can be provided after certain random hardware failures.

In [11], fault-tolerant deployments with focus on the trade-off between Performance and Reliability are optimized using a MILP-Solver. However, the approach does not consider mixed criticalities explicitly, and also at most 1 replication is supported due to the single node failure model. The analysis of deployments after hardware-faults is also not considered.

In [12], a dynamically reconfigurable vehicle control platform supporting fault-tolerance is described. The reactivation- and reallocation algorithms are executed during runtime as one of the platform's core-algorithms. However, only two clusters of application components are supported. It is ensured that at any time the most important cluster is executed with a fail-operational behavior and that the last non-faulty execution node executes the most important cluster. Anyhow, the presented work is limited to only two clusters whereas one does not have any fail-operational requirement.

V. Conclusion and Future Work

In this paper, we have shown a formal approach to calculate deployments of mixed-critical functional features in a new HW/SW architecture for vehicles. The work allows to analyze the fail-operational behavior of features in the presence of random hardware failures. It can be analyzed which features can be uphold depending on the available set of execution nodes. We defined a formal system model and ensured deployment constraints by setting up an arithmetic input model for a SMT-Solver, that calculates valid deployments.

As future work, we are going to include communication channels between ASWCs into the model. In this context, we are going to select optimal channels out from a set of channel-candidates and optimize the deployment respective to minimize the required network bandwidth. An end-2-end timing-analysis with focus on the deployment options is also planned.

Furthermore, we want to treat the integration of new software components into existing deployments during the use case of extensions of the vehicle by new functional features in a plug-and-play manner. Finally, we want to evaluate the scalability of our approach based on the system layout of a concept car that we construct.

Acknowledgment

This work is partially funded by the German Federal Ministry for Economic Affairs and Energy (BMWi) under grant no. 01ME12009 through the project RACE (Robust and Reliant Automotive Computing Environment for Future eCars) (http://www.projekt-race.de).

References

[1] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[2] S. Chakraborty, M. Lukasiewycz, C. Buckl, S. Fahmy, N. Chang, S. Park, Y. Kim, P. Leteinturier, and H. Adlkofer, "Embedded systems and software challenges in electric vehicles," in *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2012, pp. 424–429.

[3] S. Sommer, A. Camek, K. Becker, C. Buckl, A. Knoll, A. Zirkler, L. Fiege, M. Armbruster, and G. Spiegelberg, "Race: A centralized platform computer based architecture for automotive applications," in *IEEE Vehicular Electronics Conference / Int. Electric Vehicle Conference (VEC-IEVC)*, 2013.

[4] T. Henzinger, B. Horowitz, and C. Kirsch, "Giotto: A time-triggered language for embedded programming," in *Embedded Software*. Springer, 2001, pp. 166–184.

[5] International Organization for Standardization, "ISO/DIS 26262-1 - Road vehicles - Functional safety, Part 1 Glossary," Technical Committee 22 (ISO/TC 22), Geneva, CH, Tech. Rep., Nov. 2011.

[6] H. Kopetz, "Fault containment and error detection in the time-triggered architecture," in *Int. Symposium on Autonomous Decentralized Systems (ISADS)*. IEEE, 2003, pp. 139–146.

[7] M. Blanke, M. Staroswiecki, and N. E. Wu, "Concepts and methods in fault-tolerant control," in *Proceedings of the 2001 American Control Conference*, vol. 4. IEEE, 2001, pp. 2606–2620.

[8] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340, 2008.

[9] C. Shelton, P. Koopman, and W. Nace, "A framework for scalable analysis and design of system-wide graceful degradation in distributed embedded systems," in *Int. Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)*. IEEE, 2003, pp. 156–163.

[10] C. Pinello, L. P. Carloni, and A. L. Sangiovanni-Vincentelli, "Fault-tolerant distributed deployment of embedded control software," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 5, pp. 906–919, 2008.

[11] B. Boone, F. De Turck, and B. Dhoedt, "Automated deployment of distributed software components with fault tolerance guarantees," in *6th Int. Conf. on Software Engineering Research, Management and Applications (SERA)*. IEEE, 2008, pp. 21–27.

[12] M. Armbruster, "Eine fahrzeugübergreifende x-by-wire plattform zur ausführung umfassender fahr-und assistenzfunktionen," Ph.D. dissertation, Institute for Avionics Systems (Institut für Luftfahrtsysteme, ILS), University of Stuttgart, 2009.