A Formal Model for Constraint-Based Deployment Calculation and Analysis for Fault-Tolerant Systems

Klaus Becker[1], Bernhard Schätz[1], Michael Armbruster[2], and Christian Buckl[1]

[1] fortiss GmbH, An-Institut Technische Universität München,
Guerickestr. 25, 80805 München, Germany
{becker,schaetz,buckl}@fortiss.org
[2] Siemens AG, Corporate Research and Technologies,
Otto-Hahn-Ring 6, 81730 München, Germany
michael.armbruster@siemens.com

Abstract. In many embedded systems like in the automotive domain, safety-critical features are increasingly realized by software. Some of these features are often required to behave fail-operational, meaning that they must stay alive even in the presence of random hardware failures.

We propose a new fault-tolerant SW/HW architecture for electric vehicles with inherent safety capabilities that enable fail-operational features. In this paper, we introduce a constraint-based approach to calculate valid deployments of mixed-critical software components to the execution nodes. To avoid harm, faulty execution nodes have to be isolated from the remaining system. We treat the isolations of execution nodes and the required changes to the deployment to keep those software components alive that realize fail-operational features. The affected software components have to be resumed on intact execution nodes. However, the remaining system resources may become insufficient to execute the full set of software components after an isolation of an execution node. Hence, some components might have to be deactivated, meaning that features might get lost. Our approach allows to formally analyze which subset of features can still be provided after one or more isolations. We present an arithmetic system model with formal constraints of the deployment-problem that can be solved by a SMT-Solver. We evaluate our approach by showing an example problem and its solution.

Keywords: Fault-Tolerance, Fail-Operational, Mixed-Critical, Deployment, Dependability, SMT-Solver.

1 Introduction and Motivation

Many embedded systems are operated in safety-critical environments, in which unhandled faults could cause harmful system failures. This requires that those systems react on faults properly. However, handling faults by invalidating faulty data and going into a fail-safe state may cause the loss of some provided features. This is not acceptable for features that require fail-operational behavior.

D. Giannakopoulou and G. Salaün (Eds.): SEFM 2014, LNCS 8702, pp. 205–219, 2014.

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, the remaining resources should be used efficiently to keep alive those features with the highest demand with respect to safety, reliability and availability, as defined in [1]. For instance, if an execution node becomes faulty and has to be isolated from the remaining system, another execution node has to be able to provide that features that were provided by the faulty node. However, as the remaining system-resources may become insufficient to provide the full set of features, it may be needed to explicitly deactivate some low priority features. This results in a graceful degradation of the system.

We propose a new centralized HW/SW platform for vehicles, that provides inherent safety properties and supports fail-operational features without requiring mechanical fallbacks. In this paper, we address the calculation and analysis of the deployment of software components to the execution nodes inside the proposed architecture. However, with a rising number of software and hardware components, this deployment 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 describing the validity of deployments with respect to the safety-concept. Model and constraints characterize an arithmetic problem that can be solved for instance 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 2 we present the basic concepts of the proposed platform. Section 3 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 nodes. Section 4 contains an automotive example, evaluating the applicability of our approach. Related work is discussed in section 5 and the conclusion and future work is given in section 6.

2 Proposed System Architecture and Safety Concept

Fault-tolerance is the ability of a system to maintain control objectives despite the occurrence of a fault, while degradation of control performance may be accepted [2]. If a system should support fail-operational features, it has to be capable to absorb loss of execution nodes. We deploy multiple instances of software components redundantly 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 hardware failures, while ensuring the absence of harm to the users or the environment. In the following sections we briefly introduce the main ideas of the proposed platform, needed to follow the deployment concept and constraints.

2.1 System Architecture

We tackle the development of a scalable, uniform, open and thus easily expandable base platform with the aim to reduce the complexity of automotive HW/SW architectures. The basic principles of this platform have been already presented in [3] and [4].

The proposed platform 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 assemble the *Central Platform Computer* (CPC) and are connected to each other and to the Smart-Aggregates by redundant switched Ethernet-Links. The DCCs are homogeneous for flexibility in the deployment.

The proposed system has two different power supplies, named *red* and *blue*. Each execution node is supplied by either the red or the blue power supply. Hence, if one power-supply fails, only a subset of the execution nodes get lost and the residual nodes can continue the operation. As scheduling policy, we follow the concept of logical execution times [5], meaning that the software components are executed within fixed *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 every cycle.

2.2 Fault-Model and Safety Concept

In this paper, we focus on so called *random hardware failures*, as defined in the ISO 26262 [6] as failures that can occur unpredictably during the lifetime of a hardware (HW) element and that follows a probability distribution. If such a random hardware failure exists in an execution node, this node has to be isolated from the remaining system to avoid harm. Our proposed platform ensures the detection of random hardware failures with a sufficient failure detection coverage. Sufficient means that the probability to become out-of-control is acceptably low to meet the quantitative safety-requirements of the ISO26262 [6]. We focus on how to handle detected failures by performing adaptions to the deployment to meet the requirements w.r.t. fail-operationality. We assume a state-transition time of $0s$ from a faulty to an isolated state. Only if these assumptions hold, the deployment considerations shown later in section 3 can be applied. More information about the Fault-Model is also provided in [7].

In the safety concept of the proposed platform, application software components (ASWCs) are grouped into so called *ASWC-Clusters*. This is done to reduce the complexity of fault detection and handling mechanisms at runtime.

The ASWC-Clusters get deployed to the execution nodes. Those ASWCs are mapped to the same ASWC-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 FTT defines the time period that a component can fail to deliver its service without harming the safety goal. The smallest of these $FTTs$ is the so called $minFTT$ of an ASWC. The $minFTT$ of an ASWC-Cluster is the smallest $minFTT$ of the ASWCs that are mapped to this cluster.

Each ASWC-Cluster has at least one actively deployed instance in the initial deployment. If the cluster is required to be fail-operational, a second instance is deployed. In this case, the first instance is called the *master* and the second instance is called the *slave*. We distinguish between hot-standby and cold-standby *slaves* (also known as hot/cold spare). A hot-standby is active (executed in schedule), while a cold-standby is passive (only in memory, not executed in schedule). In the deployment, we consider this by distinguishing between *activations* (active deployments, ASWCs are executed) and *allocations* (inactive/passive deployments). The decision whether to create a hot- or a cold standby slave depends on the $minFTT$ of the ASWC-Cluster compared to the *fault-recovery time* (FRT) of the proposed platform. We assume the FRT to be a defined constant, as a maximum FRT can be shown because the platform ensures a worst-case time between fault-detection, confirmation and reconfiguration. In this paper, we neglect 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.

There exist several constraints for the deployment given by the safety concept. For instance, if an ASWC-Cluster has a master and a 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.

Depending on the required level of fail-operationality, meaning how many HW-failures have to be survived, additional inactive instances of a cluster are deployed. If the execution node of the master gets isolated, the slave becomes the master and if required, a passive instance becomes the new hot-standby slave. These mechanisms are presented in section 3 in a more detailed formal model.

3 Deployment Calculation and Analysis

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

3.1 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* $\alpha(s) \rightarrow \{sc_i \in SC \mid sc_i$ *contains* $s\}$ *and* $\alpha(sc) \rightarrow \{s_i \in S \mid s_i$ *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* $\chi(s) \rightarrow \{f_i \in F \mid s$ *contributes to realize* $f_i\}$ *and* $\chi(f) \rightarrow \{s_i \in S \mid f$ *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* $\delta_P(sc) \rightarrow \{e_i \in E \mid sc$ *is in memory of* e_i*, but not executed on* $e_i\}$*,* $\delta_A(sc) \rightarrow \{e_i \in E \mid sc$ *is in memory of* e_i *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 execution node. An example for a binding quality is data-transport delay.

Fig. 1 shows a visualization of the given definitions, based on an example. Two features are realized by overall three ASWCs, while the third ASWCs s_3 contributes to both features. The three ASWCs are mapped to two different ASWC-Clusters, depending on a property *failOp* that defines the level of required fail-operationality of an ASWC. As cluster sc_1 contains ASWCs that are not required to behave fail-operational, it is deployed only once ($\delta_A(sc_1) = \{e_1\}$). The other cluster sc_2 contains an ASWC that is required to behave fail-operational ($\alpha(sc_2) = \{s_3\}$ and $failOp(s_3) = 1$). Hence, this cluster is deployed twice with one active *Master* ($\delta_A(sc_2) = \{e_2\}$) and one passive *cold-standby slave* ($\delta_P(sc_2) = \{e_1\}$). If a *hot-standby slave* would have been required, then it would hold that $\delta_A(sc_2) = \{e_1, e_2\}$, $\delta_P(sc_2) = \emptyset$.

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

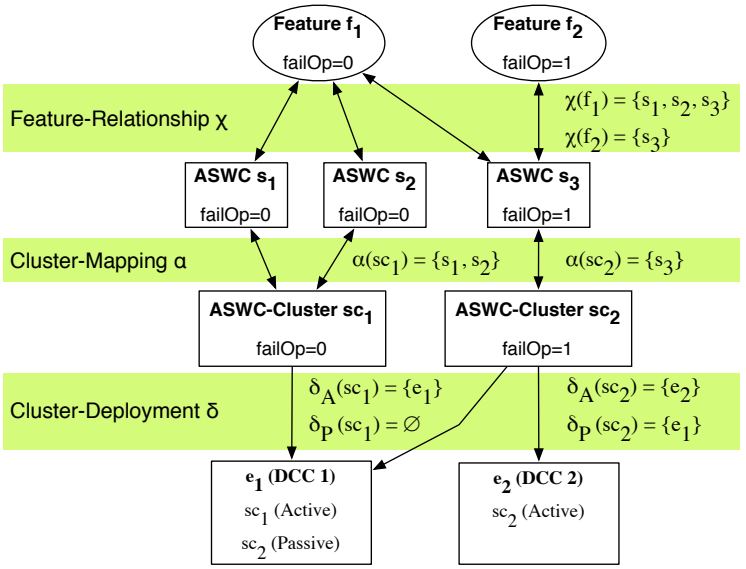


Fig. 1. Example for the definitions

3.2 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 2.2, the vehicle property $frt(\mathbb{V}) \rightarrow \mathbb{N}^+$ defines the *fault-recovery time* of the vehicle $\mathbb{V}$. The frt has influence on whether the slaves are deployed as hot or as cold-standby 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 as the WCET and the time-budget.

3.3 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 is 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 $s_i \in \alpha(sc)$. Property $sumWcets(SC)$ is defined to be equal to $\sum_{s_i \in \alpha(sc)} wcet(s_i)$. 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

The last two properties may change after isolations of execution nodes. Finally, each cluster has the properties $prioPointsMaster(SC) \rightarrow \mathbb{N}^+$ and $prioPointsHotSlave(SC) \rightarrow \mathbb{N}^+$ storing priorities of actively deployed instances of clusters. These are used to construct an order in which the cluster instances should be deactivated in case resources become insufficient. We derive the priorities depending on $asil(SC)$ and $failOp(SC)$ (cf. Listing 3). However, they could also be set in a different manner depending on the user's needs.

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 on execution node e. A constraint ensures that $\forall e \in E : usedTimeBudget(e) \leq totalTimeBudget(e)$.

On vehicle-level, the property $prioSumAllSCs(\mathbb{V}) \rightarrow \mathbb{N}$ is defined as the sum of the priorities of the actively deployed ASWC-Clusters in the initial deployment without any isolation. In addition, the property $prioSumActiveSCs(\mathbb{V}) \rightarrow \mathbb{N}$ is the sum of the priorities of all ASWC-Clusters $SC' \subseteq SC$ that are actively deployed in the current system situation with some isolations.

Finally, the following two properties define the solution matrices that contain the mapping of ASWCs S to ASWC-Clusters SC and the deployment of the ASWC-Clusters SC to the execution nodes E.

- $map(S, SC) \rightarrow \{0, 1\}$: Mapping of ASWCs $s \in S$ to ASWC-Clusters $sc \in SC$. [0: $s \notin \alpha(sc)$, 1: $s \in \alpha(sc)$].
- $deploy(SC, E) \rightarrow \{0, 1, 2, 3\}$: Deployment of ASWC-Clusters $sc \in SC$ (and it's ASWCs $s_i \in \alpha(sc)$) to execution nodes $e \in E$. [0: $e \notin \delta(sc)$, 1: $e \in \delta_P(sc)$, 2: $e \in \delta_A(sc)$ while sc is a *master* on e, 3: $e \in \delta_A(sc)$ while sc is a hot-standby *slave* on e]

Notice that the decision if an ASWC-Cluster instance becomes a master or a hot-standby slave is done dynamically at runtime by a Platform-Management component of the Runtime-Environment (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 respectively a slave alive in the presence of faults that lead to the isolation of execution nodes.

3.4 Basic Deployment Constraints

In this section we describe some exemplary constraints that limit the solution space of the calculated deployments to ensure the properties listed in section 2.2.

To define the constraints, we setup an arithmetic model. We use two conditional functions in the constraints. Function $Ite(I, T, E)$ has three parameters. The first parameter describes an *if-clause* I. If I is true, then the second parameter T is used in the constraint, else the third parameter E. The second function that we use is $Implies(I, T)$, which is true for $(\neg I \vee T)$. Both functions are provided by the SMT-Solver.

Listing 1 shows some basic constraints of the described deployment model.

```
1 ∀sc ∈ SC :
2 ∑_{e∈E} Ite(deploy(sc, e) ≠ 0, 1, 0) = failOp(sc) + 1
3
4 hotStandbySlaveReq(sc) = Ite(
5    And(failOp(sc) > 0, minFTT(sc) ≤ frt(𝕍)), 1, 0)
6
7 Implies( masterPresent(sc) = 1,
8              ∑_{e∈E} Ite(deploy(sc, e) = 2, 1, 0) = 1)
```

Listing 1. Some basic constraints

The constraint in line 2 ensures the correct number of allocations of ASWC-Clusters. Clusters with $failOp(sc) = n$ have to be allocated $n + 1$ times. Hence, $|\delta(sc)| = n + 1$.

Lines 4-5 show the constraint that defines when a hot-standby slave is required for a ASWC-Cluster. If the cluster contains fail-operational ASWCs and has a $minFTT$ smaller or equal than the vehicle's fault-recovery time (frt), then a hot-standby slave is required for that cluster.

The constraint in lines 7-8 controls the presence of the master for each cluster. If a master is present, it is ensured that it exists exactly once. To deactivate a master, $masterPresent(sc)$ has to become 0. This allows to give feedback that sc cannot be executed in the current solution.

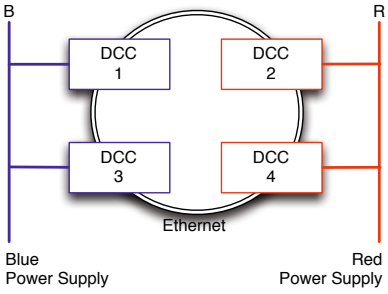
The hot-standby slaves are handled similarly by considering the property $hotStandbySlaveReq(SC)$. Additional constraints ensure for instance that if both the master and the hot-standby slave are present, then they have to be active on two execution nodes with different power-supplies.

3.5 Reconfigurations after Isolations

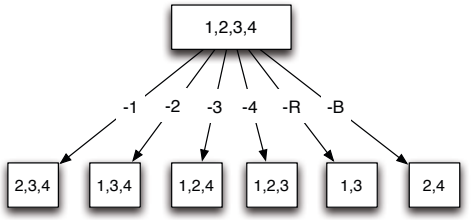
Let $E_f^C \subset E^C$ be the set of isolated execution nodes. For all $e_i \in E_f^C$, we set $isolated(e_i) = 1$. It is ensured by constraints that no ASWC-Cluster is activated anymore on one of the isolated execution nodes.

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 $E_a^C = E^C \setminus E_f^C$. The edges E describe a transition between two vertices, meaning that some $e_i \in E^C$ move from E_a^C to E_f^C. A transition happens due to an isolation or if a power-supply disappears.*

Fig. 2(a) shows an example CPC containing four central execution nodes (DCCs) and the two power-supplies (red and blue).



(a) An example Central Platform Computer (CPC) with 4 DCCs



(b) Example PAG considering only one fault

Fig. 2. Platform-Availability-Graph (PAG)

When considering only one fault, the PAG looks like shown in Fig. 2(b). The vertices are labeled with the Ids i of the alive nodes $e_i \in E_a^C$. The edges are labeled with the Id i of that $e_i \in E_f^C$ which has recently been isolated respectively with the power-supply (R, B) that has recently been broken down.

Fig. 3 shows how the deployment from Fig. 1 is reconfigured in case DCC 2 has to be isolated. The passive cold-standby slave of cluster sc_2 has to be activated, because the former master gets lost. Assuming that sc_1 and sc_2 cannot run simultaneously on e_1 due to resource constraints, sc_1 has to be passivated. This is allowed as sc_1 contains ASWCs that have no requirement to be active after a fault ($failOp = 0$). However, as sc_1 becomes passivated, feature f_1 cannot be provided anymore, because two of the three ASWCs that realize f_1 are passivated. Notice that all requirements concerning fail-operationality are met in this example.

We now show some formal constraints that describe the validity of follow-up deployments that become active after isolations of execution-nodes, forcing a transition in the PAG.

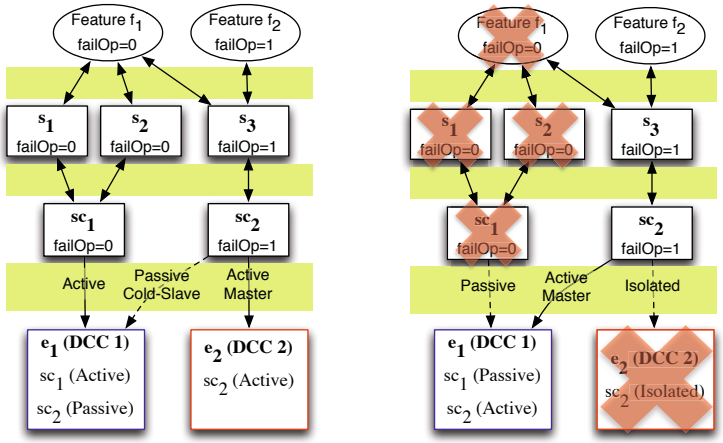


Fig. 3. Example of a gracefully degraded system after an isolation

The constraints shown in Listing 2 ensure that no present allocation or activation of an ASWC-Cluster changes unnecessarily during a PAG-transition. The notation $map_{prev}(s, sc)$ and $deploy_{prev}(sc, e)$ denote the mapping of ASWC to clusters respectively the deployment of clusters to nodes that were previously active before the PAG-transition.

```
1  ∀s ∈ S,    ∀sc ∈ SC :
2  Implies(map_prev(s, sc) = 1,    map(s, sc) = 1)
3
4  ∀sc ∈ SC,    ∀e ∈ E :
5  Implies(deploy_prev(sc, e) = 0,    deploy(sc, e) = 0)
6
7  ∀sc ∈ SC,    ∀e_m, e_s ∈ E :
8  Implies(And(deploy_prev(sc, e_m) = 2, deploy_prev(sc, e_s) = 3,
9                isolated(e_m) = 1, isolated(e_s) = 0,
10               masterPresent(sc) = 1),
11         deploy(sc, e_s) = 2)
```

Listing 2. Constraints for valid post-isolation deployments

Lines 1-2 ensure that the mapping of ASWCs to the ASWC-Clusters does not change. Lines 4-5 ensure that no reallocation of an ASWC-Cluster is performed after a PAG-Transition. Lines 7-11 ensure that if a master and a hot-standby slave were present but the execution node of the master has been isolated, then the former hot-standby slave should become the new master. The other cases, like when only a master is required, are handled in a similar manner.

In order to decide about the deactivation order for the ASWC-Clusters, each instance of a cluster gets assigned a priority. Listing 3 exemplarily shows how the cluster-priorities can be calculated and how these are summed up to the vehicle priority-points $prioSumAllSCs(\mathbb{V})$ and $prioSumActiveSCs(\mathbb{V})$.

$$
\begin{aligned}
&1\ \forall sc \in SC:\\
&2\ prioPointsMaster(sc) = asil(sc) + failOp(sc) + 2\\
&3\\
&4\ prioPointsHotSlave(sc) = Ite(hotStandBySlaveReq(sc) = 1\,,\\
&5\qquad\qquad asil(sc) + failOp(sc) + 1\,,\ \ 0))\\
&6\\
&7\ prioSumAllSCs(\mathbb{V}) = \textstyle\sum_{sc\in SC}(prioPointsMaster(sc)\\
&8\qquad\qquad + \ prioPointsHotSlave(sc))\\
&9\\
&10\ prioSumActiveSCs(\mathbb{V}) = \textstyle\sum_{sc\in SC}\sum_{e\in E}(\\
&11\quad Ite(\ \ deploy(sc,e) = 2\,,\\
&12\qquad prioPointsMaster(sc),\\
&13\qquad Ite(deploy(sc,e) = 3,\ \ prioPointsHotSlave(sc),\ \ 0)))
\end{aligned}
$$

Listing 3. Calculation of the priority points of the single deployed instances

Finally, these priority-points can be used to decide which cluster instances have to be deactivated when the system resources become insufficient. Listing 4 depicts one simple algorithm to do this.

```
1  priorityReduction := 0
2  while True:
3    s.push()
4    s.add(prioSumAllSCs(𝕍) - priorityReduction
5        = prioSumActiveSCs(𝕍))
6    result := s.check()
7    s.pop()
8    if result = sat:     break
9    priorityReduction := priorityReduction + 1
10   if prioSumActiveSCs(𝕍) = 0: exit
```

Listing 4. Determine the set of deployable instances

Before executing this algorithm, all deployment constraints and the set of isolated execution nodes are defined. When a PAG-transition is calculated, some solution properties of the former deployment are set as fixed properties for the follow-up deployment, e.g., to avoid undesired changes in the deployment.

Line 3 pushes the already set constraints onto a stack. Line 4-5 add a new constraint defining the desired value of *prioSumActiveSCs* in the solution. Afterwards, the problem is checked and the additional constraint is removed again in line 7. If there exists a solution for the problem, line 8 evaluates to $True$ and the algorithm terminates successfully with a valid follow-up deployment. If no solution exists, *prioSumActiveSCs* is decreased until a valid solution is found. Decreasing the value of *prioSumActiveSCs* allows to deactivate those cluster instances whose priorities sum up to $prioSumAllSCs(\mathbb{V}) - prioSumActiveSCs(\mathbb{V})$. This mechanism is repeated as long as the property $prioSumActiveSCs(\mathbb{V})$ becomes zero. When this is the case, the algorithm exits unsuccessfully, meaning that no valid follow-up deployment exists (line 10). Instead of this linear search, also a more efficient binary-search or other algorithms could be applied, but this was not in focus of our work. We implemented the system model, constraints and algorithms using the Z3 SMT-Solver [8].

4 Evaluation and Example

In this section we show the applicability of our approach on a simplified example from the automotive domain. Consider the following 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 : RemainingRangeCalc	A	0	0.7
	s_3 : EnergyEfficiencyAssist	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 : ManualAcceleration	D	3	1
	s_7 : ManuelBraking	D	3	1
	s_8 : ManualSteering	D	3	0.5

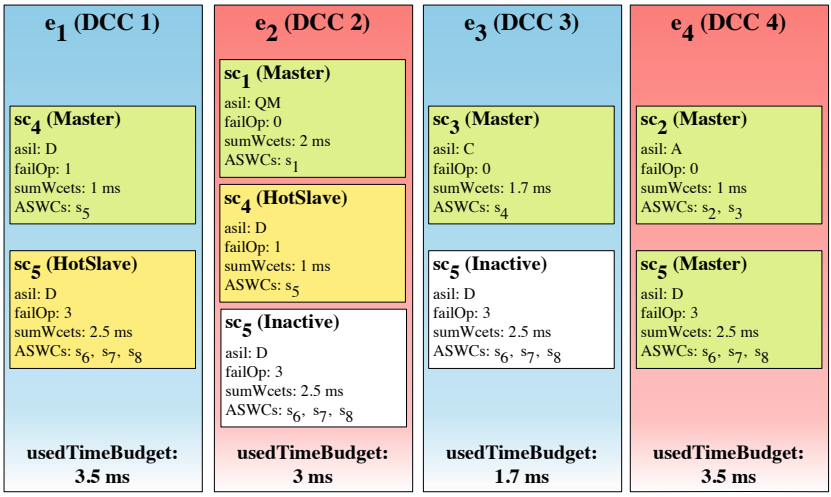
The features f_3 and f_4 are placeholders for some Advanced Driver Assistance Systems (ADAS), like an ACC or automatic parking. Let f_4 be required to stay active after a failure, but f_3 is not required to be active after a failure. As ASWC s_5 contributes to realize both f_3 and f_4, it has $failOp(s_5) = 1$. As ASWC s_4 only realizes f_3, it is sufficient that $failOp(s_4) = 0$.

In this example, five ASWC-Clusters $\{sc_1, ..., sc_5\}$ are established. The 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.

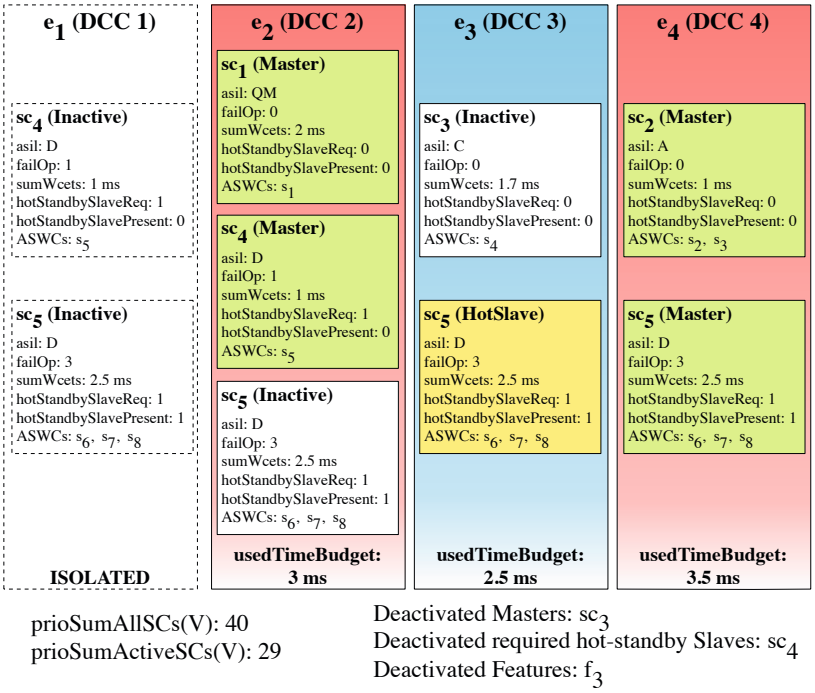
Considering a CPC with four execution nodes (DCCs) as shown in Fig. 2(a), a valid initial deployment for the example is shown in Fig. 4(a). Fig. 4(b) shows the follow-up deployment for the case that DCC 1 has been isolated. The colors (red/blue) of the execution nodes denote their attached power-supply.

We assume here that $minFTT(s_i) \leq frt(\mathbb{V})$ for the fail-operational ASWCs. Hence, hot-standby slaves are required. As provided execution time of the execution nodes per cycle, we assume $totalTimeBudget(e_i) = 4ms$. It can be seen in both Fig. 4(a) and Fig. 4(b) that $\forall e_i \in E : usedTimeBudget(e_i) \leq totalTimeBudget(e_i)$.

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 as it is not required that sc_4 is still present after the next isolation. Furthermore, the slave of cluster sc_5 gets 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 and not on e_2, because master and slave must not depend on the same power-supply. However, to be able to execute cluster sc_5 on execution node e_3, cluster 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 $\alpha(sc_3) = \{s_4\} \subseteq \chi(f_3)$.



(a) Initial deployment for the example



(b) Followup deployment after DCC1 has been isolated

Fig. 4. Example about an initial and a followup deployment

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, because $prioPointsMaster(sc_3) = 5$ and $prioPointsHotSlave(sc_4) = 6$. Hence, when DCC1 is isolated, only 29 priority points can be provided by the system (cf. Fig. 4(b)). 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 $failOp(sc_5) = 3$.

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

5 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], 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.

6 Conclusion and Future Work

In this paper, we introduced a formal model of mixed-critical systems including the relationship of functional features and software components realizing the functional features. A set of formal arithmetic constraints describe valid deployments of the software components to a fault-tolerant HW/SW platform for vehicles. Based on the model and the constraints, an approach to calculate and analyze valid deployments of mixed-critical components was provided.

The analysis focuses on 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 implemented the model as input for an SMT-Solver, which calculates the deployment solutions. We analyzed which components and features have to become inactive after certain failures. An evaluation was shown by an automotive example.

As future work, we are going to include communication channels between components into the model. Also, 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. Finally, we want to evaluate the scalability of our approach based on the layout of a concept car that we construct.

Acknowledgments. 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. Avizienis, A., Laprie, J., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. on Dependable and Secure Computing (1), 11–33 (2004)
2. Blanke, M., Staroswiecki, M., Wu, N.E.: Concepts and methods in fault-tolerant control. In: Proceedings of the American Control Conf., vol. 4. IEEE (2001)
3. Sommer, S., Camek, A., Becker, K., Buckl, C., Knoll, A., Zirkler, A., Fiege, L., Armbruster, M., Spiegelberg, G.: Race: A centralized platform computer based architecture for automotive applications. In: IEEE Vehicular Electronics Conference / Int. Electric Vehicle Conference (VEC-IEVC) (2013)
4. Armbruster, M., Fiege, L., Freitag, G., Schmid, T., Spiegelberg, G., Zirkler, A.: Ethernet-Based and Function-Independent Vehicle Control-Platform: Motivation, Idea and Technical Concept Fulfilling Quantitative Safety-Requirements from ISO 26262. In: Adv. Microsystems for Automotive Applications (AMAA), pp. 91–107 (2012)
5. Henzinger, T.A., Horowitz, B., Kirsch, C.M.: Giotto: A time-triggered language for embedded programming. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 166–184. Springer, Heidelberg (2001)
6. International Organization for Standardization: ISO/DIS 26262-1 - Road vehicles - Functional safety, Part 1 Glossary. Technical report, ISO/TC 22 (2011)
7. Becker, K., Armbruster, M., Schätz, B., Buckl, C.: Deployment Calculation and Analysis for a Fail-Operational Automotive Platform. In: 1st Workshop on Engineering Dependable Systems of Systems (EDSoS) (2014)
8. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
9. Shelton, C., Koopman, P., Nace, W.: 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), pp. 156–163. IEEE (2003)
10. Boone, B., De Turck, F., Dhoedt, B.: Automated deployment of distributed software components with fault tolerance guarantees. In: 6th Int. Conf. on Software Engineering Research, Management and Applications (SERA), pp. 21–27. IEEE (2008)