Generic Management of Availability in Fail-Operational Automotive Systems

Philipp Schleiss[1], Christian Drabek[1], Gereon Weiss[1], and Bernhard Bauer[2]

[1] Fraunhofer ESK, Munich, Germany, name.surname@esk.fraunhofer.de
[2] Department of Computer Science, University of Augsburg, Germany

Abstract. The availability of functionality is a crucial aspect of mission- and safety-critical systems. This is for instance demonstrated by the pursuit to automate road transportation. Here, the driver is not obligated to be part of the control loop, thereby requiring the underlying system to remain operational even after a critical component failure. Advances in the field of mixed-criticality research have allowed to address this topic of fail-operational system behaviour more efficiently. For instance, general purpose computing platforms may relinquish the need for dedicated backup units, as their purpose can be redefined at runtime. Based on this, a deterministic and resource-efficient reconfiguration mechanism is developed, in order to address safety concerns with respect to availability in a generic manner. To find a configuration for this mechanism that can ensure all availability-related safety properties, a design-time method to automatically generate schedules for different modes of operations from declaratively defined requirements is established. To cope with the inherent computational complexity, heuristics are developed to effectively narrow the problem space. Subsequently, this method's applicability and scalability are respectively evaluated qualitatively within an automotive case study and quantitatively by means of a tool performance analysis.

1 Introduction

Within the automotive domain, the demand for highly available systems is increasing through the vision of automated driving. As a driver is not required to constantly be part of the control loop in an automated vehicle, the underlying control system must be capable of compensating for all safety-relevant failures. With respect to availability, a system can either account for failure-induced reduction of computational capacity by means of over-provisioning, for instance with dedicated hardware and triple redundant architectures, or alternatively resort to a form of *graceful degradation* [8]. Within the transportation domain the need for *fail-operational behaviour* has traditionally been solved through dedicated redundancy in form of federated architectures [1], as most prominently seen in the triple-triple redundant architecture of modern *Fly-by-Wire* systems [16]. In contrast, the latter option of graceful degradation is of special interest in the cost-sensitive automotive industry in order to limit the required amount of spare computing resources. Here, automated vehicles may only require a limited set of functionality for a short period of time until the vehicle can be safely halted.

With respect to availability, general purpose computing platforms pose as a promising solution to limit the amount of required hardware, as there purpose can be redefined at runtime. The foundation for such reconfiguration schemes was laid in mixed-criticality research by allowing a platform to host multiple independent functions, as for instance demonstrated by the *Integrated Modular Avionics (IMA)* in the aviation domain [15]. Despite this, reconfiguration generally competes with the principles of safety-critical systems, which require an operation free of unpredictable interference. This is for instance seen in the functional safety standard ISO 26262 [7] of the automotive domain, which is restrictive with respect to reconfiguration. Therefore, the question arises how availability can be ensured through reconfiguration schemes in order to benefit from the resource saving potential of integrated architectures and graceful degradation while at least maintaining the current level of safety.

Consequently, this work develops a generic *monitoring and reconfiguration service (MRS)* to ensure the availability of multiple independent functions during runtime based on the notion of a *Safety-Element-out-of-Context*. As guaranteeing deterministic behaviour is imperative within the safety domain, this service is designed in a static manner, thereby only utilising *mitigation plans* that were previously verified. From a design perspective, the need to manage failure modes further increases the effort of developing already complex automotive systems. As such, this work further focuses on a method to define reconfiguration behaviour declaratively and automatically calculate configurations for all managed modes of operation. For this, a system model is developed, which is then enriched with scheduling information through the use of a novel set of heuristics and mixed-integer linear programme (MILP) techniques. This additional scheduling information poses as an extension of the system's interface description to guarantee the required real-time behaviour when implemented correctly on each control unit, thus providing the basis for a compositional system integration.

In detail, Section 2 introduces this monitoring and reconfiguration service, followed by a method for synthesising schedules of fail-operational systems in Section 3. Section 4 analyses applicability in an automotive case study and evaluates the performance quantitatively before concluding in Section 5.

2 Fail-Operational Safety Mechanism

2.1 Monitoring & Reconfiguration Service (MRS)

In the following, a *monitoring and reconfiguration service (MRS)* is developed in order to provide a safety mechanism that can generically ensure the availability of multiple independent functions within a set of distributed control units during runtime. For this, a synchronously operating MRS instance is deployed onto each control unit participating in the management of failure modes. The period of the software-based MRS is in turn based on the failover times of the managed functionality, which describe the maximal amount of time a functionality can remain unavailable. These figures are typically determined during a *Hazard and*

Risk Analysis. Further, the MRS itself consists of a reporting and an evaluating task that are respectively responsible for informing all other control units about the current state of the hosted applications in form of a heart-beat, and evaluating all received heart-beats to trigger a reconfiguration. Based on this heart-beat information, each control unit can determine the status of the complete system. More precisely, the state of all applications in the system is concatenated to one lookup key for use in a reconfiguration database, which is deployed on each control unit. Based on this key, a lookup occurs that either results in the control unit remaining in the current state or alternatively performing a reconfiguration based on the predetermined *mitigation plan*. A mitigation plan includes a new schedule for all application instances hosted on the respective control unit. Through this decentralised architecture it is possible to conduct reconfiguration involving multiple control units without the need for a central coordinating unit.

2.2 Hardware Architecture

As each control unit expects to receive a heart-beat from each other control unit within each period, the missing of a heart-heat is interpreted as the failure of a control unit. From a hardware-perspective, additional guarantees with respect to reliable communication links between control units are however necessary in order to deduct the failure of a control unit from a missing heart-beat. Moreover, control units utilising this reconfiguration scheme must be equipped with strong diagnostic capabilities to perform fail-silent behaviour in case of unrecoverable local faults, thereby ensuring the fail-operational properties of the entire system (see Fig. 1). To provide fail-operational behaviour between control units in a cost-efficient manner, a *1-out-of-2 safety architecture with diagnostics (1oo2D)* was deployed on basis of previous research [11]. Here, each unit is equipped with strong diagnostic capabilities in form of lock-stepping mechanisms and additional monitoring elements, such as hardware watchdogs. In addition, the use of diverse hardware platforms and bus systems further limits the potential occurrence of faults originating from a common cause.

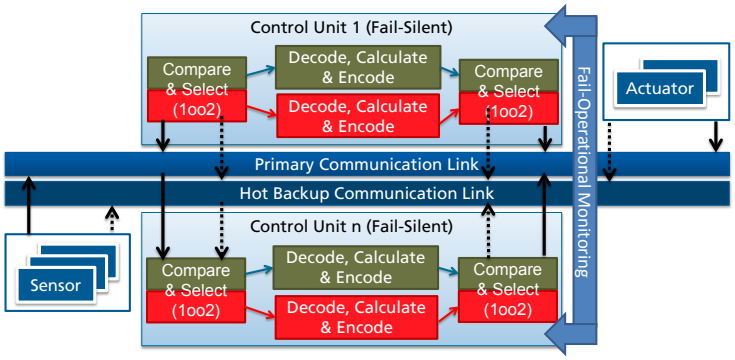


Fig. 1: 1oo2D Architecture

3 Method for Reconfiguration Planning

3.1 System Model

In order to configure the previously described monitoring and reconfiguration service, an automated design process is used to generate mitigation plans for all managed failure modes. Hereby special focus is laid on the declarative nature of the approach. This allows system designers to only specify availability requirements without the need to manually determine a schedule for each failure mode and without needing to ensure that all failover times are met. For this, a minimal system model is derived under consideration of distributed topologies, operating modes, hierarchical resources access patterns, and preemption. This model is later formalised and used as part of a reconfiguration planning approach based on the advances in constraint-based system synthesis [3].

Jobs, Tasks & Compositions. In this work, a *task* is defined as the use of exactly one resource during a specific time interval, which is in turn bounded by a task- and resource-specific *worst case execution time* (WCET), as seen in the software architecture of a simplified automated driving use case depicted in Fig. 2. Tasks in turn are logically grouped to *task composition*, to allow for an abstraction of the system (e.g. *Steer-by-Wire*). As such, multiple instantiation of a specific type of composition or task are required to provide functionality redundantly (e.g. *Wheel Tick* tasks), which are each responsible for one specific wheel of a vehicle. Moreover, a task consists of a sequence of *jobs*, each representing a specific invocation of a task. To ensure the periodic execution of a task's jobs, each task can be annotated with a period (cf. *Steering* or *Highway Pilot*). In additional, other real-time constraints, such as the maximal age of processed data (cf. *Trajectory Planning* or *Steering Engine 1*) or synchronous executions between jobs of two tasks (cf. *Steering Engine 1 & 2*) can be defined.

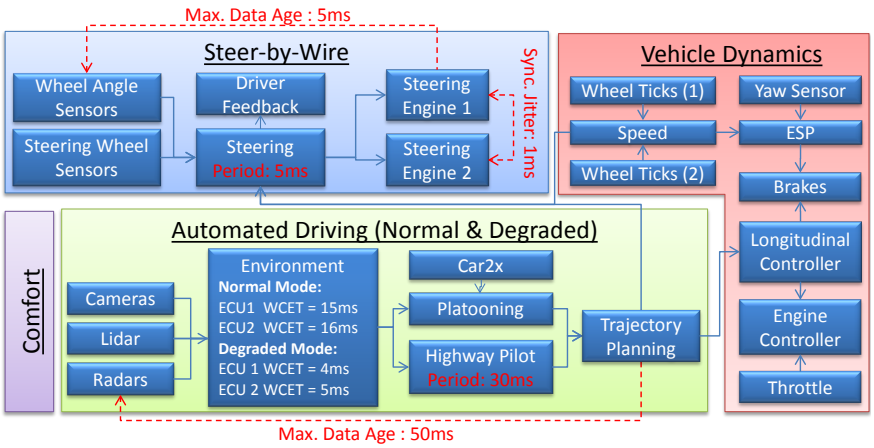


Fig. 2: Example Software Architecture

Resources. A *resource* can represent a control unit or bus, but also any other type of resource such as a hardware controller. For this, the concept of hierarchical dependencies between resources is introduced. For instance, a control unit can pose as a top-level resource without dependencies to other resources, whereas a memory region of this control unit is seen as a subordinate resource belonging to the superordinate top-level control unit resource. Based on this, hierarchical dependency between tasks are used to, for instance, describe that a subordinate task can only access the memory region of that specific control unit if its superordinate task is also assigned to the same control unit. Moreover, interleaved access patterns are possible, allowing multiple tasks to cooperatively share a resource through preemption. Despite this, some activities, such as accessing a critical memory structure, must remain atomic to prevent data corruption and indeterministic behaviour. In such cases, a resource is deemed non-preemptable.

Graceful Degradation & Modes. As it is often not desirable to completely eliminate functionality in overload conditions [4], a more fine-grained consideration of required functionality in each system mode is needed. Despite this, current mixed-criticality research often only applies a rigid model of HI and LO criticality levels. Further, computational peaks and variable workloads are often simply classified into hard and soft real-time requirements, thus squandering the potential for resource-efficient designs. Consequently, a generalised taxonomy was developed in [10] to classify resource access in the dimensions of quantity of resources and frequency of occurrence. To incorporate these previous research results, the developed system model captures such fine-grained information on resource requirements by allowing compositions to be assigned to *system modes* and sorted by their importance (e.g. the *Comfort* is less important than *Steer-by-Wire* in Fig. 3). These system modes are defined for the entire (sub-)system, including hardware component failures or environmental changes. In contrast, functionality can be modelled as multiple distinct modes of a composition that are only allowed to be admitted under mutual exclusion in order to address functionality that can degrade internally (e.g. Normal & Degraded Mode for *Automated Driving* in Fig. 3) or exhibit variable workloads. Moreover, tasks can be annotated with different *scheduling modes* to account for application-specific data consistency requirements of standby tasks. For instance, a *cold-standby* task will only be assigned to a resource and thus only attain an internal state after being scheduled, whereas a *hot-standby* task will exhibit similar scheduling demands as an active task as only the task's external effects will be suppressed.

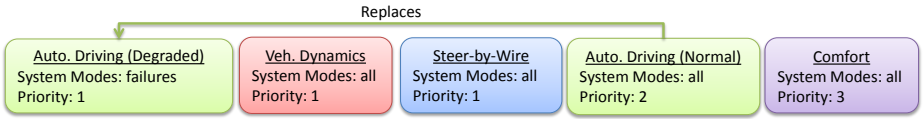


Fig. 3: Graceful Degradation Example

3.2 Mixed-Integer Linear Programme

Motivated by the fact that mixed-integer linear programmes have been successfully utilised for real-time system synthesis [13], a MILP representation of the previously introduced system model was developed. This MILP formulation can either be used to maximise the amount of active compositions or alternatively find valid schedules for a predetermined set of required compositions for each system mode. Due to the extensive nature of the derived MILP (22 constraints in total), this work focuses on the most distinctive and less straight-forward equations. All variables and constants are summarised in Table 1.

Target Function. In favour of limiting the wasteful deployment of processing resources, it is desirable to achieve a high level of resource utilisation. For this, a global view of the system is inevitable to always find an optimal configuration. As such, this problem is formulated with the intention of maximising the amount of compositions that can be successfully scheduled on a given set of resources (1). The admission of a composition $c \in C$ to the system's configuration is encoded in a binary variable u_c by setting $u_c = 1$.

$$\max \sum_{c}^{C} u_c \tag{1}$$

Graceful Degradation. To account for the possibility of insufficient resources, compositions may be assigned priorities to define a hierarchical ordering of their importance. Therefore, a composition may only be admitted if all compositions of a higher priority C_c^+ are also included in the schedule. All compositions of lower priority are defined as C_c^-. To enforce these restrictions, u_c shall only take a true value when all $u_{c'}$ for $c' \in C_c^+$ are also true. Therefore, u_c is to be multiplied with the cardinality of C_c^+ to prevent an unwanted admission of composition c (cf. Fig. 3):

$$\bigvee_{c}^{C} u_c |C_c^+| \leq \sum_{c'}^{C_c^+} u_{c'} \tag{2}$$

Tasks to Resources Mapping. To represent the smallest schedulable entities, each composition is broken down into tasks $t \in T$ which must be mapped to a single resource r out of a task-dependent set of resources R_t. The binary variable $a_{tr} = 1$ indicates that a task is permanently assigned to a certain resource. During implementation a substantial performance penalty was noticed when formulating this constraint in form of an inequation ($<= 1$). Therefore, an additional virtual resource, on which jobs can be executed with infinite speed, is introduced, thus extending the set R_t with the virtual resources to R_t^+. This allows every task to always be assigned to exactly one resource (3).

$$\bigvee_{c}^{C} \bigvee_{t}^{T_c} \sum_{r}^{R_t^+} a_{tr} = 1 \tag{3}$$

Mutual Exclusion. To account for the degradation within a functionality, it must be ensured that the same functionality is not simultaneously provided by multiple compositions. Therefore, the composition providing normal functionality can disable tasks of its degraded compositions (e.g. Normal Mode is preferable over Degraded Mode for *Automated Driving* in Fig. 3). All tasks in c that are disabled by a composition $c' \in C_c^-$ are contained in the set $T^D_{cc'}$. The equation 4 allows a number of tasks in the composition c to be disabled, if c' is enabled. However, it does not specify which tasks. As only tasks replaced by tasks from other compositions may be disabled, the a_{tr} variable of each disabled task must be forced to zero (5). Moreover, to ensure that either all or none of the tasks within a composition are admitted, the sum of all respective binary variables a_{tr} must either be equal to the amount $|g|$ of tasks within that composition or zero.

$$\bigvee_{c}^{C} \sum_{t}^{T_c} \sum_{r}^{R_t} a_{tr} = |T_c| u_c - \sum_{c'}^{C_c^-} |T^D_{cc'}| u_{c'} \tag{4}$$

$$\bigvee_{c}^{C} \bigvee_{c'}^{C_c^-} \sum_{t}^{T^D_{cc'}} \sum_{r}^{R_t} a_{tr} = |T^D_{cc'}| u_c - |T^D_{cc'}| u_{c'} \tag{5}$$

Unified Timeline. During the implementation of this work, an up to 10-fold performance benefit was attained by conceptually partitioning all resources along one sequential timeline Z (see Fig. 4). For this, the concept of a *hyper-period* is utilised. It is defined as the least common multiple of all occurring periods, thus describing the shortest time frame after which a schedule may be repeated in a symmetric manner. Based on this, the minimal length of the unified timeline Z could be determined by multiplying the system's hyper-period H with the number of resources and adding the largest deadline. This addition of $max(Deadline)$ is required to account for jobs that start within the last period of a hyper-period and potential only end after the start of the next hyper-period.

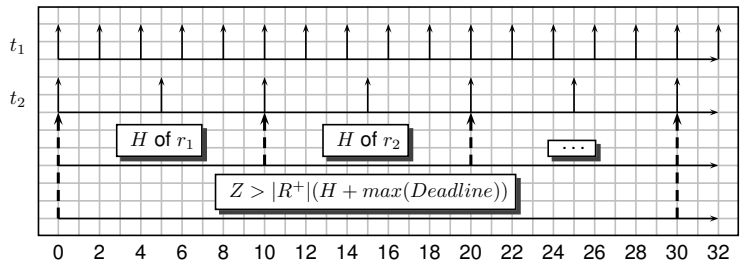


Fig. 4: Hyper-Periods in a Unified Timeline

Order of Execution & Preemption. To determine if a job j is activated after a job j' has ended, the activation time of j is represented by a real variable s_j, whereas the time of completion of j' is defined through the real variable e'_j. For each period of a task t, the start-time of a respective job $j \in J_t$ is bounded by a constant start-offset P^S_j and end-offset P^E_j. With respect to the unified timeline, the resource-specific offset must be subtracted in order to attain the actual time value (6). The jitter of start- and end-times between a task's periods can also

be bounded through this definition. In addition, the order of execution between tasks is encoded in the binary variables $b_{jj'}$ (see Fig. 5). To force the variable $b_{jj'}$ to only be zero if e'_j lies before s_j, $b_{jj'}$ is multiplied with a constant that is larger than any other variable. As such, only a positive difference of $s_j - e_{j'}$ leads to $b_{jj'} = 0$ (7).

$$\bigvee_{t}^{T} \bigvee_{j}^{J'_t} P_j^S \leq s_j - \sum_{r}^{R_t} Z_r a_{tr} \leq P_j^E \quad (6) \qquad \bigvee_{t}^{T} \bigvee_{j}^{J_t} \bigvee_{t'}^{T \setminus t} \bigvee_{j'}^{J_{t'}} 0 \leq s_j - e_{j'} + b_{jj'} Z \leq Z \quad (7)$$

An overlap free schedule is defined as the start of an execution occurring after another job's execution ends. In this case, exactly one variable in the set $(b_{jj'}, b_{j'j})$ must be true, as all other cases lead to logical contradictions or execution overlaps (see Fig. 5). If an overlapping resource access is allowed for a specific resource, one job can preempt another job. Through setting $b_{jj'} + b_{j'j} = 1$ such inter-job preemption can be selectively prohibited, for instance, to prevent negatively influencing WCETs through cache invalidation effects.

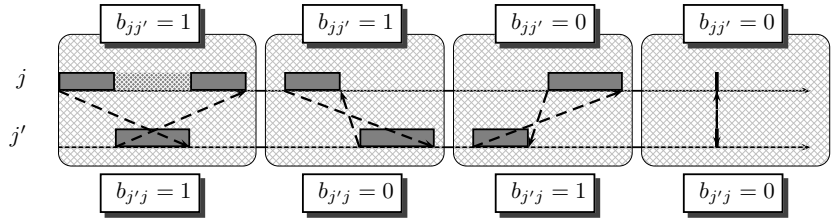


Fig. 5: Execution Overlaps

Execution Times. When assigning a task t exclusive control over a resource, the largest possible time that can elapse between its activation s_j and completion e_j is defined as a task's resource-specific WCET W_{tr}. In addition, tasks operating on shared resources must also account for the preemption-incurred prolongation of their jobs' execution times by considering the overhead of every context switch. The total time delay of a job j by any potentially preempting job $j' \in J_j^P$ and the therewith connected temporal overhead is represented by a variable $i_{jj'}$. In all, the maximal distance between s_j and e_j is defined through the task's resource-dependent WCET and the sum of all interferences (8). As a job's execution may only lie entirely before, within, or after another job's execution window, interruption $i_{jj'}$ caused by each job j' can be added together to obtain the total time of preemption for job j, as depicted in Fig. 6.

$$\bigvee_{t}^{T} \bigvee_{j}^{J_t} e_j - s_j = \sum_{r}^{R_t} a_{tr} W_{tr} + \sum_{j'}^{J_j^P} i_{jj'} \quad (8)$$

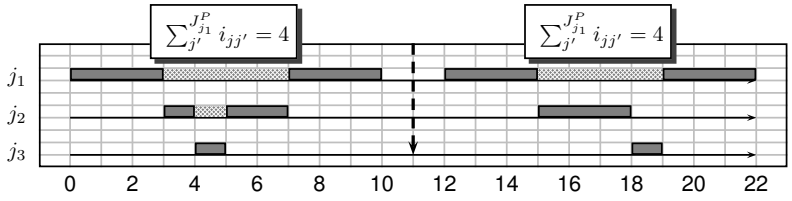


Fig. 6: Cascading Preemption

The non-negative preemption time $i_{jj'}$ of a task's job j by another job $j' \in J_{t'}$ is defined by the resource-dependent WCET $W_{t'r}$ of the preempting job and is further influenced by effects related to the used resource, such as activation jitters, or storing and loading of registers, which are all accounted for with a worst-case constant overhead O_r. As O_r is only resource-dependant, it must be chosen large enough to account for the worst performing task allocated on r. In cases where this constant overhead is however too pessimistic, as for instance seen in caching strategies, a set of finer constraints can be utilised instead of the simplified model of cascading preemptions (see Fig. 6). Moreover, preemption may be completely prohibited for a task or only permitted at predefined points. When formulating this constraint, a one sided bound on $i_{jj'}$ is sufficient, as the target function indirectly minimises $i_{jj'}$. In consequence, every overlap $i_{jj'}$ will diverge to its minimum in case of resource restrictions. Through this, the introduction of another binary variable is circumvented. Equation 9 provides a lower bound for the preemption time $i_{jj'}$:

$$\bigvee_{t}^{T} \bigvee_{j}^{J_t} \bigvee_{j'}^{J_j^P \subset J_{t'}} \bigvee_{r}^{R_t} a_{t'r} W_{t'r} + a_{tr} O_r + b_{j'j} Z + b_{jj'} Z - 2Z \leq i_{jj'} \tag{9}$$

Table 1: Table of Notations

Notation	Type	Description
a_{tr}	var $\{0,1\}$	Assignment of task t to resource r ($a_{tr} = 1$)
$b_{jj'}$	var $\{0,1\}$	Start of job j is before end of j' ($b_{jj'} = 1$)
C	const set	All compositions
C_c^+ / C_c^-	const set	Composition with higher/lower importance than composition c
e_j	var $\mathbb{R}$	End of job j
H	const $\mathbb{R}$	Hyper-period (smallest common multiple of all periods)
$i_{jj'}$	var $\mathbb{R}$	Time that j is preempted by j'
J_t	const set	Jobs of task t
J_j^P	const set	Jobs that potentially preempt job j
O_r	const $\mathbb{R}$	Task-independent preemption overhead of resource r
P_j^S & P_j^E	const $\mathbb{R}$	Begin and end of job j's period
R_t	const set	Potential regular resources of task t
R_t^+	const set	As R_t with additional virtual resource
s_j	var $\mathbb{R}$	Start of job j
T	const set	All tasks
$T_{cc'}^D$	const set	Tasks in c that can be replaced through tasks from c'
u_c	var $\{0,1\}$	Composition c is admitted to system schedule ($u_c = 1$)
W_{tr}	const $\mathbb{R}$	Raw WCET of task t on resource r
Z_r	const $\mathbb{R}$	Offset of resource r on unified timeline Z

3.3 Heuristics

Through the NP-hard nature of this optimisation problem, scalability becomes an important concern. As general MILP solving techniques cannot fully exploit problem-specific characteristics, heuristics pose as a promising solution for extending the scope of systems that can be successfully synthesised [6]. The following paragraphs describe three strategies (S1-S3) which aim at limiting the problem space while trying to only minimally impair the solution space.

S1: Resource Assignment. To decrease the problem's complexity, tasks may be pre-assigned to a certain resource in cases where multiple potential resource allocations exist. For this, tasks are ordered by the ratio between their WCET and potential window of resource access (*WCET-window ratio*). Thereby, tasks with small WCETs and large access windows are first assigned to resources. The assignment process aims at balancing tasks fairly by selecting the resource with the least utilisation. This process repeats until either the *WCET-window ratio* or the resource utilisation grow above their respective threshold.

S2: Grouping of Tasks. Moreover, multiple tasks may be combined into a single task whenever their periods are identical. This strategy requires all tasks grouped within the newly created task to be assigned to the same resource.

S3: Sequential Resource Access Windows. To eliminate further binary MILP variables, potentially conflicting executions can be resolved through sequentialisation. Here, each job's earliest start and latest end time is determined by analysing the system's data flow graph. Thereafter, the heuristic sequentialises conflicts of jobs with similar WCETs, as these jobs are unlikely to benefit from mutual preemption. In contrast, the enforcement of a sequential execution pattern on jobs with strongly varying WCETs would remove opportunities for preemption. The pairs of tasks are then sorted based on how much each job's execution window must be narrowed to allow a sequential execution (cf. Fig. 7). This sum is then weighted by the potential changes in ratios between WCETs and execution windows, in order to find conflicts that barely overlap or alternatively exhibit large WCET-window ratios. The respective earliest start and latest end times of the best job pair are then narrowed by balancing the remaining ratio between WCETs and potential execution windows. The process is repeated until no pair can be sequentialised without falling below a threshold based on the WCET-window ratio.

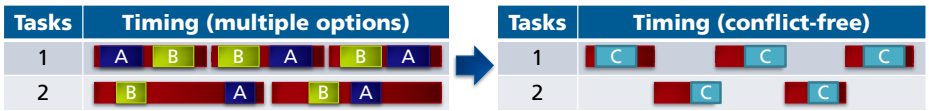


Fig. 7: S3 heuristic for sequential resource access (red: resource access windows)

4 Evaluation

To determine the value of this method, its applicability in the automotive domain and scalability are of interest, as these factors are crucial for ensuring a successful transfer of this prototypical concept into future systems throughout the industry. As such, the applicability will be evaluated qualitatively by performing an assessment based on criteria that was identified as relevant during the implementation of a full-scale e-vehicle with a fail-operational *Steer-by-Wire* system. With respect to scalability, the solving performance of the MILP- and heuristic-based system synthesis approach is analysed quantitatively for different synthetic workloads to determine its practical limits with respect to realistic future system sizes.

4.1 Automotive Case Study

Within the SafeAdapt [12] research project an electric prototype vehicle was developed that aims at integrating multiple critical functionalities, such as Steer- and Brake-by-Wire compositions, onto a shared control infrastructure based on a 1oo2D safety architecture (cf. Sec. 2.2). Here two diverse hardware platforms were utilised on basis of an AURIX safety controller and two ARM MCUs with software-based lock-stepping mechanism, which are interconnected by a time-trigger redundant ethernet backbone.

To illustrate the granularity of a typical data flow, the *Steer-by-Wire (SbW)* system is described. With this functionality, two independent timing chains exist to respectively adjust the angle of the front wheels with a redundant pair of steering rack engines in accordance to the steering wheel angle and further provide information to the driver with respect to the road surface conditions in form of vibrations. Focusing on the steering engine control loop, both engines must be readjusted periodically, actuate synchronously within a certain time margin, and only apply control signals originating from recently sampled sensor data. In addition, each of the steering wheel and steering angle position sensors is individually modelled as three tasks: one for polling the sensor, one for placing it in a designated buffer of a network chip, and one for the network controller's transmission of the queued data (cf. Fig. 8).

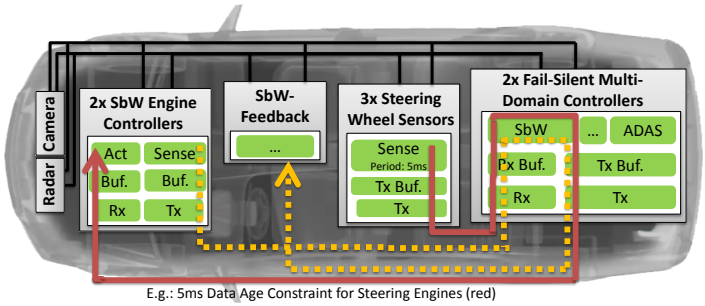


Fig. 8: Steer-by-Wire Architecture

4.2 Assessment of Applicability

To determine the applicability of the proposed method within the automotive domain, a non-exhaustive set of criteria was identified as relevant through experience gained from the previously described e-vehicle project. Based on these criteria, the method's applicability is evaluated in the following:

Tool-interoperability The generic and minimal nature of the developed system model promotes the binding of different domain specific modelling concepts. As an example, the vehicle's hardware architecture, data flow, modes of operation, availability requirements, and timing characteristics were modelled in the ARXML exchange format of the predominant AUTOSAR standard [2] and subsequently transformed into the domain-independent system model (cf. Sec. 3). The results of the configuration planning process were then again automatically added to the ARXML format, thus providing a seamless integration into existing tool chains.

Stability Extensions, changes, and fixes for individual applications are common within the life-cycle of a vehicle. To not adversely affect other functionalities, it is common practice to maintain a stable schedule for all bus systems. With respect to the developed synthesis method, such stability can be easily achieved by statically pre-defining the schedule for all applications that must remain stable. Hence, only a new schedule for the remaining applications must be found, thus guaranteeing future interoperability and the ability to only partially update an already existing system.

Standardisation As automotive system architectures are often based on components from multiple independent vendors, it is crucial to standardise any service that requires interoperability across control units. As such, the MRS (cf. Sec. 2.1) was integrated as a basic AUTOSAR service with a uniform communication protocol as part of an already standardised software architecture in order to showcase the feasibly of a future standardisation.

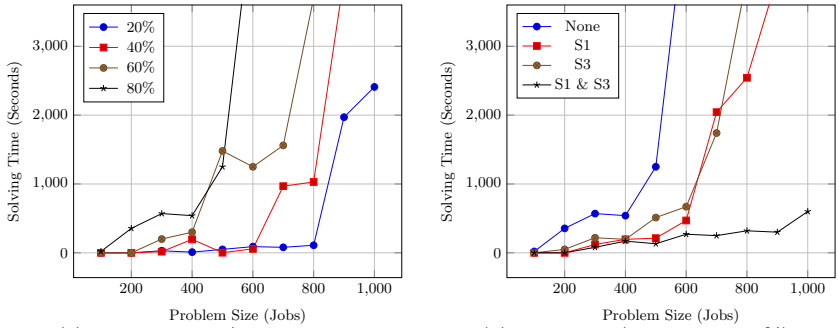
Reusability Through the design of the runtime reconfiguration mechanism based on the concept of a *Safety-Element-out-of-Context*, the required reconfiguration logic had to only be implemented once instead of individually for each functionality, thus substantially reducing the development, verification, and validation effort.

Runtime overhead With respect to runtime overhead, the generic runtime mechanism showed WCETs of less than 100μs and was executed in 5ms periods in order to always meet the strictest failover times of 10ms. Moreover, the excepted overhead remains close to constant with an increasing amount of managed functionalities. This is attributed to the fact that the execution and usage of bus systems only occurs once per period for all functionalities as compared to each functionality utilising an individual monitoring mechanism and communication slot. In addition, mechanisms for ensuring the data consistency of standby tasks more dominantly influences resource usage. Here different concepts reaching from *cold-standby* over *warm-standby* (e.g. cyclic data updates) to *hot-standby* implementation exist, depending on the task-specific requirements of the individual compositions (cf. Sec. 3.1).

4.3 Scalability & Tool Performance

Setup. To experimentally evaluate how this approach scales, the total amount of jobs, the number of interconnections, the degree of potentially overlapping executions, and the system's total utilisation were identified as factors that are likely to influence the solving time. During non-exhaustive tests, the system's utilisation was identified as the most interesting factor, as an increased utilisation already lead to monotonously growing and strongly diverging solving times at around 500 jobs. In contrast, the degree of potentially overlapping executions and the amount of interconnections showed less divergent performance differences. As such, the further evaluation focuses on determining the performance effects of different levels of utilisation with an without the use of heuristics while increasing the amount of jobs. For this, the other influencing factors are kept at a constant ratio in relation to the amount of jobs in order to enable an isolated evaluation of the utilisation parameter. The measurements are performed with the Gurobi optimisation software (version 6.0.3) [5] on an Intel Xeon E5-2660 CPU (2.2 Ghz) with 8 cores by synthetically scaling the previous automotive example.

Performance Evaluation. Based on these synthesised workloads, 10 test sets were extracted with four predefined average resource utilisations (20%, 40%, 60%, 80%). As seen in the experimental results in Fig. 9a, performance generally deteriorates with increased utilisation. Further, the strategies S1 to assign tasks to a fixed resource, S3 to sequentialise potentially conflicting resource utilisations, and the combination of both strategies were applied to the 80% utilisation test set, representing a typically used utilisation limit within the automotive industry. The strategy S2 for grouping tasks was not analysed in isolation, as its performance is directly dependent on the amount of tasks with identical periods. Moreover, non-exhaustive tests were conducted to experimentally determine an acceptable parametrisation of the heuristics. As seen in Fig. 9b, the point at which the performance deteriorates could be substantially postponed through the use of heuristics, allowing a system with 80% utilisation and 1000 jobs to be synthesised in around ten minutes. Most notably, the combination of resource assignment and sequential resource access heuristics exhibits the largest benefit, thus proving a promising opportunity to design more complex systems.



(a) Utilisations w/o Heuristics (b) Heuristics (Utilisation 80%)

Fig. 9: Performance Evaluation

4.4 Discussion

Based on the experience collected in the automotive case study, the proposed combination of a safety mechanism for generically ensuring availability and a system synthesis process poses as a viable option for designing automated vehicles in a more cost- and resource-efficient manner. Regardless, special rigour must be applied during the design of a generic availability management component, as an incorrect implementation could adversely affect multiple independent functionalities. This concern is however mitigated by the simplicity of the runtime mechanism, which is implemented on basis of a formally provable state machine. In addition, all calculated mitigation plans can be verified through an simple process by comparing the time-driven schedules of each operating mode against the formalised availability and timing requirements. This already ensures the correctness of the mitigation plans and further fosters manual quality improvements, such as jitter optimisations, by allowing an automated verification of any modification. Moreover, typically occurring timing issues during system integration can be mitigated through the early enrichment of the system's interface description with detailed timing requirements. This enables a compositional integration in which the correct timing behaviour is ensured during system integration even though each control unit was designed individually. Hereby, each control unit must however adhere to its predetermined temporal interface description. In addition, this reduced development effort can be utilised to create larger variant diversity and customised products that would otherwise be deemed infeasible.

In light of the solving performance, it seems reasonable to presume that automated schedule synthesis is a feasible method for substantially reducing development effort. In addition, the long development-cycles of safety-critical systems can even make solving times of multiple days an acceptable option. Regardless, the future use of project specific knowledge as well as the creation of more sophisticated heuristics is likely to substantially increase the method's performance and allow the synthesis of systems with higher complexity. Similarly, an alternative implementation based on saturated module theory concepts, which have proven to be useful for similar problems [14] and even outperform MILP approaches [9], could further increase performance.

5 Conclusion

In pursuit of designing fail-operational systems in a cost-efficient manner, this work exploits a monitoring and reconfiguration service that is utilised as a generic safety mechanism for ensuring availability of independent functionalities. Based on this, an accompanying synthesis process for automatically generating mitigation plans in form of schedules for all anticipated operational modes of a system was researched and implemented. The method's applicability was then demonstrated successfully on basis of experiences gained during the development of a real e-vehicle. Moreover, the performance of the synthesis was significantly increased by applying heuristic strategies, thus ensuring its applicability with respect to the more complex systems of future vehicles.

Acknowledgement. The research leading to these results has partially received funding from the European Commission within the Seventh Framework Programme as part of the SafeAdapt project (grant agreement 608945) and from the Bavarian Ministry of Economic Affairs and Media, Energy and Technology.

References

1. Di Natale, M., Sangiovanni-Vincentelli, A.: Moving from Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools. Proc. of the IEEE **98**(4), 603–620 (2010)
2. Durisic, D., Staron, M., Tichy, M., Hansson, J.: Evolution of Long-Term Industrial Meta-Models. In: 40th EUROMICRO Conf. on Software Engineering and Advanced Applications, pp. 141–148 (2014)
3. Gorcitz, R., Kofman, E., Carle, T., Potop-Butucaru, D., Simone, R.d.: On the Scalability of Constraint Solving for Static/Off-Line Real-Time Scheduling. In: Proc. of 13th Int. Conf. on Formal Modeling and Analysis of Timed Systems (FORMATS), pp. 108–123 (2015)
4. Graydon, P., Bate, I.: Safety Assurance Driven Problem Formulation for Mixed-Criticality Scheduling. Proc. of 1st Int. Workshop on Mixed Criticality Systems (WMC) pp. 19–24 (2013)
5. Gurobi Optimizer Reference Manual: (2017). URL `http://www.gurobi.com`
6. Hamann, A., Ernst, R.: TDMA Time Slot and Turn Optimization with Evolutionary Search Techniques. In: Proc. of Conf. on Design, Automation and Test in Europe (DATE), pp. 312–317 (2005)
7. ISO 26262: Road Vehicles Functional Safety (2011)
8. Kanekawa, N.: Dynamic Autonomous Redundancy Management Strategy for Balanced Graceful Degradation. In: Proc. of IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, pp. 18–23 (1994)
9. Kothmayr, T., Kemper, A., Scholz, A., Heuer, J.: Synthesizing Schedules through Heuristics for Hard Real-Time Workflows. In: IEEE Int. Conf. on Industrial Technology (ICIT), pp. 1937–1944 (2015)
10. Lin, C., Kaldewey, T., Povzner, A., Brandt, S.A.: Diverse Soft Real-Time Processing in an Integrated System. In: Proc. of 27th IEEE Real-Time Systems Symposium (RTSS), pp. 369–378 (2006)
11. Ruiz, A., Juez, G., Schleiss, P., Weiss, G.: A Safe Generic Adaptation Mechanism for Smart Cars. In: Proc. of 26th IEEE International Symposium on Software Reliability Engineering (ISSRE) (2015)
12. SafeAdapt Project: URL `http://www.safeadapt.eu`
13. Sagstetter, F., Andalam, S., Waszecki, P., Lukasiewycz, M., Sthle, H., Chakraborty, S., Knoll, A.: Schedule Integration Framework for Time-Triggered Automotive Architectures. In: 51st ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2014)
14. Steiner, W.: An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks. In: 31st IEEE Real-Time Systems Symposium, pp. 375–384 (2010)
15. Windsor, J., Deredempt, M.H., De-Ferluc, R.: Integrated Modular Avionics for Spacecraft. In: Proc. of 30th IEEE/AIAA Digital Avionics Systems Conference (DASC), pp. 1–16 (8A6) (2011)
16. Yeh, Y.C.: Triple-Triple Redundant 777 Primary Flight Computer. In: Proc. of IEEE Aerospace Applications Conference, vol. 1, pp. 293–307 (1996)