Fail-Operational Automotive Software Design Using Agent-Based Graceful Degradation

Philipp Weiss
philipp.weiss@tum.de
Technical University of Munich

Andreas Weichslgartner
andreas.weichslgartner@audi.de
Audi Electronics Venture GmbH

Felix Reimann
felix.reimann@audi.de
AUDI AG

Sebastian Steinhorst
sebastian.steinhorst@tum.de
Technical University of Munich

***Abstract*—Ensuring fail-operational behavior is critical to enable autonomous driving. With the absence of a driver as a fallback in a failure scenario it will not be sufficient to use state-of-the-art fail-safe approaches. Here, instead of costly hardware redundancy, graceful-degradation can be used by repurposing the allocated resources of non-critical applications for safety-critical applications. However, solving the mapping problem with a state-of-the-art design-time analysis leads to semi-static solutions, where the mapping is fixed and the task activation is chosen at run-time. Therefore, such solutions are unsuited for future automotive architectures that will be highly customizable and which will include frequent software updates. In this paper we introduce and analyze the effectiveness of an agent-based approach that finds application mappings at run-time, ensures the fail-operational behaviour of safety-critical applications by using graceful degradation, and reconfigures itself after ECU failures. Our results indicate that the number of tolerated ECU failures until a safety-critical application fails can be significantly improved without adding any redundant hardware resources.**

I. Introduction

In many applications for autonomous driving, a safe state can not be reached by deactivation and isolation. Thus, existing fail-safe and fail-silent technologies are not sufficient. In a fail-safe system the focus is to enable a safe shutdown state in case of a failure, while in a fail-operational system the safety-critical functionality has to maintain operational. Thus, for a fail-operational behaviour, redundancy of the corresponding components is required [1]. However, adding redundant hardware resources to keep safety-critical applications fail-operational is costly. Combined with the increased resource demand of autonomous driving functions, hardware costs would increase significantly.

At the same time, the automotive industry sees itself confronted with an increasing amount of customer needs and requirements. Automotive software systems will be highly customizable and customers will demand the latest functionality by over-the-air software updates. Automotive companies have to integrate more and more applications into their E/E architecture while each system will consist of a unique and customized configuration.

As a consequence and to cope with increasing costs and complexity, applications are being integrated on more powerful multicore control units. This leads to a consolidation of existing electronic control units (ECUs) and a more centralized E/E architecture [2]. We expect this trend to continue, such that future E/E architectures might consist of only a few, powerful central controllers.

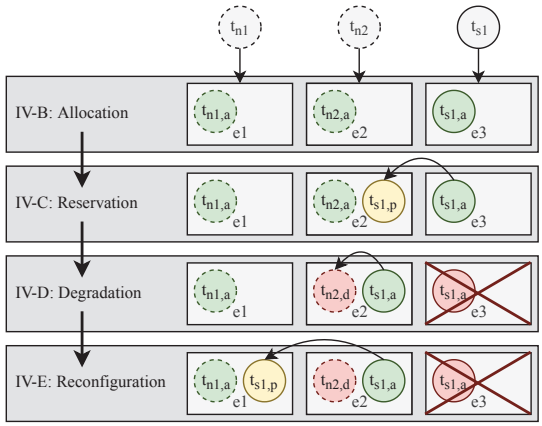


Figure 1: The safety-critical task t_{s1} and the two non-critical tasks t_{n1} and t_{n2}, each being part of a distinct application, are deployed on three ECUs using our agent-based approach as will be described in Section IV. The task states are depicted in green (active), yellow (passive) and red (failed/deactivated).

By contrast, from a software perspective, this trend leads to a decentralization. In comparison to state-of-the-art monolithic ECU software, future software will be designed modular. Such a modular design will allow dynamic shifting of software components at run-time, including activation and deactivation of components on different ECUs. This perspective allows new strategies to enable fail-operational behaviour.

Instead of using active redundancy, where a task replica is actively executed, or passive redundancy, where the system is oversized such that sufficient resources are available once a passive task is started, graceful degradation can be applied. Here, passive redundant tasks with higher priority can reuse the allocated resources of tasks with lower priority. Once a passive task with higher priority is started, lower priority tasks are disabled to free resources. This way, non-critical functionality of an automotive system can be degraded in a failure scenario to keep safety-critical functionality [3]. The advantage of such an approach is that existing hardware resources in the system can be repurposed at run-time to lower the hardware costs.

State-of-the-art design-time methods such as presented in [3] are not applicable for highly customizable automotive

With the support of the Technische Universität München – Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement n° 291763.

systems as they require a re-evaluation for every change in the software system. Furthermore, for each single failure combination, a configuration for an optimal mapping and task activation has to be evaluated and stored. With customizable and frequently updated software, an evaluation of every unique system configuration will not be possible.

To cope with the high amount of possible configurations, we need to make the system self-aware, which can be enabled by applying agent-based strategies. In an agent-based system, the control of the system is decentralized, such that there is no single-point of failure. In the field of dynamic mapping, an agent has the responsibility to find the mapping of a single application or task at run-time. In this paper, we combine graceful degradation with an agent-based system and make the following contributions:

- In the domains of dynamic mapping approaches and graceful degradation we analyze related work in Section II and identify that combining both approaches has not been considered in literature yet.
- Based on the system model introduced in Section III, we present our agent-based approach, which is depicted in Figure 1, to find task mappings and activations at run-time in Section IV. By using graceful degradation, the fail-operational requirements of safety-critical applications can be satisfied. Here, the dynamic nature of the agent-based system allows an easy reconfiguration to re-establish the fail-operational behaviour after ECU failures.
- We use our in-house developed simulation framework to evaluate the approach in Section V.

II. Related Work

The related work for our approach can be mainly separated in the two domains of graceful degradation and dynamic mapping approaches. To the best of our knowledge there is no work that combines the aspect of graceful degradation with an agent-based or a dynamic mapping approach.

A. Graceful degradation

In [4], a utility function is defined to capture the utility of a system in different degradation modes. Furthermore, a framework is introduced that aims at finding properties of systems to enable graceful degradation. The authors in [5] present a degradation-aware reliability analysis in which degradation modes are used to differ between tasks of different safety levels. A design space exploration is used to optimize the reliability of the degradation modes. At run-time, an algorithm observes the resource states and chooses the next valid task mappings. The authors in [3] present a design-time analysis to find a valid mapping of mixed-critical applications. The algorithm respects fail-operational requirements and degrades the system based on priorities.

Both approaches in [5] and [3] use a design-time analysis and, hence, are restricted regarding the applicability on highly customized automotive systems.

B. Dynamic mapping

The authors in [6] introduce a decentralized mapping algorithm for NoC architectures, where tasks are mapped by predecessor tasks. The mapping algorithm considers constraints such as computational capacities and optimizes the routing based on the chosen goal function. In [7], the authors present an agent-based run-time mapping algorithm for NoC-based systems. In this work cluster agents and global agents negotiate with each other and use heuristics to find an optimized mapping for tasks. The authors in [8] investigate different heuristics and use a centralized manager processor to optimize task mappings at run-time. In addition to pure dynamic approaches there is research on hybrid mapping schemes, which combine design-time analysis and run-time reconfiguration as presented by the authors in [9]. The authors also consider the aspect of task migration, where resources have to be allocated at run-time to migrate the task. Instead of using a reactive scheme, the authors in [10] change the mapping at run-time by proactively migrating tasks to prevent imminent hazards. By contrast, in our approach, we find task mappings to establish passive redundancy, in order to react to failures.

In summary, the discussed state-of-the-art hybrid mapping approaches are restricted for use in highly customized automotive systems as they contain a design-time analysis. Centralized approaches such as in [8] introduce single-point-of-failures into the system. Furthermore, none of the existing dynamic or hybrid mapping approaches support graceful degradation and, hence, they are inapplicable to achieve efficient fail-operational behaviour of safety-critical applications.

III. System Model

We model our system by an architecture, which includes a set of ECUs E. We assume that all ECUs communicate via an Ethernet gateway such that a direct communication between each ECU is possible. Each of the ECUs $e \in E$ has a CPU budget $C(e)$ and each of the bi-directional links $l \in L$, that connects an ECU e with the gateway, has a bandwidth budget $BW(l)$.

Our system software consists of a set of applications $a \in A$, where each application a can be modeled by an acyclic, directed, bipartite application graph $G_a(\mathcal{V}_a, \mathcal{E}_a)$. The vertices $\mathcal{V}_a = T_a \cup M_a$ consist of a set of *tasks* $t \in T_a$ and a set of *messages* $m \in M_a$. The edges in $\mathcal{E}_a$ connect a message with a task. Each message has exactly one predecessor and at least one successor.

Furthermore, we assume that the resource consumption of the CPU $c(t)$ per task is known. Similar we assume the bandwidth requirement $bw(m)$ of a message m as given.

Our mixed-critical system consists of a set of safety-critical applications A_S and a set of non-critical applications A_N. We assume that the safety-critical applications have to fulfill fail-operational requirements and, thus, have to be robust against single failures. In this paper we consider permanent ECU failures. An application is considered to behave fail-operational if all of its tasks are still operational and able to properly communicate after the failure of any ECU. If at least one task of an application fails, its application can not operate correctly and all of its remaining tasks can be deactivated.

IV. Agent-Based Degradation

To cope with the high amount of customized configurations in future automotive architectures, we investigate the effectiveness of applying agent-based strategies on task level to achieve a gracefully degrading system behaviour.

In contrast to active redundancy, where redundant tasks would actively run and use CPU resources, passive redundant tasks only reside on the memory. Only when they are activated and replace a failed task, they will require the same amount of resources. Thus, it has to be ensured that on startup of the passive redundant task sufficient resources are available. Instead of allocating the required resources without using them, graceful degradation can be used to deactivate other less critical tasks in order to free the required amount of resources. With this approach the system looses some of its non-critical functionality. On the other hand, this allows to save costs as our graceful degradation approach completely avoids the computational overhead which would be induced by active redundant tasks.

The fault-tolerant time interval (FTTI) describes the time that a fault can be present in the system before a hazard occurs and has to be determined for each safety goal according to the ISO 26262 [11]. We assume that all tasks can be restarted within their assigned FTTI and, thus, focus on the aspect that sufficient resources have to be provided once a passive redundant task is activated in order to ensure a predictable system behaviour. From a timing perspective, to achieve the restart within the FTTI, it is necessary for a redundant task to operate on the same data as the active task. For passive redundant tasks, a snapshotting approach has to provide the active task's status in a periodic fashion such that a restart within the FTTI can be ensured.

A. Agent-based system

With our agent-based methodology on task level we provide a way to ensure that all safety-critical tasks in the system have a passive redundancy and, once they are started, sufficient resources are provided to execute the task. Furthermore, the approach ensures that the communication with preceding and succeeding tasks is maintained. To mitigate multiple failures, the system is able to reconfigure and re-establish the redundancy of safety-critical tasks. The system has a predictable behaviour as the mapping ensures the reservation of sufficient resources for the passive redundant tasks, such that they are able to start in a failure scenario by disabling non-critical tasks.

In our approach both active and passive tasks are wrapped with an agent, which is in the following referred to as a *passive* or *active task agent*. The task agents are responsible for the proper execution of their respective task and for fulfilling their fail-operational requirements. The task agents themselves are always active and able to react on failures. We also use the task agents to find a valid mapping for both the active and the passive tasks. To achieve a dynamic behaviour, the task agents are able to move on the system from one ECU to another.

Furthermore, each of the ECUs is running an *ECU agent*, which starts the task agents on startup. In addition, they handle the requests from task agents to start a new redundant task agent or to move to the corresponding ECU. Note that, in contrast to a design-time optimization, agent-based approaches can dynamically and continuously optimize the mapping with respect to metrics such as link load at run-time.

The flow of our agent-based approach to find task mappings and activations at run-time is depicted in Figure 1 and includes the following steps:

- Allocation IV-B: In the initial mapping process the active task agents allocate resources to find a valid mapping.
- Reservation IV-C: The task agents reserve resources at other agents, which determines how the system will be degraded.
- Degradation IV-D: As an immediate failure reaction passive tasks are started and the system is being degraded.
- Reconfiguration IV-E: By repeating the reservation process, the fail-operational behaviour can be re-established.

B. Resource allocation

Initially, the ECU agents start all task agents depending on a configuration. On startup, the active task agents send requests to the ECU agents to allocate CPU resources and link resources for their incoming messages. The amount of allocated CPU resources $c_{alloc}(t, e)$ and link resources $bw_{alloc}(m, l)$ is stored at the corresponding ECU agent. If it is not possible to allocate enough resources for a mapping on the current ECU, the task agents will request other available ECU agents. As soon as a valid mapping could be found, a task agent moves to the corresponding ECU and starts its task. Succeeding task agents wait on their predecessors to find a valid mapping in order to allocate the correct link resources for their incoming messages. This allocation process ensures that the allocated CPU resources of all tasks in the system do not exceed the CPU budget of any ECU:

$$\forall e \in E : \sum_{a \in A} \sum_{t \in T_a} c_{alloc}(t, e) \leq C(e) \tag{1}$$

Similar it is ensured that the allocated bandwidth of all messages do not exceed the bandwidth budget of any link:

$$\forall l \in L : \sum_{a \in A} \sum_{m \in M_a} bw_{alloc}(m, l) \leq BW(l) \tag{2}$$

C. Resource reservation

Any safety-critical task agent will start a redundant passive task agent on a different ECU. The responsibility of the passive task agents is to ensure that sufficient resources are freed in case the task has to be activated. For that, task agents can *reserve* resources, that have been previously allocated, at non-critical task agents or so far unused resources at ECU agents. The agents at which the resources are reserved *promise* to free the resources if requested. If a passive task agent has to activate its task, it *claims* the reserved resources at the corresponding agents. Once a promising agent frees the claimed amount of resources it deactivates its task. Similar to the allocation process succeeding passive task agents wait on their predecessors to find a valid mapping and allocate a route to it.

Furthermore, active task agents have to reserve a second route to preceding passive task agents (next to the allocated routes to the preceding active task agents) to ensure that a valid route is available at any time. On the other hand, assuming that only one ECU fails at a time, passive task agents only have to reserve one route to either the preceding passive or the preceding active task agents. If the active task agent and the preceding active task agent have the same mapping, they would both fail at the same time. In this case the passive task agent only needs to reserve a route to the preceding passive

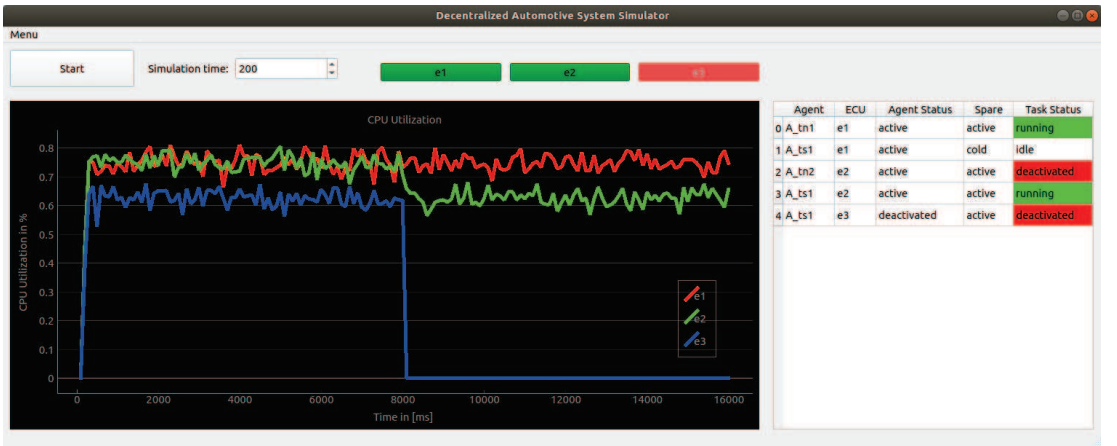


Figure 2: Simulation framework performing simulation of the example described in Figure 1. The plots show the CPU utilization of the three ECUS e_1 (red), e_2 (green), and e_3 (blue). The table on the right displays the situation at the end of the simulation. After the failure of ECU e_3 at $8000ms$, task t_{s1} restarted on ECU e_2, leading to the shutdown of t_{n2}. As t_{s1} requires slightly less resources than t_{n2}, the change can be examined in the plot.

task agent. If the active task agent and the preceding active task agents have different mappings, only one of them can fail at a time. Thus, for the case that the active task agent fails, the passive task agent only needs to reserve a route to the preceding active task agent.

D. Degradation

Once an ECU failure is detected, passive task agents, that lost their active task agent, immediately claim their resources, update the allocation status at the ECU agent, and start their task. Promising task agents, whose resources are being claimed, free the resources and deactivate their task. In addition, the allocation and reservation status at all task agents and ECU agents is updated, such that the allocated or reserved resources of failed agents are not lost.

E. Reconfiguration

For any task agent that lost its redundant counterpart or a resource reservation, the procedure from Section IV-C can be repeated. Here, the advantage of the agent-based approach is that no additional algorithm is required.

With this approach, the task agents ensure together the fail-operational behaviour of their application. The task-based reservation of resources has two specific advantages. First, the decision about which tasks are shut down in a degradation scenario does not have to be met in the time critical phase after a failure. Second, this approach allows to predict the system behaviour. If all passive task agents of an application reserved the required resources, a fail-operational behaviour can be guaranteed.

F. Example

In our example in Figure 1, a safety-critical and two non-critical tasks are deployed on three ECUs using our agent-based approach. After each task agent found an ECU and allocated the resources, the safety-critical task agent responsible for $t_{s1,a}$ starts a passive task agent on e_2. This passive task agent reserves the required resources at the task agent responsible for $t_{n2,a}$. After the failure of ECU e_3, the passive task $t_{s1,p}$ on ECU e_2 is immediately started and its task agent claims its resources at the task agent of $t_{n2,a}$, who deactivates its task. In the last step, the fail-operational behaviour of task $t_{s1,a}$ is re-established by starting another passive task agent on e_1 and repeating the reservation process.

V. Evaluation

We have implemented the approach described in Section IV in our in-house developed time-discrete and event-based simulation environment. The framework has been developed to simulate an automotive hardware architecture and system software according to our model as described in Section III. We use this framework to evaluate our agent-based approach from Section IV.

A. Simulation framework

The system parameters describing the hardware architecture and system software can be provided by a specification which uses the XML schema for specifications from the OpenDSE framework [12]. For the simulation environment we chose a process-based Discrete-Event Simulation (DES) architecture based on the SimPy framework [13].

To dynamically activate, deactivate, and move tasks on the platform at run-time, we implemented a middleware which is based on SOME/IP [14], an automotive middleware solution. This middleware includes a decentralized service-discovery to dynamically find services in the system and a publish/subscribe scheme to publish and subscribe to events. In addition, this middleware allows remote procedure calls. All tasks in the system communicate via this middleware and are modelled as clients and/or services. Tasks that have outgoing edges in our

application graph G_a are offered as a service to the system, which also publish their messages as subscribable events. Tasks that have ingoing edges behave as clients that request the corresponding services and subscribe to the events. This service-oriented approach allows a dynamic reconfiguration of the system software at run-time, such that tasks can be restarted on other ECUs and still be found by their subscribers.

Hardware access to a CPU or Ethernet link is managed by interchangeable schedulers. For the simulation, a static-priority preemptive scheduler was used for access to CPUs and a static-priority non-preemptive scheduler for access to Ethernet links. Tasks in the system are either triggered periodically if they are the anchor task of an application otherwise on the arrival of incoming messages. Once a task is triggered, it is being scheduled for execution on the CPU. After it has been granted access to the resource, it keeps the resource busy and sends out its messages as soon as it has finished execution. The unicast messages are put on the link and forwarded by the central switch. On arrival at its destination, succeeding tasks which are waiting for the message are triggered.

Furthermore, we implemented the proposed agent-based system from Section IV in our simulation framework. The agents use the same middleware and network interface for communication as the tasks in our system. Note that the major load imposed by the agents onto the system is happening during the initialization phase when all agents allocate and reserve resources. Furthemore, the size of the agent messages is relatively small, mostly consisting of a few bytes, compared to the message size of typical automotive applications. During the normal execution phase the agents do not impact the system as they are only triggered by ECU failures, where the system has to immediately react on the failure and be reconfigured. With example applications implemented, a more detailed analysis of the overhead imposed by the agents on the system is possible.

To simulate ECU failures, the framework offers the possibility to shut down ECUs. ECU failures are detected with periodic heartbeats and watchdogs. Each ECU has a running service to offer its heartbeat, whose periodic event is subscribed by the watchdogs of other ECUs.

Figure 2 shows our simulation framework performing the simulation of the example from Figure 1, which has been described in Section IV-F. From the CPU utilization on the three ECUs and the status information shown, it can be observed that t_{n2} was shut down on ECU e_2 and instead t_{s1} has been restarted, which was formerly running on ECU e_3.

B. Results

To evaluate our agent-based approach we used a simulation setup of 6 ECUs and 25 applications, of which each consisted of at least 10 tasks. We used the OpenDSE framework [12] to generate synthetic applications with workloads based on typical automotive applications. All applications had a period of 10 ms and the message sizes were set to 1500 bytes. The link speed of all links was set to 100 Mbit/s. To obtain feasible mappings in the initial mapping process, the combined CPU usage of all applications was set to 90% of the available system resources. We equalized the sum of required required computational resources $c(t)$ within each application in order to obtain a better comparison. Although the actual workload is varied

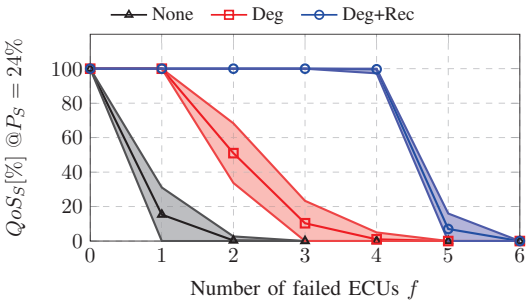


Figure 3: Simulated results with $|A| = 25$, $\forall a \in A : |T_a| \geq 10$, $|E| = 6$, $P_S = 24\%$ and 50 runs per configuration. Our agent-based approach using both degradation and reconfiguration improves the percentage of operational safety-critical applications QoS_S and the amount of ECU failures tolerated by the safety-critical applications significantly.

with a random distribution at run-time, we use the required computational resources $c(t)$ as the worst-case estimation for our allocation and reservation process.

Each configuration was run 50 times and the results (Figure 3 and Figure 4) show the corresponding mean values and standard deviation. We conducted the simulations with an Intel Xeon Gold 6130 CPU consisting of 16 cores running at 2.1 GHz and 128GB of RAM. The simulations were run for a simulation time of 6000 ms. To simulate the failures we successively shut down a random ECU every 1000 ms. On average a single simulation run with a setup of 6 ECUs and at least 250 tasks took about 119.7 s on one of the cores.

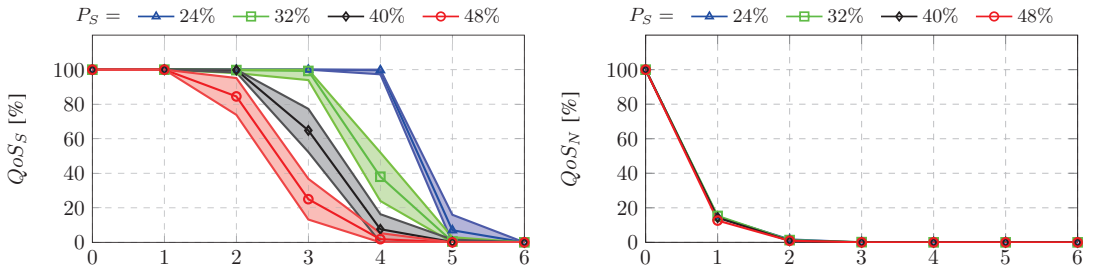
For our evaluation we define the metric

$$QoS_S(f) = \frac{|A_{S,f}|}{|A_S|}, f \in [0; |E| - 1] \tag{3}$$

where $A_{S,f}$ is the set of safety-critical applications which are running after the f-th failure and $QoS_S(f)$ the percentage of operational safety-critical applications after the f-th failure. Similar, we use the notation $A_{N,f}$ and $QoS_N(f) = \frac{|A_{N,f}|}{|A_{N,0}|}$ for the non-critical applications. Furthermore, we define $P_S = \frac{A_S}{A}$ as the percentage of safety-critical applications in the system.

Our simulation results show that the amount of tolerated ECU failures increases significantly with our agent-based approach using both degradation and reconfiguration (Figure 3). In this scenario, the first degradation of the safety-critical system occurs in the majority of the cases after the 5-th ECU failure compared to one failure tolerated by the approach without reconfiguration and zero failures tolerated without any replication.

Furthermore, we can observe that with increasing percentage P_S of safety-critical applications in the system, less failures can be tolerated until a safety-critical application fails (Figure 4a). This is plausible as an increasing amount of safety-critical applications has to share the same amount of resources. It can be also noticed that configurations with higher P_S reach a QoS_S close to 0% earlier while there would be still resources available. This comes from the fact that with more safety-critical tasks in the system, less passive task agents are able



a) Number of failed ECUs f b) Number of failed ECUs f

Figure 4: Simulated results with $|A| = 25$, $\forall a \in A : |T_a| \geq 10$, $|E| = 6$ and 50 runs per configuration. With an increasing percentage of safety-critical applications P_S in the system, the percentage of operational safety-critical applications QoS_S decreases earlier and less ECU failures can be tolerated by the safety-critical system. There is no measurable impact of P_S on the percentage of operational non-critical applications QoS_N.

to reserve resources. Once a passive task agent was not able to reserve sufficient resources it is shut down and will not be restarted. Thus, even if resources would become available with the shutdown of other safety-critical applications after the next ECU failure, no new passive task agents are started.

There is no observable impact of P_S on the percentage of operational non-critical applications QoS_N (Figure 4b) as the curves behave relatively similar. We explain this with the fact that the tasks of an application are distributed in the system and that an ECU failure leads with a high probability to the shutdown of multiple applications. This effect overlaps the number of applications that are being shut-down due to degradation. We conclude that in a system with highly distributed tasks the degradation has little impact on $QoS_N(f)$ as the non-critical applications would fail anyway.

Overall, the results indicate that our presented approach is able to significantly improve the tolerance of safety-critical applications against ECU failures. This improvement depends on the percentage of resources allocated by all safety-critical applications in the system. With our current approach a theoretical maximum of 50% CPU resources can be allocated by safety-critical applications in order to tolerate at least one fault as the remaining 50% have to be reservable by passive task agents. Concerning the link resources, assuming fully utilized links, a theoretical maximum of 33.33% of the resources can be allocated by safety-critical applications as the link resources have to be reserved twice by the task agents. Note that our graceful degradation approach is only restricted with regards to the maximum reservable resources, but completely avoids the computational and communication overhead which would be induced by using active redundancy.

VI. Conclusion

In this paper we have introduced an agent-based approach utilizing graceful degradation to ensure fail-operational behaviour of safety-critical automotive applications. The system finds task mappings and activations at run-time and is able to predict if the fail-operational behaviour of an application can be guaranteed. Furthermore, the agent-based system is able to reconfigure itself after ECU failures and re-establish fail-operational behaviour. In an experimental evaluation we have shown that the number of tolerated ECU failures until a safety-critical application fails, can be improved significantly without using additional hardware resources. This approach is a first step towards a fully adaptive system behaviour to cope with an increasing number of customized software configurations.

References

[1] A. Kohn *et al.*, "Fail-operational in safety-related automotive multi-core systems," in *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2015.

[2] S. Saidi, S. Steinhorst, A. Hamann, D. Ziegenbein, and M. Wolf, "Future automotive systems design: Research challenges and opportunities: Special session," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2018.

[3] K. Becker and S. Voss, "Analyzing graceful degradation for mixed critical fault-tolerant real-time systems," in *18th International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2015, pp. 110–118.

[4] C. P. Shelton, P. Koopman, and W. Nace, "A framework for scalable analysis and design of system-wide graceful degradation in distributed embedded systems," in *Proceedings of the 8th Int. Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)*. IEEE, 2003.

[5] M. Glaß, M. Lukasiewycz, C. Haubelt, and J. Teich, "Incorporating graceful degradation into embedded system design," in *Proceedings of the Conference on Design, Automation and Test in Europe*. IEEE, 2009.

[6] A. Weichslgartner, S. Wildermann, and J. Teich, "Dynamic decentralized mapping of tree-structured applications on NoC architectures," in *Proceedings of the Fifth ACM/IEEE International Symposium*. IEEE, 2011.

[7] M. Faruque, R. Krist, and J. Henkel, "Adam: Run-time agent-based distributed application mapping for on-chip communication," in *Proceedings of the 45th Annual Design Automation Conference*. IEEE, 2008.

[8] E. L. de Souza Carvalho, N. L. V. Calazans, and F. G. Moraes, "Dynamic task mapping for MPSoCs," *IEEE Des. Test*, vol. 27, no. 5, 2010.

[9] B. Pourmohseni, S. Wildermann, M. Glaß, and J. Teich, "Predictable run-time mapping reconfiguration for real-time applications on many-core systems," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. ACM, 2017.

[10] E. Rambo *et al.*, "The information processing factory: A paradigm for life cycle management of dependable systems," in *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. IEEE, 2019.

[11] *ISO 26262, Road vehicles - Functional Safety - Part 1-9*, 1st ed., International Organization for Standardization, 2011.

[12] F. Reimann, M. Lukasiewycz, M. Glaß, and F. Smirnov., *OpenDSE – Open Design Space Exploration Framework*, 2019. [Online]. Available: http://opendse.sourceforge.net/

[13] *SimPy Discrete Event Simulation Library for Python*, 2019. [Online]. Available: https://simpy.readthedocs.io

[14] *Scalable service-Oriented MiddlewarE over IP (SOME/IP)*, 2019. [Online]. Available: http://some-ip.com/