A Reconfiguration Approach for Fault-Tolerant FlexRay Networks

Kay Klobedanz, Andreas Koenig, Wolfgang Mueller
University of Paderborn/C-LAB
Faculty of Electrical Engineering, Computer Science and Mathematics
33102 Paderborn, Germany
Email: kay@c-lab.de, andreask@upb.de, wolfgang@c-lab.de

***Abstract*—In this paper we present an approach for the configuration and reconfiguration of FlexRay networks to increase their fault tolerance. To guarantee a correct and deterministic system behavior, the FlexRay specification does not allow a reconfiguration of the schedule during run time. To avoid the necessity of a complete bus restart in case of a node failure, we propose a reconfiguration using redundant slots in the schedule and/or combine messages in existing frames and slots, to compensate node failures and increase robustness. Our approach supports the developer to increase the fault tolerance of the system during the design phase. It is a heuristic, which, additionally to a determined initial configuration, calculates possible reconfigurations for the remaining nodes of the FlexRay network in case of a node failure, to keep the system working properly. An evaluation by means of realistic safety-critical automotive real-time systems revealed that it determines valid reconfigurations for up to 80% of possible individual node failures. In summary, our approach offers major support for the developer of FlexRay networks since the results provide helpful feedback about reconfiguration capabilities. In an iterative design process these information can be used to determine and optimize valid reconfigurations.**

I. Introduction

The increasing number of ECUs (Electronic Control Units) in modern vehicles implies the necessity of reliable communication between these components. To guarantee a correct system behavior, the dependencies between different tasks must be considered to determine a proper task-to-ECU assignment and configure an appropriate bus communication. FlexRay is the emerging standard for safety-critical automotive systems because of its deterministic behavior, bandwidth capacities and redundant communication channels, which increase safety of such networks. Nevertheless, the failure of a single ECU may result in a malfunction of the whole system. To further increase the fault tolerance of a FlexRay network, such a node failure should be compensated by redundancy. Therefore, a replication and reallocation of tasks is necessary, which implies changes in the communication dependencies and a reconfiguration of the system at run time [1], [2]. Current automotive systems usually consist of ECUs, which host functions and are also hardwired to their target sensors/actuators. A failure of a hardwired node cannot be compensated with redundancy because the connections to the peripherals get lost. Hence, we consider a modified network topology to improve fault tolerance by means of redundancy [1], at which we separate between two types of ECUs: Nodes, we call *peripheral interfaces*, which are wired to sensors and actuators, and just read/write values to/from the bus and dedicated nodes hosting the functional tasks. Since peripheral interfaces do not execute any complex task they only require low hardware capacities, which allows cost-efficient redundancy of these components. However, here we focus on the distributed functional ECUs, which receive and provide their data via FlexRay, and can therefore be considered for redundancy and reconfiguration. Because the FlexRay specification does not allow changes of the schedule at run time [3], we propose a reconfiguration using redundant slots in the schedule and/or combine messages in existing frames and slots to avoid the necessity of a complete bus restart in case of a node failure [1], [2]. Here, we present an approach inspired by a heuristic based on a genetic algorithm (GA) [4], which determines initial configurations and, based on that, calculates valid reconfigurations for the remaining nodes of the FlexRay network in case of a node failure. Finally, the results of our approach support the developer to determine and optimize valid reconfigurations in an iterative design process.

In this paper we introduce FlexRay and related work followed by a detailed description of our approach in Section IV. An evaluation is provided in Section V before the final section concludes with a summary.

II. Related Work

Several publications like [5] and [6] describe heuristics to determine proper configurations and parameterizations for the static segment of FlexRay. The presented approaches assume that the task-to-ECU assignment is statically defined, which significantly decreases the configuration flexibility. Moreover, the authors do not consider reconfigurations and redundancy to increase fault tolerance. In [7] the authors propose a replication of tasks to compensate node failures. Therefore, they determine the reconfiguration capabilities of FlexRay based on these assumptions. This publication also assumes a quite strict predefined task-to-ECU and slot assignment, whereas we maximize flexibility in the configuration. In [4] the authors present a GA for the configuration of FlexRay networks. The publication describes the representation of configurations (task-to-ECU, message-to-frame, and frame-to-slot assignments) as individuals, which are coded as strings. The approach evaluates these individuals with several weighted parameters and determines an optimized configuration by means of selection, crossover and mutation functions. Our approach is inspired by this GA-based heuristic, but we extended it by an optimized analytical methodology, which, additionally to a initial configuration, calculates valid reconfigurations for the remaining nodes of the FlexRay network in case of a node failure.

III. FlexRay Fundamentals

FlexRay was introduced to implement deterministic and fault-tolerant communication systems for safety-critical distributed real-time systems. The protocol makes use of recurring communication cycles as shown in Figure 1. It is composed of a static and an optional dynamic segment. The time-triggered static segment is based on the TDMA (Time Division Multiple Access) protocol. Therefore, the transmission slots of the segment are assigned to a sender node by a globally known synchronously incremented slot-counter. The static segment consist of a fixed initially defined number of equally sized static slots (2 – 1023). The event-triggered dynamic segment of the FlexRay communication cycle is optional. Because we focus on the static segment, the reader is referred to [3] for further details. The size of slots and frames, the cycle length, and several other parameters are defined by an initial setup of the FlexRay schedule, which cannot be changed during run time. The payload segment of a FlexRay frame is composed

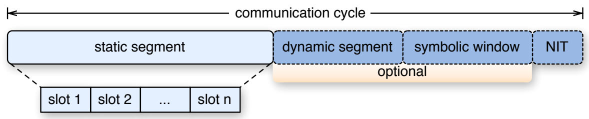


Fig. 1. Components of the FlexRay communication cycle.

of data bytes, which are numbered in ascending order. One or multiple data bytes can be combined to a PDU (Protocol Data Unit), whose data can be accessed by means of a unique ID. This can be used for *frame packing*, which allows different tasks to send their data combined in one frame. For a more detailed description of the FlexRay protocol and components, the reader is referred to [3].

IV. Approach

In this section we present our approach for the configuration and reconfiguration of FlexRay networks based on the static segment.

A. Overview

Our approach determines a capable initial system configuration based on a Directed Acyclic Graph (DAG) (see part B of this section) derived from the task dependencies, the given network topology and the parametrization of the FlexRay network. Here, several sub-steps have to be performed:

- Task-to-ECU assignment,
- Consideration of task dependencies,
- Message-to-frame and frame-to-slot assignment,
- Scheduling and validation for each ECU.

Based on the determined initial configuration our approach also calculates possible reconfigurations for the remaining nodes of the FlexRay network to increase the fault tolerance of such systems in case of a node failure. Figure 2 gives a

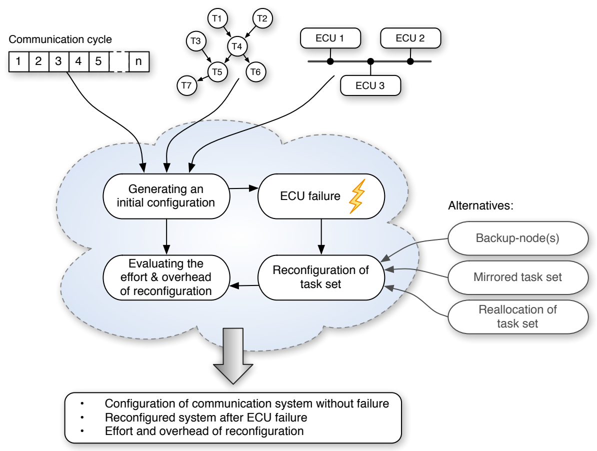

Communication cycle
1
2
3
4
5
n
T1
T2
T3
T4
T5
T6
T7
ECU 1
ECU 2
ECU 3
Generating an initial configuration
ECU failure
Evaluating the effort & overhead of reconfiguration
Reconfiguration of task set
Alternatives:
Backup-node(s)
Mirrored task set
Reallocation of task set
Configuration of communication system without failure
Reconfigured system after ECU failure
Effort and overhead of reconfiguration

Fig. 2. Overview of the (re-)configuration approach.

functional overview of our approach. By means of a predefined FlexRay communication cycle, a given network topology and a DAG representing the dependencies between the distributed tasks it calculates an initial configuration for the system with an appropriate task-to-ECU, message-to-frame and frame-to-slot assignment resulting in a valid schedule. Based on the determined initial setup, we simulate the failure of each separate ECU. To compensate the failure of an arbitrary node, we consider several possible strategies. We already described the migration and activation of redundant task instances on dedicated backup node(s) in [2] and [1]. Here we focus on the reallocation of tasks on the remaining ECUs. The main objective of our approach is to retrieve valid reconfigurations for the remaining system to compensate ECU failures. Therefore, it calculates and validates reconfigurations for the system – i. e. reallocations of the task set on the ECUs and messages on the FlexRay bus – considering the resulting effort and overhead to provide these information as feedback to the system developer.

B. Model Properties

Our model comprises safety-critical distributed systems, which are commonly composed of ECUs connected via a communication bus. The executed task set of an ECU can be assigned statically or changed dynamically during runtime. The communication in a distributed system is realized via local inter-process communication (IPC) or bus communication, e. g. FlexRay. The scheduling in our model is performed by means of earliest deadline first (EDF) [8]. The utilization based schedulability test of EDF for each ECU also validates determined configurations. Each task τ_i of a task set Γ is defined by its execution time C_i and its period T_i as:

$$\Gamma = \{\tau_i(T_i, C_i), \quad i = 1, \ldots, n\}.$$

The mentioned communication dependencies between tasks can be represented by means of a DAG as shown in Figure

3. Such a graph $G = (\Gamma, M)$ is composed of vertices (task set Γ) and edges (set of messages $M = \{m_1, \ldots, m_m\}$). The dependencies in the example of Figure 3 imply that the execution of τ_1 has to be finished before τ_3 gets executed – τ_3 needs input data from τ_1 to execute. The tasks and the whole DAG have a period of $1000\mu s$. Consequently, the execution of the whole DAG must be finished in this time. Our model considers different messages sent from one task [4] – τ_1 sending m_1 and m_2 –, which have to be assigned to the frames/slots separately and the concept of each task sending only one message for all receivers ($m_1 = m_2$) [9]. For complex systems, a DAG can be composed of several

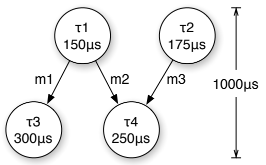


Fig. 3. Example of a DAG.

subgraphs with different periods. Like in [4] we presume for all periods T_i and the maximum period $T_{max} = max(T_i)$:

$$T_i \cdot 2^{x_i} = T_{max}, \quad x_i \in \mathbb{N}_0, \quad i = 1, \ldots, n.$$

The resulting hyperperiod T_{max} avoids scheduling over multiple communication cycles. For example, the periods of all subgraphs are $4ms$, $2ms$, $1ms$, …for $T_{max} = 4ms$. Special messages are transitions between two subgraphs, which we label as *split-messages*. Our approach schedules these split-messages as follows. If $T_{send} < T_{recv}$, τ_{send} writes messages with T_{recv} because τ_{recv} reads with a lower frequency. If $T_{send} > T_{recv}$, τ_{send} sends with T_{send}. In this case it is important that τ_{send} provides the message soon enough to minimize obsolete data.

We define three different types of task sets in a dependency graph, as proposed in [4]. Γ_{in} is the set of tasks, which provide input data to the system and have no predecessor. Γ_{out} defines the tasks, which provide output data and have no successors. Γ_{mid} are all other nodes of the DAG, which have predecessors and successors. All tasks in Γ_{mid} are incorporated in the reconfiguration. Furthermore, we assume that Γ_{in} and Γ_{out} can also be considered for the reconfiguration, because they are executed on reconfigurable ECUs and not on peripheral interfaces (see Section I).

The primary parameters of the modeled FlexRay schedule are a bandwidth of 10MBit/s a cycle length equal to the maximum period of the considered tasks ($T_{cycle} = T_{max}$) [4], a variable slot length T_{slot}, and an according number of slots ($\#slots = T_{cycle}/T_{slot}$).

C. Initial Configuration

To calculate a valid initial configuration, we developed a heuristic based on a genetic algorithm [4]. Because the initial configuration just has to be valid and not optimal, we omit the optimizing crossover and mutation functions from [4] in our resulting *pseudo-genetic* algorithm.

Using this algorithm we focus on the calculation of individuals as candidates for the initial configuration. Figure 4 shows the sub-steps of the calculation process. The assignment of tasks to ECUs is followed by the assignment of messages to frames in static slots of the communication cycle and the scheduling of the tasks on each ECU, which also validates the individual. Finally, each individual gets evaluated by means of a *fitness-function*. Based on the input containing the model

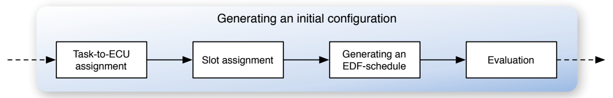
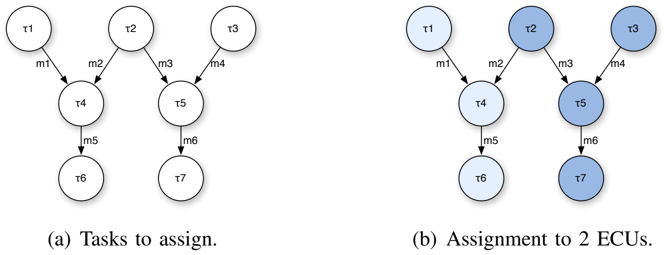

Generating an initial configuration
Task-to-ECU assignment
Slot assignment
Generating an EDF-schedule
Evaluation

Fig. 4. Process flow for the calculation of an individual.

properties, our approach calculates an initial *generation* of individuals as described above. Each calculated individual is compared to the existing individuals of the current generation regarding the task-to-ECU assignment to guarantee a heterogeneous solution-space. For further calculated generations the selected individuals are analyzed and compared analog to [4]. After calculating one or more generations, the best individual of the fitness-based sorted generation will be selected as the initial configuration for further application. To speed up the search process, we also realized a method to use the first valid individual.

The first step to calculate an individual is the task-to-ECU assignment, which is restricted by the model properties and communication dependencies. Because EDF is used for the scheduling, the basic restriction is the maximum utilization ($U = \sum_{i=1}^{n} C_i/T_i \leq 1$) for n tasks on each ECU. Hence, our approach supports arbitrary randomized or optimized strategies to determine the task-to-ECU assignment, we propose an optimized assignment based on the task dependencies. This strategy uses the model information about task dependencies to maximize the local IPC and minimizes the number of messages over the FlexRay bus to reduce the number of occupied slots. Figure 5(a) shows an example dependency



(a) Tasks to assign. (b) Assignment to 2 ECUs.

Fig. 5. Example dependency graph for optimized task-to-ECU assignment.

graph of tasks to assign. First of all, our approach checks $\Gamma_{in} = \{\tau_1, \tau_2, \tau_3\}$. Is any of the tasks in this set or the whole set not assigned to any available ECU the assignment is performed randomly based on the utilization of each ECU.

By means of this initial assignment the algorithm uses depth first search to assign succeeding tasks to the same ECU as its predecessors. If this is not possible because of the utilization the algorithm chooses another ECU randomly. Figure 5(b) shows the resulting assignment for 2 ECUs. The tasks in Γ_{in} are randomly assigned – marked lighter and darker blue. Along the communication path the assignment is proceeded as described resulting in the assignment shown in Figure 5(b).

Every message m_i that is not handled via IPC has to be assigned to a static slot in the communication cycle. But not every message needs a dedicated slot, because multiple messages from one task (node) can be packed in one frame (see Section III). This significantly increases the flexibility of the configuration. The determined release times and deadlines for the tasks must be aligned if they are communicating via FlexRay. Based on the initial values, they are recalculated to determine the first and last possible assignable slot for a transmission. By means of the equations for EDF with precedence constrains [8], we calculate the release time r_m of a message as $r_m = \max(r_{send} + C_{send})$ and the deadline as $d_m = \min(d_{recv} - C_{recv})$. After initializing r_m and d_m, we determine available slots for the message. Figure 6 illustrates the possible delay between the finishing of a task τ_1 and the beginning of a transmission in an available slot. Starting

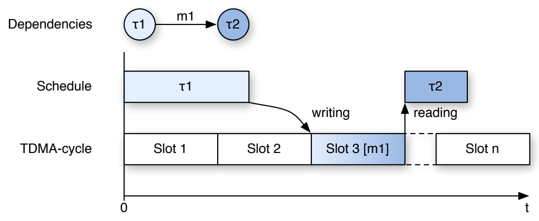


Fig. 6. Example for a valid slot assignment.

the slot-IDs with 1, the first available slot is calculated as $\text{slot}_{first} = \lceil r_m/T_{slot} \rceil + 1$, and the last usable slot is calculated as $\text{slot}_{last} = \lfloor d_m/T_{slot} \rfloor$. The difference between slot_{last} and slot_{first} is the number of possible slots. Our approach randomly chooses a slot, which is free or already transmitting a frame from the same ECU with free capacities. If the period of a sending task is shorter than the communication cycle, multiple slots have to be assigned. To avoid multiple tests per cycle, the tasks are sorted ascending by period length. Thus, only available slots for the first message instance have to be analyzed. As shown in Figure 6, the assignment of a slot changes the release time of the reading task (τ_2). To keep the available slots valid, the values for the release times and deadlines must be updated after every assignment. The updated deadline for the sending task (start of assigned slot) is calculated as $d^* = (\text{slot-ID} - 1) \cdot T_{slot}$. The deadlines of the predecessors are calculated bottom up by means of the DAG. The updated release times of receiving tasks (end of assigned slot) are calculated as $r^* = \text{slot-ID} \cdot T_{slot}$. The release times of the successors are calculated top down. Based on these values r_m and d_m are also updated.

After the assignments are determined, the individual is validated implicitly by a schedule for each ECU. Therefore, every ECU is scheduled with EDF over the hyperperiod (communication cycle length) as described in part B of this section. Hence, tasks with $T_i < T_{max}$ have to be instantiated and scheduled multiple times with corresponding release times and deadlines whose absolute values are calculated from relative values. If all deadlines are met and the schedulability test is positive, the individual is valid. Finally, the evaluation of the individuals in one generation is performed by means of a fitness-function based on several weighted parameters. These parameters are the number of empty slots, which can be increased by IPC or frame packing, the number of unused ECUs, and the last assigned slot. Additionally, several status information for the validity of the task-to-ECU and slot assignments and the final schedule are included.

D. Reconfiguration in Case of a Node Failure

Our approach uses the results of the initial configuration to calculate a reconfiguration in case of a node failure. To avoid a restart of the bus, no new static slots may be assigned to the nodes. Our approach determines valid reconfigurations and evaluates their resulting effort and overhead – i. e. how many additional redundant slots have to be included in the FlexRay schedule. Several strategies for the compensation of node failures are supported. Here, we focus on the reallocation of the tasks on the remaining ECUs. Figure 7 illustrates an

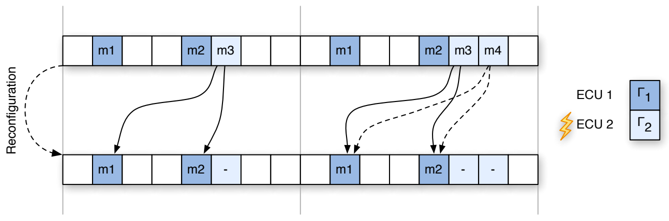


Fig. 7. Reconfiguration for reallocated tasks (Γ_2 on ECU1).

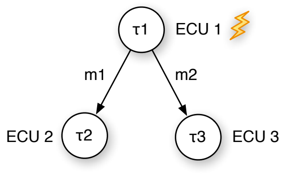
example for two ECUs with their assigned task sets Γ_1 and Γ_2. After a failure of ECU2 several sub-steps have to be performed to get an adequate reconfiguration. In the process the effort and overhead of a reconfiguration by means of additional slots should be minimized. Therefore, it is desirable that, after a reallocation of Γ_2 on ECU1, the messages m_3 and m_4 are transmitted on already used slots. The general sub-steps for a reconfiguration are:

1) Reallocation of the task set to the remaining ECUs considering their particular utilization.
2) Reconfiguration based on the initial configuration. Here, the effort and overhead should be minimized.

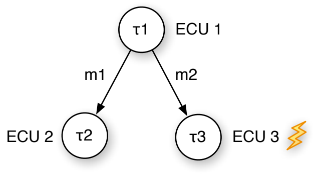
Instead of using the presented pseudo-genetic algorithm for the calculation we propose an optimized analytical methodology for the reallocation of the task set. Primarily, we have to assure that the n remaining ECUs (ECU^i_{ok}) provide sufficient resources to execute the reallocated task set of the failing ECU (ECU_{fail}). The available resources must be sufficient to handle the utilization U of the faulty ECU:

$$U(\text{ECU}_{fail}) \leq \sum_{i=1}^{n} 1 - U(\text{ECU}_{ok}^{i}).$$

If a reallocation is possible, our approach initializes the assignment of the remaining ECUs and slots. Here, we have to consider that slots, assigned to the faulty ECU, are not available anymore. Figure 8(a) shows a failure of ECU1 executing τ_1. The assigned slots for ECU1 are lost and m_1 and m_2 need a reassignment. Furthermore, we have to check the received messages of the reallocated tasks. If – as shown in Figure 8(b) – a failure of ECU3 occurs, it depends on the reallocation of τ_3 if the reconfiguration must assign a slot to m_2 or IPC can be used. The assignments of all tasks and messages not affected



(a) Failure of ECU1.



(b) Failure of ECU3.

Fig. 8. Examples for ECUs failures.

by the node failure are retained unchanged. This results in an initial "empty" reconfiguration reduced by the faulty ECU with consistent assignments, release times and deadlines. Based hereon, our approach calculates an adequate reconfiguration with appropriate ECU and slot assignments.

The assignment of the reallocated task set Γ_{fail} is performed by means of the DAG along the communication paths analog to the initial configuration. Our approach starts with an iteration of $\Gamma_{current} = \Gamma_{in}$. If $\tau_{current} \in \Gamma_{fail}$, this task has to be reallocated. The algorithm proceeds with the updated task set $\Gamma^{*}_{current}$ composed of the successors of the tasks in $\Gamma_{current}$ until every task in Γ_{fail} is processed. We optimize the reconfiguration by means of determining an adequate "best possible" ECU assignment for every task in Γ_{fail}. Figure 9 shows an example DAG for a failure of ECU x and the resulting reallocation of τ_x. Because the assignment is performed top down, it is guaranteed that the predecessors are already processed. The algorithm maximizes the IPC to all connected tasks in the DAG, assuming that probably there will not be determined a proper combination of period, ECU and slot assignment for the messages m_3 and m_4 in the next iterations. Therefore, we determine the hosting ECU for every predecessor and successor. The ECU executing most of these tasks is analyzed for an assignment. In Figure 9 τ_x is connected to tasks executed on ECU1 and ECU2. Because ECU1 hosts two of these tasks it is analyzed for a reallocation of τ_x. If there is no adequate candidate or the utilization of a determined ECU is to high, the task is assigned to an ECU, which offers sufficient capacities and hosts at least one task τ^{*} with the period of $T(\tau^{*}) \leq T(\tau_x)$. Hereby, it is guaranteed for a bus communication that there could be at least one more

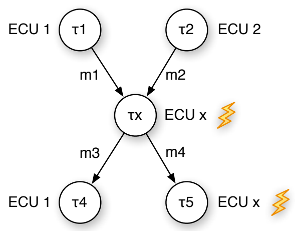


Fig. 9. Example for the determination of an adequate ECU.

task, whose slot is possibly usable by τ_x to send its message via frame packing.

The determination of valid slots is nearly identical to the initial configuration, including the updating of release times and deadlines. But here we distinguish between optimal assignments, where existing frames and slots are occupied (frame packing), and valid solutions with as less as possible additional frames and slots. Figure 10 illustrates this distinction. If m_4 with $T(m_4) = T_{cycle}/2$ is transmitted by ECU1 for an optimal assignment, only slots with ID = $2 + i \cdot \#slots/2$ for $i \in 0, 1$ are suitable. In this case slot 2 and 7 (second instance of m_4). If m_4 is sent by ECU2 and $r(m_4)$ implies slot 4 as earliest transmission possibility then slot 3 and 8 are omitted due to release time constraints and only slot 4 and 9 are applicable. Hence, one additional slot (9) transmitting m_4 is necessary. For the reconfiguration the messages are processed sorted by

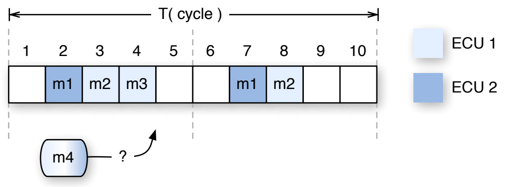
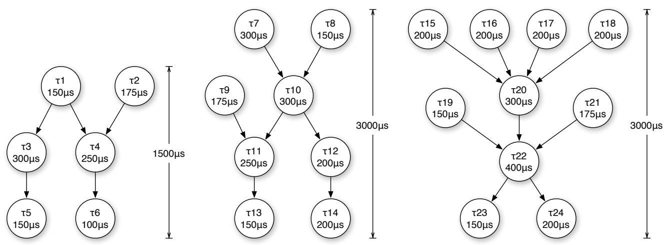


Fig. 10. Example for slot assignment: $T(m_4) = T_{cycle}/2$.

period lengths and release times. The valid slots, determined by means of release time and deadline of m_i, are analyzed sequentially regarding the following properties: Are the current and possible subsequent slots for multiple instances of m_i assigned to the same ECU? Do all considered frames offer enough capacities for the message instances? Candidates are sorted and finally assigned based on the collective number of slots needed for all transmitted instances of m_i in the period of one cycle. The validation of the reconfiguration is also performed by a conclusive EDF scheduling of every ECU, as described in Part C of this section. If the schedulability test is positive, our approach determines a valid reconfiguration with no or a minimal number of additional slots.

V. Evaluation

We evaluated our approach by means of realistic safety-critical automotive real-time systems [9]. Figure 11 shows the DAGs for an EPS (Electric Power Steering), ACC (Adaptive Cruise Control), and TC (Traction Control) system. For a more detailed description of theses systems the reader is referred to [9]. Table I contains the related message information.



(a) EPS (b) ACC (c) TC

Fig. 11. Examples for safety-critical automotive real-time systems [9].

The evaluation is performed based on the model properties described in IV-B. Every task sends only one message to one or multiple receivers. The presented optimized initial task-to-ECU and slot assignments based on task dependencies are evaluated by means of the parameters described in Section IV-C. For the evaluation of the initial methods for the task-to-ECU assignment, we used a complex combined version of the above described graphs as described in [4]. Here, we define 6 ECUs that are dynamically assigned, whereas Figure 12 compares the optimized methodology to a random assignment for 10 configurations on an average, using the best individual out of a generation with 2000 individuals. The optimized methodology was able to generate much more free ECUs. This leads to more IPC and therefore less communication over the FlexRay system. Because the number of slots (80 available slots) is high, the use of frame packing is for both methods not significant. The last occupied slot is further ahead, because of less communication over the FlexRay system. All that distinctly increases the flexibility for the reconfiguration.

TABLE I
MESSAGE INFORMATION FOR THE DAGS IN FIGURE 11.

Msg	(Send, Recv)	Byte	Msg	(Send, Recv)	Byte
m_1	(τ_1, τ_3),(τ_1, τ_4)	12	m_{10}	(τ_{12}, τ_{14})	10
m_2	(τ_2, τ_4)	12	m_{11}	(τ_{15}, τ_{20})	12
m_3	(τ_3, τ_5)	20	m_{12}	(τ_{16}, τ_{20})	12
m_4	(τ_4, τ_6)	12	m_{13}	(τ_{17}, τ_{20})	12
m_5	(τ_7, τ_{10})	12	m_{14}	(τ_{18}, τ_{20})	12
m_6	(τ_8, τ_{10})	12	m_{15}	(τ_{19}, τ_{22})	10
m_7	(τ_9, τ_{11})	10	m_{16}	(τ_{20}, τ_{22})	22
m_8	(τ_{10}, τ_{11}),(τ_{10}, τ_{12})	12	m_{17}	(τ_{21}, τ_{22})	20
m_9	(τ_{11}, τ_{13})	10	m_{18}	(τ_{22}, τ_{23}),(τ_{22}, τ_{24})	6

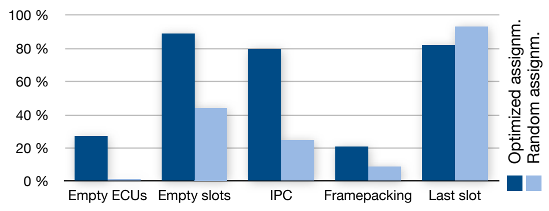


Fig. 12. Parameter rates for optimized assignments.

To evaluate the quality of the reconfigurations, we simulated ECU failures using the above described use-case from [9] and calculated reconfigurations based on arbitrary initial configurations. Therefore, we considered several network sizes (number of ECUs). Figure 13 illustrates the rate of determined optimal and valid reconfigurations (see Section IV-D) for different numbers of ECUs. The rate is increasing with a growing network size as expected, because of the growing flexibility and capacities for the task-to-ECU assignment. 40% to 60% of individual ECU failures can be reconfigured optimal without any additional slots depending on the number of ECUs. The rate of valid reconfigurations with a minimal addition of 2 slots is up to 80%.

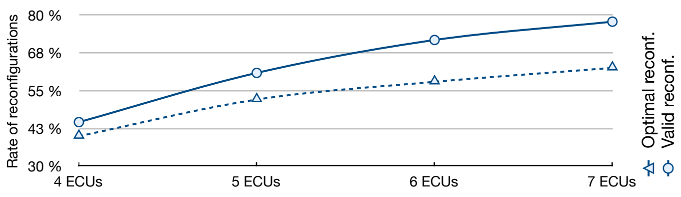


Fig. 13. Rate of reconfigurations depending on number of ECUs.

VI. CONCLUSION

We presented a reconfiguration approach for FlexRay networks, which supports the developer to increase the fault tolerance of the system. Our approach is a heuristic, which determines initial configurations and based on them calculates reconfigurations for the remaining ECUs in case of a node failure. The evaluation showed that our approach determines valid reconfigurations for the majority of possible individual node failures. In summary, it provides helpful feedback about reconfiguration capabilities to determine and optimize valid reconfigurations and reduce their effort and overhead regarding additional redundant slots.

ACKNOWLEDGEMENTS

This work was partly funded by the DFG Collaborative Research Centre 614 and by the German Ministry of Education and Research (BMBF) through the ITEA2 project VERDE (01S09012H).

REFERENCES

[1] K. Klobedanz *et al.*, "Distributed Coordination of Task Migration for Fault-Tolerant FlexRay Networks," in *Proc. of SIES 2010*, 2010.
[2] K.Klobedanz *et al.*, "Task Migration for Fault-Tolerant FlexRay Networks," in *Proc. of DIPES 2010*, 2010.
[3] FlexRayConsortium, "FlexRay Communications System Protocol Specification Version 2.1 Rev. A," Dec 2005, www.flexray.com.
[4] S. Ding *et al.*, "An Effective GA-Based Scheduling Algorithm for FlexRay Systems," *IEICE - Trans. Inf. Syst.*, 2008.
[5] M. Grenier *et al.*, "Configuring the Communication on FlexRay: the case of the static segment," in *Proc. of ERTS'08*, 2008.
[6] M. Lukasiewycz *et al.*, "FlexRay schedule optimization of the static segment," in *Proc. of CODES+ISSS '09*, 2009.
[7] R. Brendle *et al.*, "Dynamic Reconfiguration of FlexRay Schedules for Response Time Reduction in Asynchronous Fault-Tolerant Networks," in *Proc. of ARCS*, 2008.
[8] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 1997.
[9] N. Kandasamy *et al.*, "Dependable Communication Synthesis for Distributed Embedded Systems," in *Proc. of SAFECOMP*, 2003.