Task Migration for Fault-Tolerant FlexRay Networks

Kay Klobedanz, Gilles B. Defo, Henning Zabel, Wolfgang Mueller, and Yuan Zhi

University of Paderborn/C-LAB

Abstract. In this paper we present new concepts to resolve ECU (Electronic Control Unit) failures in FlexRay networks. Our approach extends the FlexRay bus schedule by redundant slots with modifications in the communication and slot assignment. We introduce additional backup nodes to replace faulty nodes. To reduce the required memory resources of the backup nodes, we distribute redundant tasks over different nodes and propose the migration of tasks to the backup node at runtime. We investigate different solutions to migrate the redundant tasks to the backup node by time-triggered and event-triggered transmissions.

1 Introduction

FlexRay is the emerging standard for safety-critical distributed real-time systems in vehicles [5][15]. It implements deterministic behavior and comes with high bandwidths. For increased safety, it provides redundant channels to guarantee communication if one channel is corrupted. Nevertheless, since an ECU (Electronic Control Unit) failure still often results in the malfunction of the whole system, the main question remains how to ensure the correct behavior of a safety-critical distributed system in such a case. As presented in [4], different techniques for tolerating *permanent*, *transient*, or *intermittent* faults are applied. In our article, we consider ECU failures based on permanent hardware faults which are compensated by means of redundancy. We focus on the replication of tasks and the activation of backup nodes. The failure results in an execution of the redundant tasks on a different node which induces changes in the communication at runtime. Unfortunately, FlexRay only supports static bus schedules where each slot is reserved for an individual sender and the slot assignments can only be changed by a bus restart, whose timing is not exactly predictable. In contrast, our approach extends bus schedules by redundant slots and considers communication dependencies already at the system design phase before network configuration. Additionally, as ECU functionalities are distributed over several ECUs, the failure of an ECU which executes several functions may have a big impact on the correct operation of the whole system. We adopt this approach and assign two functions to an ECU.[1] One implements the main ECU function; the other one is a redundant mirrored instance of another ECU function, which is activated on a failure of the other ECU (see Fig. 1 and [4]). Here, just the failure of one ECU can be compensated. A second node failure may lead to the failure of the complete system again. In Figure 1, for example, the failure of node n1 disables its own task set and the backup for node n2. The additional failure of node n4 irrecoverable corrupts the functionality of node n1. We introduce backup nodes to

[1] For the matter of simplicity, we presume that a function corresponds to a task.

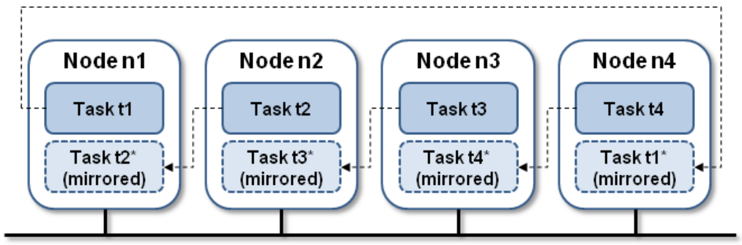

Fig. 1. Redundancy with mirrored tasks on other nodes.

completely replace any faulty node and presume a homogeneous network of nodes with identical resources. This redundancy raises the fault tolerance of distributed networks and we can furher improve it by simply increasing the number of backup nodes. This approach additionally requires the migration of the main and the redundant function of the faulty ECU from other nodes and their migration to the backup node. The additional advantage is that the redundant function can immediately start executing before the migration, e.g., t1* on node n4. After the migration, the execution is resumed by the backup node. This yields to the initial setup where every node is executing only its main functionality again and keeps a redundant instance of a function from another ECU which compensates an additional failure of an arbitrary node.

In this article, we present different variants for such a task migration and evaluate them with respect to their time consumption, predictability, and impact to the communication. A detailed description of our approach is given in Section 4. The evaluation is presented in Section 5 before the final section concludes with a summary.

2 Related Work

Several approaches like [14] deal with the analysis of the FlexRay protocol and its optimization. They present several heuristics to determine proper configurations and parameterizations for the FlexRay bus based on the static [11][7][10] and the dynamic segment [14][13]. In general, their optimizations and the resulting configurations assume that the executed tasks are statically linked to the nodes of the FlexRay network. [4] considers a replication of tasks and a more flexible task to node assignment. Based on these assumptions they determine the reconfiguration capabilities of the FlexRay bus.

Task migration itself is a hot topic in current automotive research. For example, [1] describes a concept for a middleware, which uses task migration further described in [9] to increase the reliability of distributed embedded systems with soft real-time constraints, e.g., for infotainment systems. In contrast to our work, they do not consider safety-critical components and the runtime reconfiguration of FlexRay networks.

Task migration at runtime was considered in the context of mobile agents like [12] and [2]. However, we are not interested in principle architectures rather than on their technical realization and the efficient task migration in the context of FlexRay. We are not aware of other related approaches in this area.

3 FlexRay

FlexRay was introduced to implement deterministic and fault-tolerant communication systems for safety-critical distributed real-time systems like x-by-wire. The main benefits of FlexRay are:

- **Synchronous and asynchronous data transmission**: FlexRay offers cycle-based time-triggered communication complemented by an optional event-triggered transmission mode.
- **Determinism**: The time triggered transmission mode of FlexRay ensures real-time capabilities of the communication because it guarantees deterministic behavior with a defined maximum message latency.
- **Redundant communication channels with large bandwidth**: FlexRay offers two redundant channels for safety-critical systems. Each channel offers a bandwidth up to 10 $Mbit/s$ with little latency.

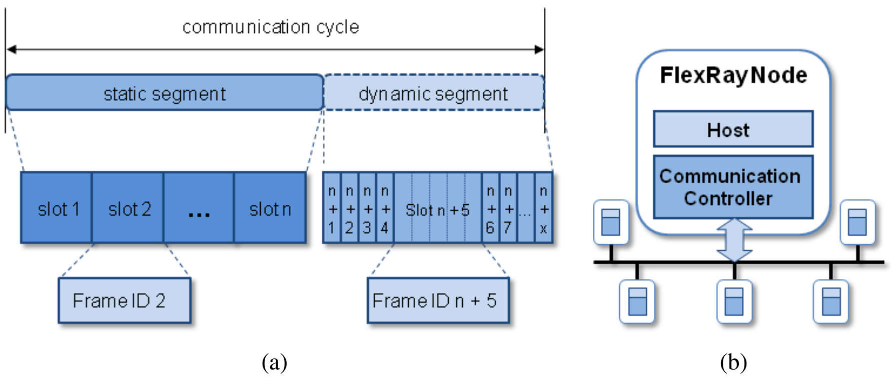

Fig. 2. Components of the FlexRay communication cycle (a) and communication controller (b).

A communication cycle can be composed of a static and an optional dynamic segment (see Figure 2(a)). In the static segment, the time-triggered data transfer is carried out via TDMA (Time Division Multiple Access). The transmission slots of the segment are assigned to one sender node by a globally known synchronously incremented slot-counter. The static segment consists of a fixed number of equally sized static slots (2 – 1023). The event-triggered dynamic segment realizes the bus access via FTDMA (Flexible Time Division Multiple Access) and consists of dynamic slots with variable size. Dynamic slots are composed of minislots whose number depends on the length of the message to transmit (max. 7986 per cycle). The arbitration is accomplished by a priority assignment to nodes (Frame IDs). If a node has nothing to send, only one minislot is unused and the next node gets the opportunity for transmission. The size of the slots and minislots, the cycle length, the size of messages (frames) as well as several other parameters are defined through an initial setup of the FlexRay schedule, which cannot be changed during runtime. Figure 2(b) shows the basic components of a FlexRay node. A host for the functionality of the ECU and a communication controller (CC). The CC is the core component of FlexRay because it implements the actual protocol. It provides the communication interface to the bus through a bus driver and it performs the sending/receiving and the decoding/coding of data messages (frames). For more detailed introduction to FlexRay, the reader is referred to [6].

4 Migration of Redundant Tasks in FlexRay Networks

Our concepts are based on existing approaches for redundant tasks to improve the robustness of FlexRay networks in case of a node failure. To further increase the fault tolerance of a FlexRay network, our concept extends them by the improvements outlined in the next subsection.

4.1 Overview

Our improvements for an increasing fault tolerance are:

- **Extension with additional backup node(s):** Backup nodes can completely replace any faulty node as they provide the same resources as the other nodes. To reduce the memory demand for the backup node, we migrate necessary tasks rather than storing instances of all tasks on a backup node. This results in a homogenous system topology with similar node capacity.
- **Migration to the backup node at runtime:** During the migration, the functionality is immediately executed by the redundant instance on another ECU. After the migration, the corresponding task execution is resumed on the backup node. This yields to the initial system setup where every node executes a main function and keeps an additional redundant instance from another ECU.

Additionally, we add a coordinator component, which is in charge of the communication in the system. This can be realized by two different strategies. First, the coordinator can be distributed on different nodes. Second, the coordinator can – as it is realized here – run on an extra node to keep the topology described above with ECUs executing one main function, represented as one task. Figure 3 presents the topology of a FlexRay network with an extra backup and coordinator node.

As shown in Figure 3, the coordinator maintains a task list, with main and redundant tasks of every node and a communication matrix (see Table 1) with the corresponding transmission/reception slots. These structures will be updated when modifications in task and slot assignments are required due to node failures, execution of redundant tasks, or task migrations. The coordinator is configured to monitor messages from all

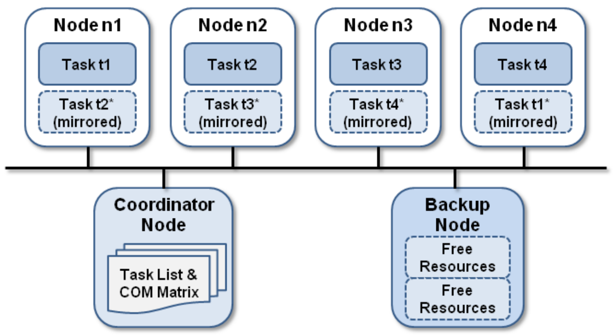

Fig. 3. Topology with backup and coordinator node before a failure of Node n1.

slots. Using the information from the task list and the communication matrix, the coordinator detects node failures monitoring the bus traffic. If a malfunction of an ECU is detected, the coordinator sends a message to the appropriate nodes to activate the corresponding redundant task instance and to start the transmission of the particular tasks to the backup node. This message also contains information about changes in the slot assignment for receiving nodes. When the transmission is complete, this is recognized by the coordinator. Then, the coordinator activates the migrated task on the backup node at the same time it deactivates the redundant task and informs the receiving nodes about rearrangements in the slot assignments. Figure 4 illustrates this process for the system shown in Figure 3 in case of a failure of node n1. The synchronous and asynchronous

	static segment					dynamic segment		
Slot	s1	s2	s3	s4	...	d1	d2	...
Node n1	t1:tx	-	t1:rx	-	...	-	-	...
Node n2	t2:rx	t2:tx	t2:rx	-	...	-	-	...
Node n3	t3:rx	-	t3:tx	t3:rx	...	-	-	...
Node n4	-	-	-	t4:tx	...	-	-	...

Table 1. Example of a communication matrix for a topology with 4 nodes.

data transmission of the FlexRay protocol allows different implementations of our approach. It can be realized in the static segment, in the dynamic segment, or in both. In the following we introduce alternative scenarios for the previous example.

4.2 Exclusive Usage of Static Segment

For an exclusive usage of the static segment, additional static slots for the coordinator, the backup node, and the migrations have to be reserved. This extends the static seg-

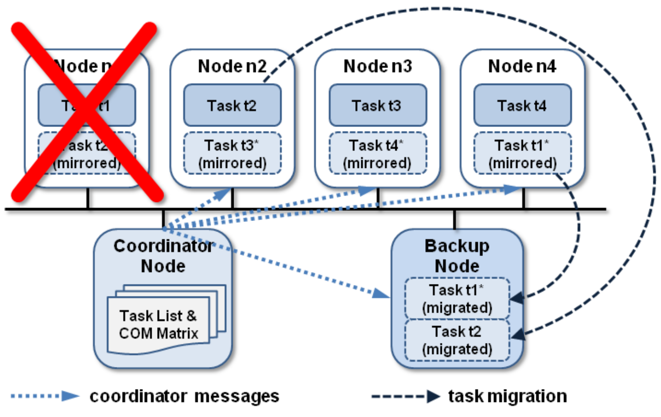

Fig. 4. Topology with backup and coordinator node after a failure of Node n1.

ment as shown in Table 2. The advantage of this solution is the predictability for the transmissions of all nodes as well as the task migration. The duration $\Delta_{migration}$ of the copy process for a task with the size Φ_{task} can be computed by

$$\Delta_{migration} = \#_{cycles} \cdot \Delta_{cycle} \;\; with \;\; \#_{cycles} = \left\lceil \frac{\Phi_{task} \cdot gdBit}{\Delta_{slot}} \right\rceil, \tag{1}$$

where $\#_{cycles}$ is the number of cycles and Δ_{cycle} the length of a cycle. The cycle length and the nominal bit time (inverted bit rate) *gdBit* are configured in the FlexRay schedule as well as the slot length Δ_{slot}, which is related to the number of needed cycles for a task migration ($\#_{cycles}$) (ref. (1)). Table 2 presents the static slots and the resulting assignments and changes for a failure of node n1. In addition to slots s1,..,s4 for the primarily communication between the four nodes, slots for the coordinator (s5) and the transmission of the redundant task during the migration process (s6,...,s9) – here t1* transmits in s9 – have to be reserved in the schedule. For the migration, a slot for each node (s10, ..., s13) and a slot for the backup node (s14) has to be reserved. The communication matrix shows that all nodes receive the messages from the coordinator (s5) and the coordinator monitors all slots. Node n4 uses slot s13 to copy the task t1* to the backup node (t1*:mig) and node n2 uses s11 (t2:mig). During that period, the redundant task t1* transmits its data in slot s9 and receives from slot s3. The corresponding tasks/nodes are informed via the coordinator message to listen to s5. When the migration is finished, the backup node receives the advice (bk:rx) to activate its migrated instance of t1*. Thereafter, task t1* on the backup node reads from slot s3 and sends via slot s14 whereas the receiving tasks are informed to listen to this slot. Simultaneously, node n4 stops executing its instance of task t1*. All transmissions as well as the migration are time-triggered and have a guaranteed maximum latency. But only few of the reserved slots are required in the case of a node failure. This results in a big overhead of slots. The setting of an appropriate static slot size (Δ_{slot}) makes the situation even worse. To get a small number of cycles needed for a migration ($\#_{cycles}$), slots should be considerably large. Beside the fact that FlexRay as well as system properties (e.g., sampling rates of tasks) limit the maximal slot length, this causes a significant increase of the overhead because every slot needs to be equally sized independent of its transmitted content.

4.3 Exclusive Usage of Dynamic Segment

To minimize the overhead described above, it is possible to exclusively use the event-triggered dynamic segment. To guarantee the correct transmission of data within the dynamic segment, FlexRay only allows a node to send if its frame completely fits into the remaining minislots (ref [6]). In the case that it does not fit, the node has to wait for the next cycle. This makes the communication nondeterministic, particularly if the dynamic segment is also used for other event-based messages like error codes etc. Thus, the calculation of the time needed for the migration within the dynamic segment is more complex. The value for ($\Delta_{migration}$) can be determined using equation (1). The slot size (Δ_{slot}) is dynamic and is determined by

$$\Delta_{slot} = \Delta_{dynamic} - \Delta_{hp} \;\; with \;\; \Delta_{hp} = \sum_{k=1}^{i} \Delta_{frame_i}. \tag{2}$$

Equation (2) shows that the available dynamic slot size for the migration is given by the complete length of the dynamic segment ($\Delta_{dynamic}$) decremented by the slot sizes used by frames with higher priority (Δ_{hp}). These results inserted into Equation (1) yield to the migration time derived from the segment length and the priority assignment. The communication matrix in Table 3 shows the assignment. The coordinator gets the highest priority (d1) because it transmits messages of highest priority. During the migration process task t1* running on node n4 uses d2 to send its data. The other reserved slots (d3-d5) are unused. When the migration is finished, the activated backup node blocks the dynamic slot d6 and d2 remains unused. The communication matrix also shows that the migration of the tasks gets the lowest priorities (d7,..., d10). Here, d8 is used for the migration of t2 and d10 for the migration of t1*. To permit access to the dynamic segment, the data size of the migration process must fit in the space left influenced by prior messages. Additionally, the coordinator informs the backup node how much data to transmit per cycle. This usage of the dynamic segment makes this solution more flexible than the use of the static segment. Even though the transmission is event-triggered and nondeterministic, it can be guaranteed that sufficient data are transmitted due to a proper priority assignment. The size of the dynamic segment has to be initially configured based on the system properties and the message sizes to reach the desired migration process duration $\Delta_{migration}$ along Equation (2). Because the utilization in the dynamic segment is more flexible and the static slot size is independent of the migration data, the potential overhead of this solution is considerably less than in the static version of Table 2. In particular, each unused slot in the dynamic segment only consumes one minislot. Nevertheless, the major drawback is the partial loss of determinism, which is an important requirement for safety-critical systems.

4.4 Usage of Static and Dynamic Segments

A compromise between the previous two solutions is given by the configuration given in Table 4. Here, we achieve a reduction of overhead in compliance with a deterministic communication. The communication matrix shows that the coordinator node (s5), the redundant task instances (s6,..., s9), and the backup node (s10) communicate time-triggered over reserved static slots like in the exclusive static segment solution. In contrast, the migration itself is performed via the dynamic segment (d1,...,d4), which reduces the overhead significantly as the size of the migration frames only influences the size of the dynamic segment and unused slots in the dynamic segment generate less overhead. Hence, the size of the static slots remains independent of the migration data. In summary, this yields to a higher flexibility in the schedule configuration and combines the benefits. On the one hand, we guarantee a maximum latency and a deterministic transmission for the important messages using the static segment. On the other hand, we reduce the overhead by the exclusive assignment of the dynamic segment to the migration process. Through this, the migration time is even more predictable because the capacity required by prioritized tasks is omitted.

5 Experimental Results

We evaluated the presented alternatives by simulations with our SystemC Flex-Ray library. SystemC is a system design language providing means to model application-specific hardware and software at different levels of abstraction [3]. The implemented FlexRay CC supports the simulation of static and dynamic segments for one communication channel. All necessary modules specified in the FlexRay standard (see [6]) are also covered by the implementation. Our model consists of six modules implementing communication nodes along the topology shown in Figure 3. Each node uses an instance of the CC for the communication. All CCs are in turn connected to a transaction level (TLM) bus object. The CC model can be configured with the same controller host interface (CHI) files like hardware CCs. This file contains all necessary parameters to configure the registers and message buffers for the communication. The bus communication applies TLM 2.0 [8]. Figure 5 depicts a communication sequence between two communication controllers. The communication controllers act as initiators of the transaction during the communication process and the bus acts as the target. We use approximately timed TLM 2.0 coding style to model the communication with the bus module. The communication is divided into four phases: Begin/end request and begin/end response. As shown in Figure 5, CONTROLLER1 starts a write transaction, with a begin request. Immediately after the receipt of the transaction, the bus module notifies CONTROLLER2 about the start of the data transmission with the time required for the data transfer. Afterwards, CONTROLLER2 starts a read transaction. Both controllers then wait until they have received a confirmation from the bus about the end of the data transfer. The SystemC model consists of 6 nodes, 6 CCs, and 1 bus module in total. Nodes communicate via their respective CCs. Communication between nodes and CCs is realized via callback methods. Each node has to implement receive and transmit function, that are called by the FlexRay CC. Since we are using an abstract model for the evaluation, we did not actually implement the migration process itself. Instead we simulate the reconfiguration/migration duration since timing analysis is our main focus.

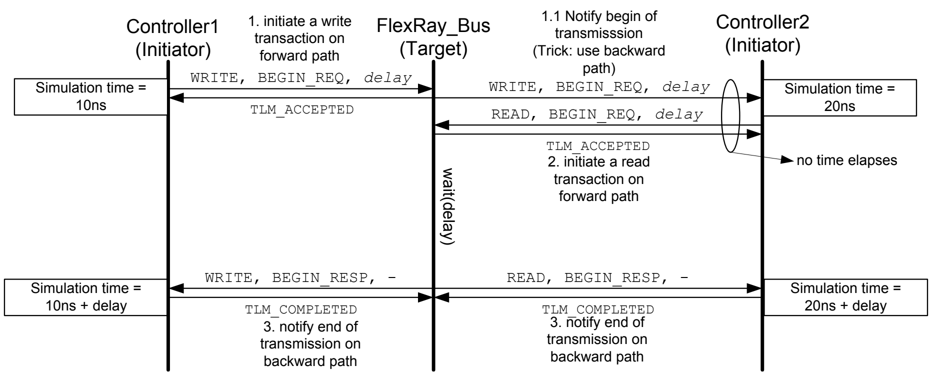
Controller1 (Initiator)
1. initiate a write transaction on forward path
FlexRay_Bus (Target)
1.1 Notify begin of transmisssion (Trick: use backward path)
Controller2 (Initiator)
Simulation time = 10ns
WRITE, BEGIN_REQ, delay
TLM_ACCEPTED
WRITE, BEGIN_REQ, delay
READ, BEGIN_REQ, delay
Simulation time = 20ns
TLM_ACCEPTED
2. initiate a read transaction on forward path
no time elapses
wait(delay)
Simulation time = 10ns + delay
WRITE, BEGIN_RESP, -
TLM_COMPLETED
3. notify end of transmission on backward path
READ, BEGIN_RESP, -
TLM_COMPLETED
3. notify end of transmission on backward path
Simulation time = 20ns + delay

Fig. 5. Simulation of bus transmission.

Simulation Results In the following, we present the simulation results for one example. For that we assume a size of 1 $kByte$ = 8000 bit for a task to migrate (Φ_{task}) and configure the FlexRay_Bus schedule by:

– $\Delta_{cycle} = 600\mu s$ for static solution,
– $\Delta_{cycle} = 600\mu s + 400\mu s = 1000\mu s$ for dynamic and mixed solution,
– $\Delta_{slot} = 50\mu s$ for the static segment with 14 slots,
– $gdBit = 0,1\mu s/bit$ (equates to a bandwith of 10 $Mbit/s$).

The time needed for the migration within the static segment can be determined by means of Equation (1) as:

$$\#_{cycles} = \left\lceil \frac{\Phi_{task} \cdot gdBit}{\Delta_{slot}} \right\rceil = \left\lceil \frac{8000\,bit \cdot 0,1\,\mu s/bit}{50\mu s} \right\rceil = 16,$$

$$\Delta_{migration} = \#_{cycles} \cdot \Delta_{cycle} = 16 \cdot 600\mu s = 9,6ms$$

For the (exclusive) assignment of the dynamic segment to the migration and the usage of both transmission modes, the simulation also confirms the computed values, e.g.:

$$\#_{cycles} = \left\lceil \frac{\Phi_{task} \cdot gdBit}{\Delta_{slot}} \right\rceil = \left\lceil \frac{8000\,bit \cdot 0,1\,\mu s/bit}{400\mu s} \right\rceil = 2 \;\; with \;\; \sum_{k=1}^{i} \Delta_{frame_i} = 0,$$

$$\Delta_{migration} = \#_{cycles} \cdot \Delta_{cycle} = 2 \cdot 1000\mu s = 2ms.$$

All numbers were additionally validated by our simulations which thus confirms the applicability of our approach.

6 Conclusion & Outlook

This paper presented different alternatives of redundant tasks and slots for the compensation of node failures in safety-critical FlexRay networks. We introduced backup nodes, which can replace any faulty node when our task migration is applied. With this scalable approach, we further increase redundancy and fault tolerance through the compensation of an additional failure of an arbitrary node. We presented three different task migration strategies based on the transmission capabilities of FlexRay and evaluated them by a SystemC simulation. The comparison of the proposed solutions showed that the combined usage of static and dynamic segment improves the benefits and minimizes the disadvantages by providing deterministic communication with low overhead and flexibility for task migration. On the one hand, it results in a maximum latency for the transmission of functional relevant messages in the static segment. On the other hand, it reduces the overhead through the exclusive assignment of the dynamic segment to the migration process.

In future work, we will examine the distribution and redundancy of the coordinator component within the system. By this, we avoid the potential "single-point of failure" induced by the solution with a single coordinator ECU. The additional small memory requirement resulting from the stored task list and communication matrix on several nodes is neglectable.

7 Acknowledgements

The work described herein is supported by the BMBF through the ITEA2 VERDE project (01S09012H).

References

1. R. Anthony, D. Chen, M. Törngren, D. Scholle, M. Sanfridson, A. Rettberg, T. Naseer, M. Persson, and L. Feng. Autonomic middleware for automotive embedded systems. In *Autonomic Communication*, 2009.
2. Lubomir F. Bic, Munehiro Fukuda, and Michael B. Dillencourt. Distributed computing using autonomous objects. *Computer*, 29(8), 1996.
3. Donovan Jack Black David C. *SystemC: From the Ground Up*. KLUWER ACADEMIC PUBLISHERS NEW YORK, BOSTON, DORDRECHT, LONDON, MOSCOW, 2004.
4. Robert Brendle, Thilo Streichert, Dirk Koch, Christian Haubelt, and Jürgen Teich. Dynamic reconfiguration of flexray schedules for response time reduction in asynchronous fault-tolerant networks. In *ARCS*, 2008.
5. Julian Broy and Klaus D. Müller-Glaser. The impact of time-triggered communication in automotive embedded systems. In *SIES*, 2007.
6. FlexRay Consortium. Flexray communications system protocol specification version 2.1 rev. A, Dec 2005. www.flexray.com.
7. Shan Ding, Hiroyuki Tomiyama, and Hiroaki Takada. An effective ga-based scheduling algorithm for flexray systems. *IEICE - Trans. Inf. Syst.*, E91-D(8), 2008.
8. Open System Initiative. Osci tlm2 user manual, software version tlm 2.0 draft 2, dcument version 1.0.0, 2007.

9. F. Kluge, J. Mische, S. Uhrig, and Th. Ungerer. Building Adaptive Embeddd Systems by Monitoring and Dynamic Loading of Application Module. In L. Almeida et al., editor, *Workshop on Adaptive and Reconfigurable Embedded Systems*, St. Louis, MO, USA, April 2008.
10. Martin Lukasiewycz, Michael Glaß, Jürgen Teich, and Paul Milbredt. Flexray schedule optimization of the static segment. In *CODES+ISSS '09*, New York, NY, USA, 2009. ACM.
11. L. Havet M. Grenier and N. Navet. Configuring the communication on flexray: the case of the static segment. In *ERTS'08*, 2008.
12. H. Peine and T. Stolpmann. The architecture of the ara platform for mobile agents. In *Proc. of the First International Workshop on Mobile Agents*. Springer-Verlag, 1997.
13. T. Pop, P. Pop, P. Eles, and Z. Peng. Bus access optimisation for flexray-based distributed embedded systems. In *DATE'07*, 2007.
14. Traian Pop, Paul Pop, Petru Eles, Zebo Peng, and Alexandru Andrei. Timing analysis of the flexray communication protocol. *Real-Time Syst.*, 39(1-3), 2008.
15. A. Schedl. Goals and architecture of flexray at bmw. slides presented at the Vector FlexRay Symposium, Mar 2007.

	static segment													
Slot	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	s12	s13	s14
~~Node n1~~	~~t1:tx~~	-	~~t1:rx~~	-	**~~t1:rx~~**	-	-	-	-	-	-	-	-	-
Node n2	t2:rx	t2:tx	t2:rx	-	**t2:rx**	-	-	-	**t2:rx**	-	**t2:mig**	-	-	***t2:rx***
Node n3	t3:rx	-	t3:tx	t3:rx	**t3:rx**	-	-	-	**t3:rx**	-	-	-	-	***t3:rx***
Node n4	-	-	**t1*:rx**	t4:tx	**t4:rx**	-	-	-	**t1*:tx**	-	-	-	**t1*:mig**	-
Coordinator	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:tx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**
Backup	-	-	***t1*:rx***	-	**bk:rx**	-	-	-	-	-	**mig:rx**	-	**mig:rx**	***t1*:tx***

Table 2. Communication matrix for exclusive usage of the static segment.

	static segment				dynamic segment									
Slot	s1	s2	s3	s4	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10
~~Node n1~~	~~t1:tx~~	-	~~t1:rx~~	-	**~~t1:rx~~**	-	-	-	-	-	-	-	-	-
Node n2	t2:rx	t2:tx	t2:rx	-	**t2:rx**	-	-	-	**t2:rx**	***t2:rx***	-	**t2:mig**	-	-
Node n3	t3:rx	-	t3:tx	t3:rx	**t3:rx**	-	-	-	**t3:rx**	***t3:rx***	-	-	-	-
Node n4	-	-	**t1*:rx**	t4:tx	**t4:rx**	-	-	-	**t1*:tx**	-	-	-	-	**t1*:mig**
Coordinator	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:tx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**
Backup	-	-	***t1*:rx***	-	**bk:rx**	-	-	-	-	***t1*:tx***	-	**mig:rx**	-	**mig:rx**

Table 3. Communication matrix for exclusive usage of dynamic segment.

	static segment										dynamic segment			
Slot	s1	s2	s3	s4	s5	s6	s7	s8	s9	s10	d1	d2	d3	d4
~~Node n1~~	~~t1:tx~~	-	~~t1:rx~~	-	**~~t1:rx~~**	-	-	-	-	-	-	-	-	-
Node n2	t2:rx	t2:tx	t2:rx	-	**t2:rx**	-	-	-	**t2:rx**	***t2:rx***	-	**t2:mig**	-	-
Node n3	t3:rx	-	t3:tx	t3:rx	**t3:rx**	-		-	**t3:rx**	***t3:rx***	-	-	-	-
Node n4	-	-	**t1*:rx**	t4:tx	**t4:rx**	-	-	-	**t1*:tx**	-	-	-	-	**t1*:mig**
Coordinator	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:tx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**	**co:rx**
Backup	-	-	***t1*:rx***	-	**bk:rx**	-	-	-	-	***t1*:tx***	-	**mig:rx**	-	**mig:rx**

Table 4. Communication matrix for ”mixed” usage of static segment and dynamic segment.