The Influence of Real-time Constraints on the Design of FlexRay-based Systems

Stephan Reichelt
Audi Electronics Venture GmbH

Oliver Scheickl
BMW Car IT GmbH

Gökhan Tabanoglu
Volkswagen AG

Abstract

This article describes important challenges regarding the design, specification and implementation of FlexRay-based automotive networks. The authors outline a design approach that especially accounts for timing constraints of the network, namely end-to-end and cycle timing constraints. The schedule generation for electronic control units (ECU) as well as bus entities is addressed and constraint compatibility with basic FlexRay configuration properties is investigated. The discussed design approach considers three practical design challenges of the automotive industry: first, the function-based cycle timing constraints and their dependency to basic bus design is presented. Second, the challenge of distributed development of modern on-board networks by many different teams and an approach for collaboration improvement is discussed. Finally, the third part describes the configuration of time-triggered ECU schedules with respect to different constraint types.

1 Introduction

Automotive electrical and electronic functions are often subject to physical constraints regarding their temporal behavior. These physical constraints are valid for any technical implementation. For example, the time interval from the reading of a sensor value until a reaction occurs at an actuator typically has a so called end-to-end timing constraint (e.g. 10 milliseconds maximum). Synchronizing the activation of four damper actuators is another example for the need of reliable temporal behavior. An implementation of functions through communicating software components, which are deployed on ECUs, must fulfill all constraints.

One solution to tackle hard real-time constraints and the corresponding design complexity is the introduction of the deterministic FlexRay bus system [8] [4]. However, this communication technology implies some new challenges to be solved within different phases of system development due to its two-step development approach [6]. On the one hand, basic bus configuration parameters like the number and size of static slots [4] and the length of a communication cycle need to be defined. On the other hand, due to the high dependency between bus and ECU schedules in case of synchronized time bases, both need to be developed concurrently at a pre-runtime phase.

As an additional challenge induced by the characteristics of the automotive industry, many distributed development teams need to exchange information. In particular, ECU and bus timing-related information needs to be exchanged - even across the borders of companies. Common development methodologies lack the capability of transferring timing-related information. On system level, local timing constraints (i.e. budgets) referred to a particular ECU or software component (SW-C) have to be derived from mutual dependent end-to-end timing constraints (typically sensor-actuator or synchronicity constraints). On ECU level, these budgets as well as several other non-functional constraints (like protection or functional precedence) have to be considered during local scheduling.

In this publication a set of cycle and end-to-end timing constraints for the communication of software components is considered. The authors present a comprehensive design approach to a) check the compatibility of cycle timing constraints and basic timing properties of a given FlexRay configuration, b) develop a bus schedule as well as ECU schedule budgets according to cycle and end-to-end timing constraints and c) configure the ECU schedules according to other non-functional constraints, e.g. safety-related constraints. In addition, so-called *Timing Interfaces* for a precise exchange of timing constraints between development teams are discussed.

2 Proposed Design Flow

The proposed design flow (see Fig. 1) is aligned to the AUTOSAR methodology [1]. AUTOSAR gains increasing importance in the automotive domain and real-time systems development [9].

The input information, which is the software architecture and the ECU topology, is considered to be given. The mapping of software components to ECUs is one major step in the AUTOSAR methodology. The mapping is also considered to be given. Besides this, the following input is required for the proposed design flow:

- The software architecture represents a decomposition of functionality by means of software components.

For their interaction end-to-end timing constraints are specified.

- Mapped SW-Cs imply certain constraints on the sending interval of signals throughout the network, called *cycle timing constraint*.
- A defined ECU topology implies given FlexRay configuration properties. For example, the bus bandwidth and the FlexRay cycle [4] are fixed (not the schedule!).

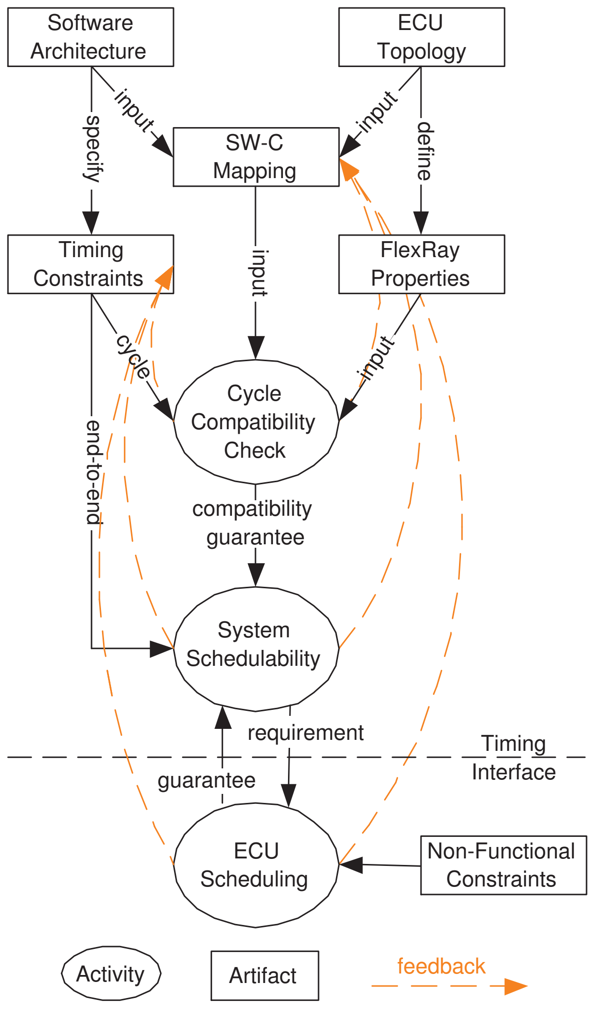

Figure 1. Overall Design Flow

The cycle timing constraints are processed by the *Cycle Compatibility Check* as depicted in Fig. 1. In this activity, cycle timing constraints are analyzed with regard to their compatibility to the timing properties of a given FlexRay configuration.

The next step, analyzing the *System Schedulability*, relies on the compatibility guarantee and the end-to-end constraints given by the functions to be integrated. This activity decomposes the function-based end-to-end constraints into chain segment constraints to be fulfilled by the corresponding ECUs. A methodology is introduced by defining a timing interface concept between the scheduling activities of system and ECU level. This allows a fast and efficient application of global bus scheduling strategies.

The last step of the proposed approach, outlined in Sec. 5, describes a local *ECU Scheduling* strategy. This task consists of many constraints to fulfill and has a direct dependency to the *System Schedulability* step. Timing constraints to be fulfilled by an ECU are transferred using the timing interface described in Sec. 4. In addition to this, other non-functional constraints have to be regarded for the development of local execution schedules. By reducing these constraints to a weighted attribute for the task set to be scheduled, a constructive approach is introduced for the development of local execution schedules. This is a typical optimization problem with additional parameters to be regarded. The proposed design flow results in a handover of timing constraints from the local *ECU Scheduling* step to the *System Schedulability* step. In this case, the constraints have the semantic representation of a guarantee.

Each step has implications on the decisions and inputs retrieved from prior steps in the proposed design flow. These implications are indicated as feedback arrows in Fig. 1. It is part of our future work to develop strategies how this feedback can be dealt with. We consider concepts, how either incompatibilities or non-schedulability of timing constraints are tackled. End-to-end constraints are mostly given by physics and thus are fixed. Hence, we basically see two main options. Either the cycle constraints for communication can be adapted or the software mapping can be reconfigured. The core of our future work will be to derive appropriate proposals for adjustment of these input parameters.

3 Cycle Compatibility Check

The fulfillment of cycle timing constraints by a FlexRay network is basically limited by the timing properties of the given FlexRay configuration. This mandatory limitation yields a division of given cycle timing constraints into two subsets. The first subset consists of so-called *configuration-compatible* cycle timing constraints. These are constraints, which can basically be fulfilled by the given FlexRay configuration without any adjustment of a FlexRay property. The second subset consists of so-called *configuration-incompatible* timing constraints. These are constraints, which can not be fulfilled without any adjustment even if all static slots [4] are available. To be able to detect configuration-incompatible timing constraints in an early phase of the development process, a method for checking the compatibility of signals timing constraints regarding a given FlexRay configuration was developed and is presented in this section. In case of detecting configuration-incompatible timing constraints different adjustment strategies such as changing timing properties of a given FlexRay configuration, changing the function-based timing constraints or reconfiguring the software mapping may be applied to establish the compatibility. Although these adjustment strategies are not in the scope of this paper, it should be noted that any adjustment strategy is a trade-off between technical feasibility of functions and development effort.

The proposed *Cycle Compatibility Check* is applied to signal's cycle timing constraints. It is not of vital importance whether a single constraint refers to one or a group of signals. In case of an incompatibility detection of a timing constraint each signal with this constraint is basically not schedulable by the given FlexRay configuration.

It should be noted that only periodic sending requests are considered by the *Cycle Compatibility Check*. Additionally the dynamic segment is not considered for meeting timing constraints due to its non-deterministic behavior. Dependencies between timing constraints are also not considered since this is a task for the *System Schedulability* analysis step (see Fig. 1).

The basis for the *Cycle Compatibility Check* is both the timing properties of a given FlexRay configuration and the signals cycle timing constraints. The timing properties of a FlexRay configuration are mainly given by the values of several configuration parameters [4]. For the *Cycle Compatibility Check*, as depicted in Fig. 2, primarily the following FlexRay properties are of interest [4]:

- length of communication cycle (T_C)
- length of dynamic segment (T_{DynS})
- length of network idle time (T_{NIT})
- length of symbol window (T_{SW})
- length of one static slot (T_{sSlot})
- number of static slots (N_{sSlot})

The signal cycle timing constraint, as described before, is the periodic sending request (T_{PSR}).

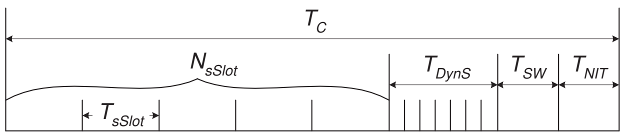

Figure 2. Relevant configuration parameters

An essential element of the *Cycle Compatibility Check* is the greatest common divisor (GCD) as defined in (1) and (2).

(1) $$p, q \in \mathbb{Q}$$

(2) $$\mathrm{GCD}(p,q) := max\,\{k \mid \exists\, n, m \in \mathbb{Z}\colon n \cdot k = p \wedge m \cdot k = q\}$$

The GCD of the cycle length and the periodic sending request, $\mathrm{GCD}(T_C, T_{PSR})$, represents the minimum time interval between two required bus accesses of the corresponding node. This time interval is an essential part of the *Cycle Compatibility Check* which is based on the following conditions.

(3) $$\mathrm{GCD}(T_C, T_{PSR}) > T_{DynS} + T_{NIT} + T_{SW}$$

(4) $$\frac{T_C}{\mathrm{GCD}(T_C, T_{PSR})} \leq N_{sSlot}$$

In case of $\mathrm{GCD}(T_C, T_{PSR}) \neq T_C$ additionally condition (5) must be fulfilled.

(5) $$\mathrm{GCD}(T_C, T_{PSR}) \bmod T_{sSlot} = 0$$

In condition (3) it is required that the minimum time interval between two required bus accesses must be greater than the total duration of the dynamic segment, the network idle time and the symbol window. This is justified due to the non-deterministic characteristics of these parts [4]. In (4) it is required that the number of required bus accesses per communication cycle is less or equal than the number of available static slots. By this condition it is also ensured that the minimum time interval between two required bus accesses is greater than the duration of a single static slot. In (5) it is required that a whole number of slots fits into the time interval. This is needed because sending in the middle of a slot is not possible.

If all cycle timing constraints pass the *Cycle Compatibility Check*, it can be guaranteed that all of them are compatible regarding a given FlexRay configuration. Note that this does not necessarily mean that an appropriate FlexRay schedule exists. This is justified due to the non-consideration of dependencies between timing constraints, which is part of future work. However, the determined compatibility is the basis for the *System Schedulability* analysis in the next step, see Fig. 1.

4. System Schedulability of Automotive Real-time Systems

4.1 Roles and their Activities in Distributed Development

The automotive industry is characterized by a distributed development strategy. That is, the overall vehicle electrical system is developed by many different teams. The vehicle electrical system consists of several domain clusters (chassis, power train, etc.). In this work we consider a cluster as a system, e.g. the FlexRay cluster of the chassis domain. For a description of our approach to improve distributed development of an automotive system we first identify three important roles.

One role is the *system integrator* who designs, and later on, integrates the system. The system integrator (typically a car manufacturer) specifies the system's ECU network (sensors, controllers, actuators) and the software components, which are mapped onto the ECUs (see Fig. 1). The ECUs are then developed by *ECU integrators* (typically first tier suppliers). This role integrates the software components and basic software according to the system integrator's specification. As one of AUTOSAR's proposed benefits, software components may be delivered by third party *software component suppliers*, which is the third role. It is the system integrator's task to integrate all ECUs to a functioning system. It is the ECU integrator's task to integrate all software components on an ECU according to its specification. To avoid inconsistencies and misunderstandings between these three roles a common specification format is required. Therefore AUTOSAR offers a standardized software architecture and a standardized exchange format [1]. However, the standard must still be extended to a standardized specification of timing information [9]. Furthermore, methods must be developed to use this timing information for improved distributed development with respect to the fulfillment of global timing constraints.

4.2 Real-time Constraints in Distributed Development

The system integrator typically specifies the software architecture, the available hardware and the mapping of the software components to the hardware (see Fig. 1). As described in the previous section, ECUs are often developed and integrated by different engineering teams. Thus, the function-based timing constraints now affect the integration work of different teams (see Fig. 3).

In a time-triggered FlexRay system, as it is discussed in this publication, the dependency of function-based and ECU-based (component-based) timing constraints is obvious: physically distributed software components that share data need to communicate via the FlexRay bus. At the same time the software components have to be executed in statically scheduled OS tasks. The overall latency of an end-to-end signal path thus is influenced by one or even several ECU schedules that run the software components and the bus schedule. In a FlexRay system, these schedules typically have a common time base and therefore they are interconnected as one *global schedule*. Changes on one of the schedules can affect the data availability at other ECUs. To reduce complexity, an approach is needed to decouple the development of the ECU schedules and the bus schedule from each other. This means global timing constraints need to be decoupled to component-specific timing constraints. A decoupling of ECU and bus schedule configuration would enable the teams in charge of a certain role to focus on their specific tasks:

- The *system integrator*, who has the knowledge of global function-based timing constraints, must define a bus schedule. In his schedule generation process he must consider global timing constraints. As he knows which ECUs participate in which of these constraints he must define the FlexRay schedule accordingly. To guarantee timing correctness of the system he must specify appropriate timing requirements to the ECU integrators, with respect to the given bus schedule.

- The *ECU integrator* must be able to configure the local ECU schedule according to the bus. He does not necessarily know the global timing constraints, which his components are involved in. Thus, his local changes shall not influence the global timing behavior. Therefore he needs ”configuration boundaries”, i.e. well-defined local timing constraints.

- The *software component supplier* cannot directly influence the timing behavior of his software component as it is scheduled by the ECU integrator. However, he can provide component-based timing properties like execution times or runnable entity execution order constraints.

Timing interfaces are an approach to achieve this desired decoupling of responsibilities. The next section describes how the proposed timing extension of AUTOSAR [9] can be utilized for this purpose.

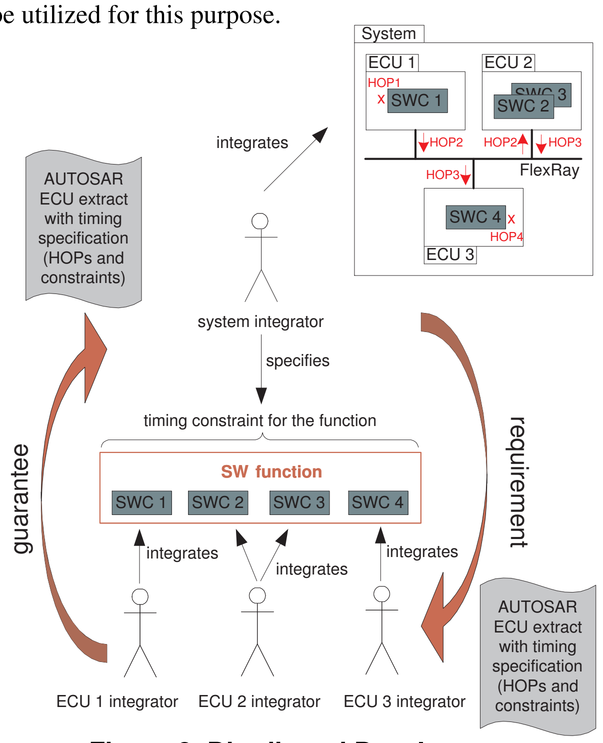

Figure 3. Distributed Development

4.3 Using Timing Interfaces for a more Flexible System Design

In this section we want to outline, how timing interfaces can be used to obtain a more flexible design process. Basically they enable a decoupling of system, ECU and software component timing treatment. The proposed AUTOSAR timing extensions [9] can directly be used to specify system-, ECU- and component timing constraints. The AUTOSAR timing extensions are based on so called timing chains. An expected causal sequence of observable events can be specified by using timing chains. Typically, timing chains follow a signal path, e.g. of an end-to-end communication. In a distributed development scenario such a signal path can affect different teams, e.g. two ECU integrators and a system integrator (signal comes from a sensor ECU, data is transmitted to a controller and actuator ECU). That is, certain parts of the specified timing chain and their events belong to different responsibilities. However, a timing constraint for the complete path must be fulfilled. Figure 3 depicts such kind of scenario. Timing interfaces are based on the concept of contract-based timing specification. Rich Components [3] are a similar concept to add non-functional requirements to component models. In our context, timing interfaces define explicit hand-over-points (HOPs) within timing chains [10] to add timing contract information. Thus, HOPs can be attached to observable events which concern more than one responsibilities, i.e. development teams. In this way, timing constraints can be added to AUTOSAR-compliant entities. It is the system integrator's task to translate function-based end-to-end timing constraints (for the complete timing chain) to appropriate local timing constraints, from one HOP to another HOP of the timing chain. The goal is that the overall timing constraint must in all cases be fulfilled, if the local constraints are fulfilled. In this way each ECU integrator has an own local scheduling goal and does not necessarily need to know about the global dependencies, which his component is participating in. This way his schedule development is decoupled from the bus schedule development. The local constraints can for example directly be attached to the AUTOSAR ECU Extract, which describes the ECU configuration (software components, basic software etc.). Of course, the following challenges have to be considered:

- Global timing constraints have to be translated to local constraints using HOPs. Defining HOPs is not a difficult task as responsibilities are clear and the AUTOSAR timing extensions allow for a well defined set of observable events. However, local constraints must be defined wisely and with respect to the rest of a complete timing chain. The system integrator must precisely define the local constraints to be met.
- The system integrator has to consider that in an automotive system many signal paths with timing chain constraints can exist. First, these timing chains influence each other. Second, several timing chain segments that belong to one responsibility all have to be fulfilled by this role. Thus, the system integrator should propose a set of constraints for each ECU integrator that probably is satisfiable.

If global timing constraints have successfully been translated to local constraints, each role can perform its dedicated scheduling according to all its constraints. That is, bus scheduling is done by the system integrator and the ECU scheduling is done by the ECU integrators. The system integrator proposes the local constraints as *requirements* of the ECU configuration (see Fig. 3).

The timing interface approach targets an iterative process. ECU integrators are able to reflect their scheduling results back to the system integrator as so called *timing guarantees* with respect to the given *requirements*. The system integrator can collect this data and, in a next iteration, can process adapted local constraints. It is unrealistic that a solution is found at once that produces local constraints that can all be fulfilled by the ECU integrators. In the envisioned iterative process a good global solution can be approached. Between process steps, timing requirements and guarantees are exchanged that are attached to AUTOSAR-compliant specification sheets.

5. Improving ECU Configuration and Scheduling

As mentioned before, the ECU integrator retrieves timing constraints to be fulfilled by the implementation and, furthermore, by the scheduling configuration of the local ECU. This section outlines an approach to construct local schedules on the basis of such *"boundaries"*, in the following termed as constraints.

Figure 3 shows the exemplary ECU 2, which is to be configured. It has access to a FlexRay bus and two software components are mapped on this ECU. For simplicity, each SW-C implements one runnable entity (RE) with cycle timing constraints for the execution, e.g. 5 and 20 milliseconds. Basic software was left out of this consideration completely. Obviously, the timing constraints need well-defined entities within the system which they refer to. Therefore, HOPs were introduced in Sec. 4.3. They are also marked in Fig. 2. Based on these HOPs, local timing constraints can be specified.

For example, such a constraint may restrict the latency between two referenced observable events to a minimum and a maximum time value. Typically, this kind of constraint is not the only one needed for a proper description of the ECU's timing behavior. The proposed AUTOSAR timing extensions [9] provide some more of them, especially related to communication.

There are a lot more non-functional constraints affecting the local scheduling problem of the ECU. Runnable entities of SW-Cs have to be mapped to operating system tasks. This mapping decision may be restricted by such non-functional constraints. For example, external requirements can prescribe that certain REs have to be separated for the sake of system's safety. This may lead to a non-functional constraint that these REs cannot be mapped to the same task. Another example for a non-functional constraint is the overhead for preemption, activation and termination of tasks. Certain suppliers provide a fixed task-set for mapping of REs only. Thus, there is no freedom of task definition with other cycle times anymore.

The constraints mentioned in the last paragraph do not affect the scheduling problem but rather the task-set construction problem. Much work has been done on task-dispatching algorithms, but either they rely on a previously defined set of tasks [7] to be scheduled or dynamic scheduling algorithms [5] are discussed. Both of these are not feasible for automotive ECU implementations. The following section outlines basic ideas for a task-construction approach. Task-construction and scheduling is a search problem. Of course, this concept regards all non-functional constraints mentioned before.

5.1 Task Set Construction

Typically, there are different severities of constraints. Certain constraints are really stringent and have to be fulfilled in any case. Others can be relaxed if no ECU configuration can be found otherwise. This leads to a weighting approach for each constraint to be regarded. One example for a stringent constraint is the required execution cycle time of runnable entities. This helps in reducing the search space to be traversed.

Let RE be a runnable entity, $t(x)$ the cycle time of (task or RE) x and T the task-set to be constructed. The allocation between runnable and task cycle times leads to a very easy but mighty heuristic:

$$\forall RE : \exists j \in T : t(j) \mid t(RE) \tag{6}$$

With this restriction, the potential tasks for the RE mapping are directly derived. Applying additional constraints like the exclusion of mapping two runnable entities together on one task leads to additional implications to the constructed task-set.

The weightings not only help to reduce search space, but also serve as a quality metric for the constructed task set. If an appropriate task-set has been found, standard approaches [2] can be applied to this set respecting local execution budgets. The found schedule is planned statically by means of a schedule table. Thus, it is possible to traverse given data paths (available in the software architecture) and summing up execution times of involved REs. This results in a declaration of latency within the configured ECU. Providing this latency over the timing interface, the ECU scheduling activity can guarantee the timing constraints required by the *System Schedulability* step.

6. Summary

This paper presented a comprehensive design approach for improving the distributed development of FlexRay-based automotive real-time systems. Mainly three challenges were discussed. First, the relation and the basic compatibility of communication-specific cycle timing constraints regarding a given FlexRay configuration was discussed. This leads to a method for checking their compatibility. Second, the challenge of distributed development of modern onboard networks by many different teams was outlined. This leads to an approach for collaboration improvement. Therefore, the concept of timing interfaces was explained. Third, the configuration of time-triggered ECU schedules with respect to timing constraints as well as other non-functional constraint types was considered.

7 Acknowledgments

The authors would like to thank Manfred Broy (Technische Universität München), Wolfram Hardt (Technische Universität Chemnitz) and Wolfgang Rosenstiel (Universität Tübingen) for their mentoring and many fruitful discussions.

References

[1] AUTOSAR Development Partnership. AUTOSAR Standard Specification. http://www.autosar.org, 2008.

[2] R. Baumgartl. Echtzeitsysteme. Vorlesungsskript Technische Universität Chemnitz, 2006.

[3] W. Damm, A. Votintseva, A. Metzner, B. Josko, T. Peikenkamp, and E. Böde. Boosting Re-use of Embedded Automotive Applications Through Rich Components. In *Proceedings of Foundations of Interface Technologies*, 2005.

[4] FlexRay Consortium. FlexRay Protocol Specification. http://www.flexray.com, 2007.

[5] G. Fohler. Analyzing a Pre Run-Time Scheduling Algorithm and Precedence Graphs. Research Report 13/1992, Technische Universität Wien, Institut für Technische Informatik, 1992.

[6] H. Kopetz, M. Braun, C. Ebner, A. Krüger, D. Millinger, R. Nossal, and A. Schedl. The Design of Large Real-Time Systems: The Time-Triggered Approach. In *Proceedings of the 16th Real-Time Systems Symposium*, Pisa, Italy, 1995.

[7] R. Nossal. Pre-Runtime Planning of a Real-Time Communication System. Research Report 1/1996, Technische Universität Wien, Institut für Technische Informatik, 1996.

[8] S. Reichelt, K. Schmidt, F. Gesele, N. Seidler, and W. Hardt. Nutzung von FlexRay als zeitgesteuertes automobiles Bussystem im AUTOSAR-Umfeld. In *Mobilität und Echtzeit*, pages 79–87, 2007.

[9] O. Scheickl and M. Rudorfer. Automotive Real Time Development Using a Timing-augmented AUTOSAR Specification. *Proceedings of ERTS2008*, 4, 2008.

[10] O. Scheickl, M. Rudorfer, C. Ainhauser, N. Feiertag, and K. Richter. How Timing Interfaces in AUTOSAR can Improve Distributed Development of Real-Time Software. In *GI Jahrestagung (2)*, pages 662–667, 2008.