Modular Scheduling of Distributed Heterogeneous Time-Triggered Automotive Systems

Martin Lukasiewycz
TUM CREATE
Singapore
martin.lukasiewycz@tum-create.edu.sg

Reinhard Schneider, Dip Goswami, Samarjit Chakraborty
TU Munich
Munich, Germany
{reinhard.schneider, dip.goswami, samarjit.chakraborty}@rcs.ei.tum.de

ABSTRACT

This paper proposes a modular framework that enables a scheduling for time-triggered distributed embedded systems. The framework provides a symbolic representation that is used by an Integer Linear Programming (ILP) solver to determine a schedule that respects all bus and processor constraints as well as end-to-end timing constraints. Unlike other approaches, the proposed technique complies with automotive specific requirements at system-level and is fully extensible. Formulations for common time-triggered automotive operating systems and bus systems are presented. The proposed model supports the automotive bus systems FlexRay 2.1 and 3.0. For the operating systems, formulations for an eCos-based non-preemptive component and a preemptive OSEK-time operating system are introduced. A case study from the automotive domain gives evidence of the applicability of the proposed approach by scheduling multiple distributed control functions concurrently. Finally, a scalability analysis is carried out with synthetic test cases.

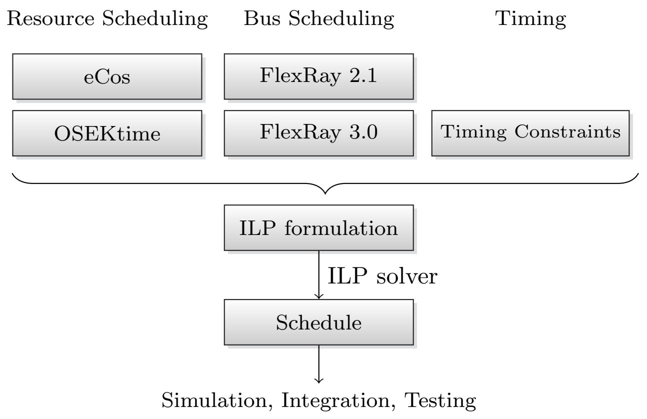


Figure 1: Design flow of the proposed modular scheduling framework.

1. INTRODUCTION

With the introduction of the FlexRay bus, the design paradigm in the automotive domain shifted towards time-triggered systems. In case the Electronic Control Units (ECUs) are not synchronized with the bus, undesired delays and jitter in the communication are possible. Therefore, a synchronous time-triggered system that exploits the advantages of the high data-rate and short transmission times of the FlexRay bus is desirable. Functions that are executed periodically benefit from such a time-triggered architecture, leading to an improved control quality due to the deterministic behavior and minimal jitter. At the same time, simulation, integration, and testing efforts are reduced significantly due to the predictability of the system. These properties make such time-triggered systems ideal candidates for next-generation automotive architectures. In particular, such architectures have a growing need in electric vehicles due to the increased software and electronics content.

Contributions of the paper. In this paper, a framework based on Integer Linear Programming (ILP) is presented that performs schedule synthesis for time-triggered systems. The proposed modular framework is illustrated in Figure 1. The scheduling is performed at task-level, compliant with the AUTOSAR specification. A model for the FlexRay bus including versions 2.1 and 3.0 [1, 5, 15] is presented. For the operating systems, a non-preemptive scheduler based on eCos [4] and the preemptive Last In - First Out (LIFO) scheduler in OSEKtime [11] are introduced in the model. Finally, where necessary, a generic formulation constrains the end-to-end delay for each path in a function. This formulation has the following advantages: (1) The models of each component are generated separately, (2) an incremental scheduling of legacy systems is possible, and (3) there are no restrictions on the maximal end-to-end delays or distribution of the application across a communication cycle.

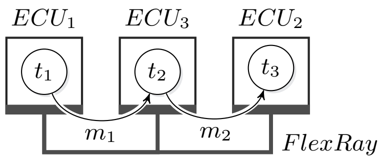
Organization of the paper. The remainder of the paper is outlined as follows: Section 2 discusses related work. Section 3 introduces the scheduling framework, including the linear formulations for the eCos-based and OSEKtime operating systems as well as the FlexRay bus 2.1 and 3.0. In Section 4, a case study and scalability analysis are presented. Finally, Section 5 summarizes the contributions of the paper.

2. RELATED WORK

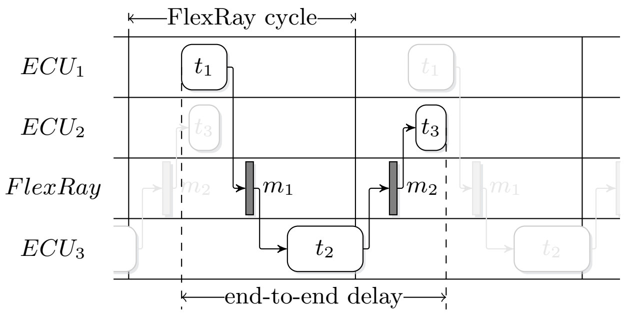
Time-triggered systems in the automotive domain are based on the FlexRay [5] bus system. FlexRay was developed by an industrial consortium, including *BMW*, *Daimler*, *General Motors*, and *Volkswagen*. A large part of recent research focuses on scheduling the bus for the *asynchronous* case, i.e., the ECUs are not synchronized to the bus. The goal of these scheduling strategies is mostly the minimization of the required number of slots in order to ensure the extensibility of the bus for future applications. The scheduling of the static segment is discussed in [3, 6, 10, 13] while the dynamic segment scheduling is presented in [7]. Another field of research on time-triggered systems in the automotive domain is the determination of end-to-end delays of FlexRay-based systems for the dynamic segment [8, 12].

Recently, [16, 17] presented an ILP formulation for the synchronous scheduling of FlexRay-based systems at *job-level*. In contrast to this, the proposed approach in this paper provides a framework for scheduling buses and ECUs at the *task-level*, thereby enabling an AUTOSAR compliant system. Moreover, it is not necessary to model the entire system within the general hyper-period of all functions. By providing a modular approach that allows to model each ECU and bus independently, necessary end-to-end latency constraints are added separately to restrict the delays of functions. This enables a more efficient formulation and better extensibility. In this paper, it is shown how to model either FlexRay 2.1 or 3.0 for the buses

Supported in part by the National Research Foundation (NRF), Singapore.



(a) Topology



(b) Scheduling

Figure 2: Example topology (a) with three resources $R = \{ECU_1, ECU_2, ECU_3\}$, executing a function with three tasks and two messages. A scheduling (b) determines the start times of all tasks as well as the slot assignment for all messages.

as well as a non-preemptive operating system based on eCos and the OSEKtime operating systems for ECUs.

3. SCHEDULING FRAMEWORK

In the following, the scheduling framework is presented. An illustration of the design flow is given in Figure 1. The models for resource scheduling, bus scheduling, and end-to-end timing constraints are defined separately and added to the ILP formulation. An ILP formulation consists of a set of linear constraints expressed as equalities and inequalities, respectively, with either real or integer variables, see [2]. In the next step, the ILP formulation is solved with an ILP solver to obtain a feasible schedule. Finally, this schedule might be simulated, integrated into a full system, and tested.

In this paper, an objective function is applied in the experimental results with the goal to minimize the end-to-end latencies. However, the focus of this paper is the presentation of a modular framework and therefore a detailed discussion of possible objective functions is omitted. Potential objective functions are discussed in [16] and they are also applicable here.

This section is organized as follows. First, the problem definition is presented using an illustrative example. The task scheduling for the ECUs is introduced for the time-triggered non-preemptive eCos-based and the preemptive LIFO OSEKtime operating systems. The message scheduling is presented for both FlexRay 2.1 and 3.0. Finally, constraints are introduced to define an upper bound for the end-to-end delay of functions.

3.1 Problem Definition

We consider a system with a single bus and multiple ECUs as illustrated by a system consisting of three ECUs in Figure 2(a). Each ECU might also have dedicated links to sensors or actuators. Note that the following formulation can be extended in a straightforward manner to consider multiple buses.

An application is executed on the system and usually consists of multiple independent functions. Each function consists of data-dependent processes that communicate directly if they are implemented on the same ECU, adjacent sensors, or actuators. Each function and its processes are executed with a specific period with a predefined offset. If the communication is carried out over the bus, a message has to be sent in a specified slot on the FlexRay bus. Each function might have a specified maximal delay that depends on the characteristics of the function. The hardware setup is based on prototype components from *Elektrobit* (EB) and the software layer was synthesized using SIMTOOLS and a model-based design flow, see [14].

The system is defined as follows:

- $T = T_p \cup T_m$ - set of all tasks with T_p being processes and T_m being messages
- R - set of all ECUs/sensors/actuators
- $target : T \rightarrow R$ - determine the target resource of a process $t \in T_p$ and the sender of a message $m \in T_m$.

Based on the operating systems and scheduling policies of the ECUs as well as the bus scheduling, a feasible schedule needs to be determined that respects all given end-to-end delays. A schedule for the example in Figure 2(a) is given in Figure 2(b). The schedule is defined by specific start times for each process as well as a slot assignment for the messages.

3.2 Resource Scheduling

The processes on each ECU (including sensors and actuators) have to be scheduled in compliance with the used operating system scheduler. Here, the processes T_r are scheduled for each ECU $r \in R$ separately. Each process is associated with its period p_t and worst-case execution time e_t.

The given constants are defined as follows:

- $T_r = \{t | target(r) = t, t \in T_p\}$ - set of all processes $t \in T_p$ running on ECU $r \in R$
- p_t - period of process $t \in T_p$
- e_t - execution time of process $t \in T_p$
- $H_r = \underset{t \in T_r}{lcm}(p_t)$ - hyper-period of ECU $r \in R$ (lcm determines the least common multiple)

For each ECU, the schedule is determined by the offset of each process (offset from the start of the first communication cycle of the bus). Thus, the following value has to be determined for each process:

- o_t - offset of process $t \in T_p$

In the following, the start time of processes equals the offset, i.e, $o_t = \mathbf{s_t}$. The formulation for a non-preemptive time-triggered scheduling and a preemptive scheduling is proposed. For the non-preemptive case, an eCos-based operating system is assumed. For the preemptive scheduling case, OSEKtime is used which is using a LIFO scheduler.

3.2.1 eCos-based System Scheduling

An eCos-based operating system is used as a non-preemptive time-triggered component. Thus, each process is started with a predefined offset between 0 and its period and always finishes its execution before any other process is started. An example of the eCos-based scheduling with two processes is illustrated in Figure 3.

The formulation for the eCos operating system requires the following variables:

- $\mathbf{s_t} \in \mathbb{R}$ - start time of process $t \in T_p$
- $\mathbf{f_t} \in \mathbb{R}$ - finish time of process $t \in T_p$
- $\mathbf{of_t} \in \{0, 1\}$ - offset for finish time $t \in T_p$
- $\mathbf{y^{i,j}_{t,\tilde{t}}} \in \{0, 1\}$ - 1 if job i of process $t \in T_p$ finished before job j of process $\tilde{t} \in T_p$

The bounds on the variables are determined as follows:
$\forall r \in R, t \in T_r$:

$$0 \leq \mathbf{s_t} \leq p_t \tag{1}$$

$$0 \leq \mathbf{f_t} \leq p_t \tag{2}$$

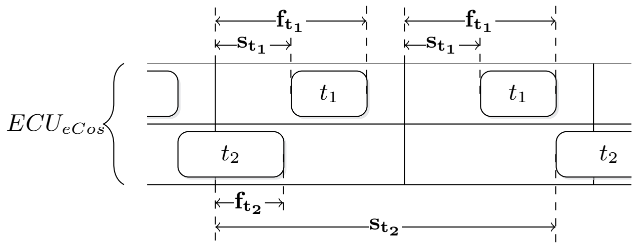


Figure 3: Example of an eCos-based non-preemptive scheduling for two processes t_1 and t_2 (periods: $2 \cdot p_{t_1} = p_{t_2}$) on a resource ECU_{eCos}. Preemption is not allowed such that two processes never coincide.

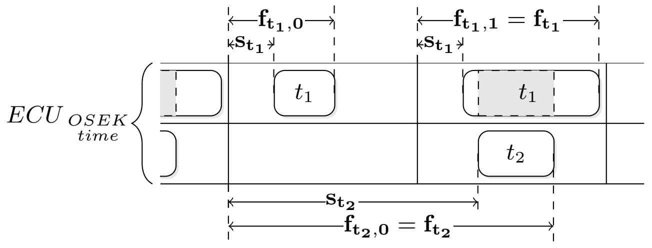

$\mathbf{f_{t_1,0}}$
$\mathbf{s_{t_1}}$
$\mathbf{f_{t_1,1} = f_{t_1}}$
$\mathbf{s_{t_1}}$
$ECU_{\substack{OSEK\\time}}$
t_1
t_1
t_2
$\mathbf{s_{t_2}}$
$\mathbf{f_{t_2,0} = f_{t_2}}$

Figure 4: Example of the OSEKtime scheduling with two processes t_1 and t_2 (periods: $2 \cdot p_{t_1} = p_{t_2}$) on a resource $ECU_{OSEKtime}$. Process t_2 preempts t_1 every second cycle.

The constraints are determined as follows:
$\forall t \in T_r$:

$$\mathbf{f_t} + p_t \cdot \mathbf{of_t} = \mathbf{s_t} + e_t \tag{3}$$

$\forall t, \tilde{t} \in T_r, t \neq \tilde{t}, i = \{0, .., \frac{2 \cdot H_r}{p_t} - 1\}, j = \{0, .., \frac{2 \cdot H_r}{p_{\tilde{t}}} - 1\}$:

$$i \cdot p_t + \mathbf{s_t} + e_t \leq j \cdot p_{\tilde{t}} + \mathbf{s_{\tilde{t}}} + 2 \cdot H_r \cdot (1 - \mathbf{y^{i,j}_{t,\tilde{t}}}) \tag{4}$$

$$j \cdot p_{\tilde{t}} + \mathbf{s_{\tilde{t}}} + e_{\tilde{t}} \leq i \cdot p_t + \mathbf{s_t} + 2 \cdot H_r \cdot \mathbf{y^{i,j}_{t,\tilde{t}}} \tag{5}$$

The bounds (1) and (2) specify that the start and finish times are within the period of the process. Constraint (3) determines the finish time of each process. In case the process finishes in the next cycle, the finish time is smaller than the start time (see process t_2 in Figure 3). In this case, the variable $\mathbf{of_t}$ becomes 1 to fulfill constraint (3). The constraints (4) and (5) ensure that two processes never preempt each other within two hyper-periods ($2 \cdot H_r$). The variable $\mathbf{y^{i,j}_{t,\tilde{t}}}$ is used for switching, i.e., one of the constraints (4) or (5) is trivially satisfied depending on $\mathbf{y^{i,j}_{t,\tilde{t}}}$.

3.2.2 *OSEKtime Scheduling*

The OSEKtime operating system is a preemptive operating system that uses a LIFO scheduler. All processes are executed depending on a specified offset that lies between 0 and their period. A process that is executed might be preempted by another process. In this case, the preempted process is suspended until the process task that was started later finishes its execution. An example of this scheduling is given in Figure 4. Here, process t_2 preempts t_1 in the second cycle. To determine the maximal finish time, the finish times of jobs have to be considered as illustrated in Figure 4 for process t_1.

The formulation for the OSEKtime operating system requires the following variables:

- $\mathbf{s_t} \in \mathbb{R}$ - start time of process $t \in T_p$ (equals o_t)
- $\mathbf{f_t} \in \mathbb{R}$ - finish time of process $t \in T_p$
- $\mathbf{f_{t,i}} \in \mathbb{R}$ - finish time of job i of process $t \in T_p$
- $\mathbf{of_t} \in \{0, 1\}$ - offset for finish time $t \in T_p$
- $\mathbf{y^{i,j}_{t,\tilde{t}}} \in \{0, 1\}$ - 1 if job i of process $t \in T_p$ starts before job j of process $\tilde{t} \in T_p$
- $\mathbf{z^{i,j}_{t,\tilde{t}}} \in \{0, 1\}$ - 1 if job i of process $t \in T_p$ is preempted by job j of process $\tilde{t} \in T_p$

The bounds on the variables are determined as follows:
$\forall r \in R, t \in T_r$:

$$0 \leq \mathbf{s_t} \leq p_t \tag{6}$$

$$0 \leq \mathbf{f_t} \leq p_t \tag{7}$$

$\forall r \in R, t \in T_r, i = \{0, .., \frac{2 \cdot H_r}{p_t} - 1\}$:

$$e_t \leq \mathbf{f_{t,i}} \leq 2 \cdot p_t \tag{8}$$

The constraints are determined as follows:
$\forall r \in R, t, \tilde{t} \in T_r, t \neq \tilde{t}, i = \{0, .., \frac{2 \cdot H_r}{p_t} - 1\}, j = \{0, .., \frac{2 \cdot H_r}{p_{\tilde{t}}} - 1\}$:

$$i \cdot p_t + \mathbf{s_t} < j \cdot p_{\tilde{t}} + \mathbf{s_{\tilde{t}}} + 2 \cdot H_r \cdot (1 - \mathbf{y^{i,j}_{t,\tilde{t}}}) \tag{9}$$

$$j \cdot p_{\tilde{t}} + \mathbf{s_{\tilde{t}}} < i \cdot p_t + \mathbf{s_t} + 2 \cdot H_r \cdot \mathbf{y^{i,j}_{t,\tilde{t}}} \tag{10}$$

$$\mathbf{y^{i,j}_{t,\tilde{t}}} = 1 - \mathbf{y^{j,i}_{\tilde{t},t}} \tag{11}$$

$$\mathbf{z^{i,j}_{t,\tilde{t}}} \leq \mathbf{y^{i,j}_{t,\tilde{t}}} \tag{12}$$

$$\mathbf{z^{i,j}_{t,\tilde{t}}} + \mathbf{z^{j,i}_{\tilde{t},t}} \leq 1 \tag{13}$$

$$i \cdot p_t + \mathbf{f_{t,i}} \leq j \cdot p_{\tilde{t}} + \mathbf{s_{\tilde{t}}} + 2 \cdot H_r \cdot \mathbf{z^{i,j}_{t,\tilde{t}}} + 2 \cdot H_r \cdot (1 - \mathbf{y^{i,j}_{t,\tilde{t}}}) \tag{14}$$

$$j \cdot p_{\tilde{t}} + \mathbf{f_{\tilde{t},j}} \leq i \cdot p_t + \mathbf{f_{t,i}} + 2 \cdot H_r \cdot (1 - \mathbf{z^{i,j}_{t,\tilde{t}}}) + 2 \cdot H_r \cdot (1 - \mathbf{y^{i,j}_{t,\tilde{t}}}) \tag{15}$$

$\forall r \in R, t \in T_r, i = \{0, .., \frac{2 \cdot H_r}{p_t} - 1\}$:

$$\mathbf{f_{t,i}} = \mathbf{s_t} + e_t + \sum_{\tilde{t} \in T_r \setminus t} \sum_{j = \{0, .., \frac{2 \cdot H_r}{p_{\tilde{t}}} - 1\}} \mathbf{z^{i,j}_{t,\tilde{t}}} \cdot e_{\tilde{t}} \tag{16}$$

$$\mathbf{f_t} + p_t \cdot \mathbf{of_t} \geq \mathbf{f_{t,i}} \tag{17}$$

$$\mathbf{f_t} + p_t \cdot \mathbf{of_t} - \mathbf{s_t} \leq p_t \tag{18}$$

The bounds (6) and (7) specify that the start and finish times are within the period of the process. On the other hand, the finish time of each job of a process might be different as expressed in bounds (8) and illustrated in Figure 4 for process t_1. Constraints (9) and (10) state that if job i of process t starts before job j of process $\tilde{t}$, $\mathbf{y^{i,j}_{t,\tilde{t}}}$ becomes 1 and 0 otherwise. Constraint (11) ensures the consistency of start time order of jobs. Constraint (12) states that a job i of process t can only be preempted by a job j of process $\tilde{t}$ if the latter starts earlier. Constraint (13) prevents mutual preemption. Constraints (14) and (15) ensure that a job i of process t finishes after a job j of process $\tilde{t}$ if it is preempted by this other job. Constraint (16) determines the finish time of a job i of process t based on the preemption of all other jobs. Constraint (17) sets the finish time of process t. The finish time $\mathbf{f_t}$ is the maximum of all job finish times $\mathbf{f_{t,i}}$ and is adapted to the bounds (7) using the binary variable $\mathbf{of_t}$. Constraint (18) states that the Worst Case Response Time (WCRT) of a task always has to be less than its period.

3.3 Bus Scheduling (FlexRay)

The FlexRay bus is used as a central communication bus in the proposed architecture. Again, the framework may be easily extended to support other bus protocols as well. This bus system enables a synchronization of the ECUs and the establishment of a fully time-triggered system. For message scheduling, we assume the static segment of FlexRay is used. In the automotive domain it is common to use the static segment for critical data and the dynamic segment for maintenance, calibration, and diagnosis which have no constraints on end-to-end delays. The static segment of the FlexRay is organized in n_{fx} static slots with each slot having a duration of e_{fx} and available payload of l_{fx} in bytes. The static segment is transmitted in the beginning of every FlexRay cycle

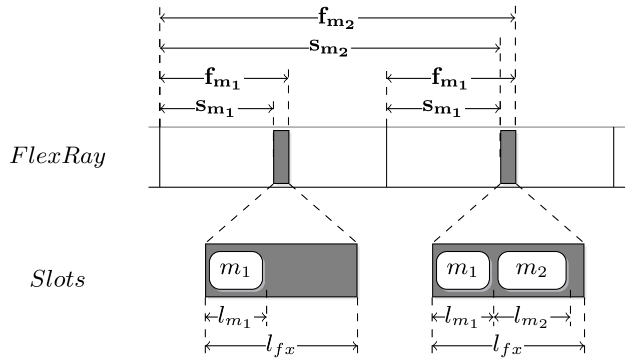


Figure 5: An example of a FlexRay scheduling for two messages m_1 and m_2 that are transmitted in the same slot for two cycles. The repetition for m_1 is 1 and for m_2 it is 2.

which has a duration of p_{fx}. FlexRay is available in the versions 2.1, see [10], and 3.0, see [15], with some differences: In FlexRay 2.1, the number of cycles is 64 and each slot is assigned to at most one ECU. In FlexRay 3.0, the number of cycles is a multiple of 2 with a lower bound of 8 and upper bound of 64 while a slot might be assigned to different ECUs for each cycle. Thus, FlexRay 3.0 is more flexible but also more complex to model.

A message that is transmitted on the FlexRay bus, is defined by the repetition that is deduced from the period, the execution time that equals the slot duration, and the length in bytes. An example of a FlexRay scheduling is illustrated in Figure 5.

The given constants are defined as follows:

- p_{fx} - cycle duration of the FlexRay bus
- n_{fx} - number of static slots
- c_{fx} - number of cycles (64 for FlexRay 2.1, between 8 and 64 for FlexRay 3.0)
- e_{fx} - duration of a static slot (it holds $n_{fx} \cdot e_{fx} \le p_{fx}$)
- l_{fx} - capacity of one FlexRay static slot in bytes
- p_m - period of message $m \in T_m$
- $r_m = \frac{p_m}{p_{fx}}$ - repetition of message $m \in T_m$ with $c_{fx} \% r_m = 0$
- $e_m = e_{fx}$ - transmission time of message $m \in T_m$
- l_m - length of a message $m \in T_m$ in bytes

For each message, a slot and base cycle has to be determined. From this value, the assignments of slots to ECUs is deduced implicitly. Thus, the following values have to be determined for each message:

- s, b - slot id and base cycle for each message $m \in T_m$

The slot and base is encoded in the binary variable $[\mathbf{s},\mathbf{b}]_\mathbf{m}$. Independent of the FlexRay version, the following variables are required:

- $\mathbf{s_m} \in \mathbb{R}$ - start time of message $m \in T_m$
- $\mathbf{f_m} \in \mathbb{R}$ - finish time of message $m \in T_m$
- $[\mathbf{s},\mathbf{b}]_\mathbf{m} \in \{0,1\}$ - slot and base cycle pair for each message $m \in T_m$ with $s \in \{1,..,n_{fx}\}$, $b \in \{0,..,r_m-1\}$

These variables are bounded as follows:
$\forall m \in T_m$:

$$0 \le \mathbf{s_m} \le p_m - e_{fx} - (p_{fx} - n_{fx} \cdot e_{fx}) \tag{19}$$

$$e_{fx} \le \mathbf{f_m} \le p_m - (p_{fx} - n_{fx} \cdot e_{fx}) \tag{20}$$

These constraints are defined as follows:
$\forall m \in T_m$:

$$\sum_{s \in \{1,..,n_{fx}\}} \sum_{b \in \{0,..,r_m-1\}} [\mathbf{s},\mathbf{b}]_\mathbf{m} = 1 \tag{21}$$

$$\mathbf{s_m} = \sum_{s \in \{1,..,n_{fx}\}} \sum_{b \in \{0,..,r_m-1\}} ((s-1) \cdot e_{fx} + b \cdot p_{fx}) \cdot [\mathbf{s},\mathbf{b}]_\mathbf{m} \tag{22}$$

$$\mathbf{f_m} = \mathbf{s_m} + e_{fx} \tag{23}$$

$\forall s \in \{1,..,n_{fx}\}, b \in \{0,.., \frac{\operatorname{lcm}_{m \in T_m}(p_m)}{p_{fx}} - 1\}$:

$$\sum_{m \in T_m} l_m \cdot [\mathbf{s},\mathbf{b\%r_m}]_\mathbf{m} \le l_{fx} \tag{24}$$

The bounds (19) and (20) constrain the start and finish times of the transmission of a message/slot. The upper bound on the start time is defined by the period without the transmission of the last slot and the dynamic segment of FlexRay which is transmitted after the static segment. The bounds for the finish time are shifted by the transmission time of a slot e_{fx}. Constraint (21) states that each message is scheduled in exactly one slot at a specific base cycle. Constraint (22) determines the start time of a message transmission based on the slot and base cycle. Constraint (23) defines the finish time of a message transmission depending on the duration of the transmission of a static slot. Constraint (24) ensures that the capacity of a slot is not exceeded.

Depending on the version of the FlexRay protocol, additional constraints have to be defined that ensure a correct assignment of slots to ECUs as presented in the following.

3.3.1 FlexRay 2.1 (Slot to ECU assignment)

For FlexRay 2.1, additional variables have to be defined that specify the assignment of slots to ECUs.

- $\mathbf{s_r} \in \{0,1\}$ - becomes 1 if slot s is assigned to ECU r and 0 otherwise

These constraints are defined as follows:
$\forall s \in \{1,..,n_{fx}\}$:

$$\sum_{r \in R} \mathbf{s_r} \le 1 \tag{25}$$

$\forall m \in T_m, s \in \{1,..,n_{fx}\}, b \in \{0,..,r_m-1\}, r = \mathit{target}(m)$:

$$[\mathbf{s},\mathbf{b}]_\mathbf{m} \le \mathbf{s_r} \tag{26}$$

Constraint (25) states that a slot can be assigned to a most one ECU. Constraint (26) ensures that if a message is sent in a specific slot, the slot is assigned to the sending ECU.

3.3.2 FlexRay 3.0 (Slot to ECU assignment)

For FlexRay 3.0, additional variables have to defined that specify the assignment of slots and at specific base cycles to ECUs.

- $[\mathbf{s},\mathbf{b}]_\mathbf{r} \in \{0,1\}$ - becomes 1 if slot s is assigned to ECU r in cycle b and 0 otherwise

These constraints are defined as follows:
$\forall s \in \{1,..,n_{fx}\}, b \in \{0,.., \frac{\operatorname{lcm}_{m \in T_m}(p_m)}{p_{fx}} - 1\}$:

$$\sum_{r \in R} [\mathbf{s},\mathbf{b}]_\mathbf{r} \le 1 \tag{27}$$

$\forall m \in T_m, s \in \{1,..,n_{fx}\}, b \in \{0,.., \frac{\operatorname{lcm}_{m \in T_m}(p_m)}{p_{fx}} - 1\}, r = \mathit{target}(m)$:

$$[\mathbf{s},\mathbf{b\%r_m}]_\mathbf{m} \le [\mathbf{s},\mathbf{b}]_\mathbf{r} \tag{28}$$

Constraint (27) states that a slot at a specific cycle can be assigned to a most one ECU. Constraint (28) ensures that if a message is sent in a specific slot and cycle, the slot is assigned to the sending ECU at this cycle.

3.4 Timing Constraints

Some functions have strict end-to-end timing constraints. In this case, it becomes necessary to define constraints that restrict the latencies of these functions. The functions and maximal delays are defined as follows:

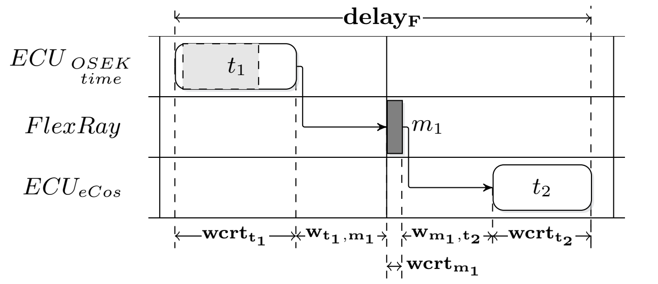


Figure 6: Illustration of the determination of the end-to-end delay of a function F as the sum of all waiting times and Worst Case Response Times (WCRTs). Note that process t_1 is preempted by some other process on the resource $ECU_{OSEKtime}$.

- $\mathcal{F}$ - set of functions that have to fulfill end-to-end timing constraints
- $maxdelay_F \in \mathbb{R}$ - maximal tolerated delay of a function $F \in \mathcal{F}$

Additionally, Π_F determines all paths of a function from the source processes (sensors) to the sink processes (actuators). Here, it is assumed that functions do not have cycles. In case of cycles, the Π_F has to be adapted appropriately to break the cycles.

The end-to-end delay is determined in an additive manner as follows. Along the paths the delay is determined by the sum of the execution times of all processes and messages as well as the waiting time between the data-dependent tasks. This approach is illustrated in Figure 6.

To constrain the end-to-end delay, the following variables are required:

- $\mathbf{delay_F} \in \mathbb{R}$ - the maximal delay of a function $F \in \mathcal{F}$
- $\mathbf{wcrt_t} \in \mathbb{R}$ - worst case response time of task $t \in T$
- $\mathbf{w_{t,\tilde{t}}} \in \mathbb{R}$ - waiting time between the finish of $t \in T$ and start of $\tilde{t} \in T$
- $\mathbf{ow_{t,\tilde{t}}} \in \{0,1\}$ - 0 if $t \in T$ starts before $\tilde{t} \in T$, 1 otherwise

The bounds on these variables are defined as follows:
$\forall F \in \mathcal{F}, \pi \in \Pi_F, (t,\tilde{t}) \in \pi :$

$$0 \leq \mathbf{w_{t,\tilde{t}}} \leq p_t \tag{29}$$

$\forall F \in \mathcal{F}, \pi \in \Pi_F :$

$$\sum_{i=1}^{x} e_{t_x} \leq \mathbf{delay_F} \leq maxdelay_F \tag{30}$$

The constraints are defined as follows:
$\forall F \in \mathcal{F}, t \in F \cap T_p :$

$$\mathbf{wcrt_t} = \begin{cases} e_t, & \text{if } t \text{ is on FlexRay or eCos,} \\ \mathbf{f_t} + p_t \cdot \mathbf{of_t} - \mathbf{s_t}, & \text{if } t \text{ is on OSEKtime.} \end{cases} \tag{31}$$

$\forall F \in \mathcal{F}, \pi \in \Pi_F, t \in \pi, (t,\tilde{t}) \in E_T :$

$$\mathbf{w_{t,\tilde{t}}} = \mathbf{s_{\tilde{t}}} - \mathbf{f_t} + p_t \cdot \mathbf{ow_{t,\tilde{t}}} \tag{32}$$

$\forall F \in \mathcal{F}, \pi \in \Pi_F :$

$$\mathbf{delay_F} \geq \sum_{t\in\pi} \mathbf{wcrt_t} + \sum_{(t,\tilde{t})\in\pi} \mathbf{w_{t,\tilde{t}}} \tag{33}$$

$\forall F \in \mathcal{F}, t, \tilde{t} \in F \cap (T_s \vee T_a) :$

$$\mathbf{s_t} = \mathbf{s_{\tilde{t}}} \tag{34}$$

The bounds on the waiting time between two data-dependent tasks is between 0 and the period of the function as stated in (29). The maximal delay is restricted in the boundary constraint (30). Constraint (31) determines the Worst Case Response Time (WCRT) of all tasks. Here, the WCRT of messages on the FlexRay bus and processes on the eCos operating system are constant. The WCRT of processes on ECUs with the OSEKtime operating system are determined by the difference between the latest relative finish time and the start time. Constraint (32) determines the waiting time between two data-dependent tasks (two neighboring tasks along a path). The binary variable $\mathbf{ow_{t,\tilde{t}}}$ ensures that the waiting time is always within the predefined bounds, i.e., not negative. Constraint (33) determines the end-to-end delay of a function. The end-to-end delay is defined as the maximal latency along all paths of a function. Constraint (34) is optional and necessary for control functions where the start time of sensor tasks (T_s) and actuator tasks (T_a), respectively, have to be identical.

function F	p	$maxdelay_F$	$\lvert T_p\rvert$	$\lvert T_m\rvert$
legacy0 *	5	-	8	3
legacy1 *	5	-	8	5
legacy2 *	5	-	8	5
legacy3 *	40	10	4	3
DC Motor	80	20	4	3
cruise control	20	10	5	3
car suspension	10	5	6	4
airbag system	5	3	5	3

Table 1: The used case study: Overall eight functions have to be scheduled. Given is the period p, maximal end-to-end delay $maxdelay_F$, the number of processes $|T_p|$, and the number of messages $|T_m|$ for each function.

4. EXPERIMENTAL RESULTS

In this section, we present our experimental results using an automotive subsystem as a case study consisting of multiple control functions. Finally, a scalability analysis is carried out using synthetic test cases. For all experiments, the CPLEX ILP solver [9] is used. All experiments were carried out on an Intel Core i5 2.53 GHz with 4 GB RAM.

4.1 Case Study

We consider a FlexRay subsystem consisting of six ECUs. Three of the ECUs used the eCos-based non-preemptive operating systems, the other three used OSEKtime. The FlexRay bus was configured to have a cycle duration of 5ms with 60 static slots in the static segment. The payload of a slot was 40 bytes and the duration of a slot was 0.065ms. FlexRay 3.0 was used. Overall eight functions consisting of 48 processes and 29 messages are executed in this system. A detailed overview of these functions is given in Table 1.
Three of the functions in Table 1 are common automotive control functions: a DC motor speed control, a cruise control system and a car suspension system. Additionally, an airbag system with a high sampling rate of $5ms$ and a very low maximal end-to-end latency was used. The legacy functions are assumed to be already scheduled.
Control Functions: The $maxdelay_F$ of all control functions, see Table 1, was determined by an investigation the control model and its dynamics. Each control function was sampled with a constant period (sampling interval) p. The overall feedback control model was a discrete-time *sampled-data* system where the feedback signals experience a constant delay. The following feedback control model was considered,

$$\dot{x}(k+1) = Ax(k) + B_0(\tau)u(k) + B_1(\tau)u(k-1), \tag{35}$$

where k indicates k^{th} sampling instant, i.e., $p \cdot k$ time units, $x(k) \in R^n$, R^n are the *states* of the system, $u(k)$ is the input to the system, $A \in R^{n\times n}$, $B_{0,1}(\tau) \in R^{n\times 1}$ are the system and input matrices, respectively. The system *states* $x(k)$ are *measured* by n processes in T_p. Additionally, another two tasks are responsible for computation of control algorithm $u(k)$ and *actuation*. Hence, for a control application of dimension n it holds $|T_p| = n + 2$ and $|T_m| \leq n + 1$ since communicating processes on the same ECU do not require messages. The periods of all processes and messages of a function are equal. τ is the constant feedback delay and $\tau = \mathbf{delay_F}$.

		delay$_{\mathbf{F}}$	
function F	*maxdelay*$_F$	*feas*	*opt*
DC Motor	20	9.175	2.275
cruise control	10	8.195	3.5
car suspension	5	2.775	1.48
airbag system	3	2.885	1.58

Table 2: Results for the case study: *feas* shows the end-to-end delays when a solution is obtained that satisfies all constraints, *opt* shows the optimized delays when the objective function from Eq. (36) is used.

For $\tau > maxdelay_F$, the resulting dynamics (35) fails to meet stability requirement of the control functions.
Results: The first four functions (marked with * in Table 1) were already mapped to the system and the schedule had to be extended incrementally for mapping the other four functions. The proposed methodology is capable of extending a schedule incrementally by setting the start times of the tasks in the ILP. The resulting ILP formulation consists of 23437 variables and 60092 constraints.

The time required to find a feasible schedule was 2.91 seconds. This is a very reasonable runtime and provides a solution to the designer almost instantaneously. The results are given in Table 2 in the column *feas*. The resulting solution is tested for plausibility and successfully synthesized using SIMTOOLS and a model-based design flow, see [14].
Optimization Objective: Using an ILP also allows to define a linear objective function. In general, arbitrary linear objective functions might be applied such as minimizing the used FlexRay slots. The consideration of different objective functions is not in the scope of this paper and hence as an example, a minimization of the sum of the end-to-end delays of the considered functions is used:

$$min : \sum_{F \in \mathcal{F}} \mathbf{delay_F} \tag{36}$$

The runtime of this approach is 542 seconds with the resulting end-to-end delays as given in the Table 2 in the column *opt*. To reduce this runtime, the user might use an incremental design as is common in the automotive domain, i.e., scheduling some of the functions sequentially.

4.2 Scalability

Finally, experiments were carried out to show the scalability of the proposed approach. For this purpose, synthetic applications with 1 to 16 functions were generated and mapped onto a set of ECUs. The number of ECUs were determined as $\lfloor |T|/25 \rfloor + 3$ (at least three ECUs and for each 25 tasks one additional ECU). The execution time for each process was between 0.1ms and 2.0ms while the periods of the functions were either 5ms, 10ms, or 20ms. For each application size, 40 different applications were generated to obtain a significant mean value.

The results of the scalability analysis are illustrated in Figure 7. The results show that scheduling up to 100 processes and messages at the same time is possible in a reasonable amount of time. In practice, this value is enough for the automotive domain since usually an incremental design is carried out. The results of the scalability analysis ensure that even a much higher number of processes and messages can be scheduled with the proposed approach than in the presented case study.

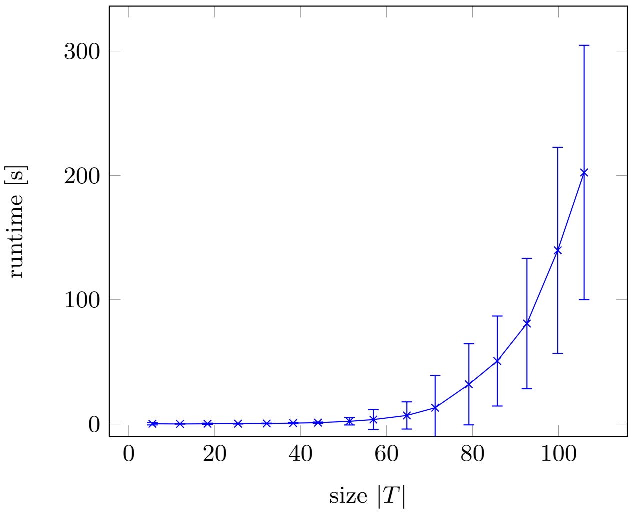


Figure 7: Results of the scalability analysis: Number of processes and messages ($|T| = |T_p| + |T_m|$) vs. runtime in seconds. The vertical error bars show the deviation.

5. CONCLUDING REMARKS

This paper proposed a framework for scheduling time-triggered automotive systems. Using a modular ILP formulation, both, different bus access schemes and operating system schedules were compositionally modeled. In particular, FlexRay 2.1 and 3.0, as well as non-preemptive eCos-based and the OSEKtime operating systems were modeled. The proposed framework has several advantages compared to known approaches – the modeling of each component is done separately, an incremental scheduling of legacy systems is possible, and there are no restrictions on the maximal end-to-end delays or spanning of the application over the communication cycle. A case study gave evidence of the effectiveness of the proposed approach. The scalability of the proposed approach was also studied on a set of synthetic test cases. It was shown that the approach is applicable to large instances and might be used in an incremental manner for an entire system, thereby enabling an efficient design of fully time-triggered automotive systems.

6. REFERENCES

[1] AUTOSAR. Specification of the FlexRay Interface Version 3.0.2, 2008. http://www.autosar.org.
[2] G. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1998.
[3] S. Ding, N. Murakami, H. Tomiyama, and H. Takada. A GA-based Scheduling Method for FlexRay Systems. In *Proc. of EMSOFT 2005*, pages 110–113, 2005.
[4] eCos. embedded Configurable operating system. http://ecos.sourceware.org/.
[5] FlexRay Consortium. FlexRay Communications Systems - Protocol Specification. http://www.flexray.com.
[6] M. Grenier, L. Havet, and N. Navet. Configuring the Communication on FlexRay: The Case of the Static Segment. In *Proc. of ERTS 2008*, 2008.
[7] E. Guran Schmidt and K. Schmidt. Message Scheduling for the FlexRay Protocol: The Dynamic Segment. *IEEE Transactions on Vehicular Technology*, 58(5):2160–2169, 2009.
[8] A. Hagiescu, U. D. Bordoloi, S. Chakraborty, P. Sampath, P. V. V. Ganesan, and S. Ramesh. Performance Analysis of FlexRay-based ECU Networks. In *Proc. of DAC 2007*, pages 284–289, 2007.
[9] IBM. ILOG CPLEX. http://www.ibm.com/software/, Version 12.2.
[10] M. Lukasiewycz, M. Glaß, P. Milbredt, and J. Teich. FlexRay Schedule Optimization of the Static Segment. In *Proc. of CODES+ISSS 2009*, pages 363–372, 2009.
[11] OSEK/VDX. Time Triggered Operating System. http://www.osek-vdx.org/.
[12] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing Analysis of the FlexRay Communication Protocol. *Real-Time Systems*, 39(1):205–235, 2008.
[13] K. Schmidt and E. Guran Schmidt. Message Scheduling for the FlexRay Protocol: The Static Segment. *IEEE Transactions on Vehicular Technology*, 58(5):2170–2179, 2009.
[14] SIMTOOLS. Model-Based Design Tools for FlexRay-based Applications. http://www.simtools.at/.
[15] P. Spindler. Automotive Networking Protocol Overview, 2010. Freescale Semiconductor presentation.
[16] H. Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli. Schedule Optimization of Time-Triggered Systems Communicating Over the FlexRay Static Segment. *IEEE Transactions on Industrial Informatics*, 7(1):1–17, 2011.
[17] H. Zeng, W. Zhengzheng, M. Di Natale, A. Ghosal, P. Giusto, and A. Sangiovanni-Vincentelli. Scheduling the flexray bus using optimization techniques. In *Proc. of DAC 2009*, pages 874–877, 2009.