Concurrent Architecture and Schedule Optimization of Time-triggered Automotive Systems

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

Samarjit Chakraborty
TU Munich
Munich, Germany
samarjit.chakraborty@rcs.ei.tum.de

ABSTRACT

This paper presents a methodology for the concurrent optimization of the architecture and scheduling of upcoming synchronous time-triggered automotive systems. A fully synchronous time-triggered system is highly predictable and therefore the best candidate for safety and drive-by-wire functions with strict real-time constraints. While the architecture of these systems has to be optimized in terms of resource allocation, task mapping, and message routing by taking multiple conflicting objectives into account, the scheduling has to be carried out such that application deadlines are satisfied. In case of stringent real-time constraints, available approaches that address architecture optimization and scheduling as separate problems become inapplicable as most architectures do not permit a feasible schedule. As a remedy, a novel and efficient approach based on conflict refinement is presented. For a given architecture, either a schedule might be obtained or a conflict refinement is performed to determine and exclude the architecture decision that prevents a feasible schedule. In this paper, an extended architecture model is presented and the scheduling and refinement approaches are given for time-triggered architectures based on the FlexRay protocol. This approach can be extended to other protocols and scenarios. A case study of a realistic time-triggered system gives evidence of the efficiency of the proposed approach, solving a large design problem from the automotive domain.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems

General Terms

Algorithms, Design

Keywords

Architectural Design Space Exploration, Synchronous Scheduling, Conflict Refinement

This publication is made possible by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme.

CODES+ISSS'12, October 7-12, 2012, Tampere, Finland.

1. INTRODUCTION

Modern automotive systems consist of dozens of Electronic Control Units (ECUs) interconnected by a set of different buses. With a growing degree of freedom introduced by software component concepts like AUTomotive Open System ARchitecture (AUTOSAR) [1], the optimization of these automotive systems becomes a challenging task due to the very large design space. Often multiple conflicting objectives have to be considered like monetary costs, reliability, or energy consumption. The goal of the architectural optimization is the determination of the resource allocation and configuration, task mapping, and message routing for the entire system. Finally, an appropriate scheduling has to be applied such that each application satisfies its stringent deadline constraints.

Since the first broad introduction of automotive electronics in the 1980s, many automotive systems are based on event-triggered systems. These systems are using priority-based operating systems like OSEK [17] and the priority-based Control Area Network (CAN) bus [2] for the communication. Event-triggered systems have the advantage of being very flexible and easily extensible with the goal of a very high utilization. This is a reasonable approach in case the computational power is a restrictive factor and the number of functions that require very short execution and communication latencies is low. In case functions have stringent latency and jitter constraints, these systems require often a complex validation of end-to-end timing behavior using tools like Symbolic Timing Analysis for Systems (SymTA/S) [20] or Modular Performance Analysis (MPA) [3]. This is becoming a challenging design task and often leads to high over-approximations which are not acceptable.

With a growing number of passive and active safety functions in the automotive domain as well as the introduction of drive-by-wire functions for electric vehicles, predictability of the entire system becomes a major design goal. Therefore, the design paradigm in the automotive domain is changing towards synchronous time-triggered systems. The main component for such a system is a communication bus like FlexRay [7] or Precision Time Protocol (PTP) Ethernet [21] that allows a synchronization of the ECUs. Additionally, the ECUs require a time-triggered operating system that is either non-preemptive using for instance eCos [5] or preemptive using operating systems like OSEKtime [17]. These time-triggered systems enable the design of predictable systems with guaranteed end-to-end latencies. Thus, very short end-to-end latencies can be achieved with no jitter. This enables the design of very robust applications and control func-

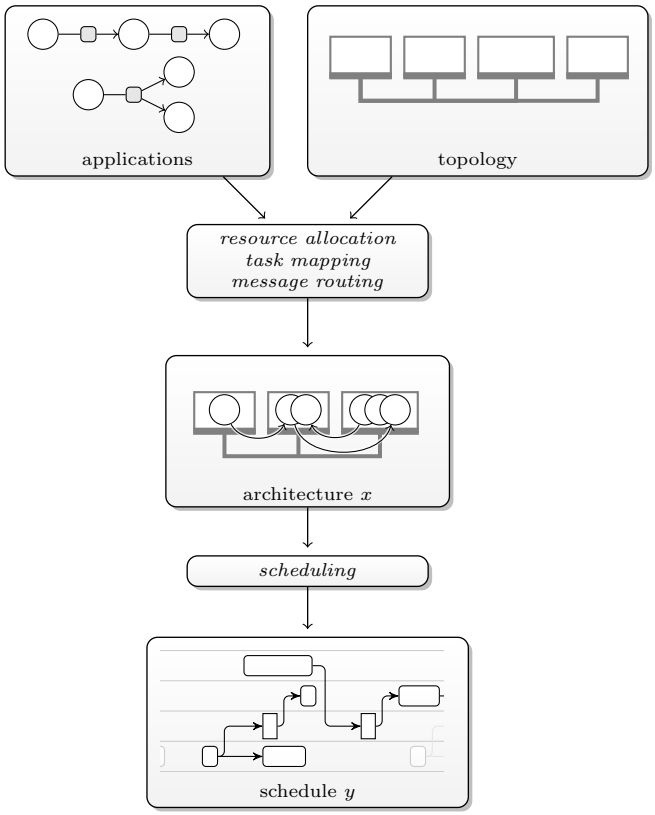

Figure 1: Y-chart approach for the design of time-triggered automotive systems. An architecture x is obtained from the mapping of the applications to a topology of hardware resources. In a second step, a scheduling has to be performed to obtain a schedule y.

tions without any oversampling which is often necessary for event-triggered systems. Besides a significant reduction of the computational demands, this enables the implementation of functions with stringent real-time constraints.

In order to enable a full optimization of time-triggered systems at system-level, both the architecture optimization and scheduling have to be performed. A general y-chart approach is illustrated in Figure 1. Here, first an architectural exploration is carried out that maps the applications to a hardware topology, followed by a scheduling that determines a feasible schedule. In particular for time-triggered systems, the scheduling is a complex design task that has to determine offset times for processes on ECUs and appropriate bus schedules. For complex systems not every architecture permits a feasible schedule. Therefore, a concurrent optimization of the architecture and scheduling becomes necessary. In this paper, this methodology is presented for the first time to perform this complex optimization efficiently within a proposed framework.

Contributions of the paper. This paper proposes a methodology for the optimization of time-triggered automotive systems, taking architecture and scheduling decisions *concurrently* into account. To cope with the inherent problem complexity, the architecture optimization and schedule determination are separated into two coupled tasks. In case no feasible schedule exists for a given architecture, a conflict refinement [9, 16] and justification deduction are performed. This approach effectively prunes the search space and enables an efficient optimization, considering multiple objectives. In case the objectives are linear, an iterative optimization is performed until the optimal system architectures with feasible schedules are obtained. In case at least one objective is non-linear, an iterative heuristic approach is suggested to enable an efficient optimization. In this paper, the methodology is applied to time-triggered systems in the automotive domain that are based on the FlexRay [7] protocol. The methodology might be extended to support other protocols and might be also applied in other domains like avionics or automation technology.

Besides the novel approach to use conflict refinement for concurrent architecture and scheduling optimization, the contributions are the following:

- An extended architecture model is presented that takes resource configuration, resource utilization, and path delays concurrently into account. This enables a very efficient exploration by pruning already a large fraction of the search space.
- For the scheduling, an Integer Linear Programming (ILP) formulation is defined to determine the FlexRay bus scheduling and offsets for the non-preemptive task scheduling on an eCos-based [5] operating system. Here, a hierarchical approach for each cluster of applications is proposed that significantly improves the scalability of the synchronous scheduling.
- A group-based conflict refinement strategy is proposed that takes domain knowledge of the architecture model into account. This enables a very efficient determination of architecture decisions that prevent a feasible schedule.

Finally, the methodology is applied to a large case study from the automotive domain. A realistic case study of an automotive subsystem is presented to give evidence of the efficient multi-objective optimization in terms of RAM size and flash memory size of program code.

Organization of the paper. The remainder of the paper is organized as follows: Section 2 discusses related work. Section 3 introduces the optimization scheme, using conflict refinement. In Section 4, the architecture model is presented and extended by additional constraints to restrict the search space. In Section 5, the scheduling model for time-triggered systems and a hierarchal scheduling approach are proposed. Moreover, a justification scheme and a problem-specific group-based refinement strategy are presented. Section 6 presents experimental results, before the paper is concluded in Section 7.

2. RELATED WORK

In embedded system design, many optimization problems are highly complex such that multi-step optimization approaches are applied, see [8, 15]. In a combined architecture and scheduling optimization, first an optimal architecture is obtained followed by a scheduling. For the event-triggered scheduling, heuristic strategies for priority optimization [11] and exact approaches for period optimization [4] exist. For time-triggered systems, a synchronous scheduling [22] of all components has to be obtained which might be a very complex design task. In case a feasible schedule cannot be

obtained, a different architecture has to be found, making the optimization approach very time-consuming and unpredictable if stringent real-time constraints are existent. In [18, 19], a strategy is proposed that combines the architecture and scheduling model for event-triggered systems by a pruning approach based on SMT solvers. This approach finds a feasible system if one exists, but is not used for the optimization in terms of system objectives. Moreover, it is restricted to event-triggered systems which can be expressed by a binary formulation. In contrast, the proposed approach in this paper is flexible in terms of the scheduling and is highly scalable by separating the optimization problems into coupled tasks using a conflict refinement feedback. Thus, a time-triggered scheduling within an architectural optimization becomes efficiently applicable for the first time.

The proposed work is based on previous work in [14] and [13]. The work in [14] presents the basic model for the architecture exploration. In the work at hand, this model is appropriately adapted and extended by additional configuration constraints, utilization constraints, and path delay constraints that help to prune a large fraction of the search space. The work in [13] presents a basic scheduling framework for time-triggered automotive systems. In this paper, it is extended by a proposed hierarchical scheduling scheme to improve the scalability significantly. Moreover, a problem-specific conflict refinement and justification deduction for the scheduling are introduced to enable the concurrent architecture and scheduling optimization.

3. OPTIMIZATION SCHEME

In the following, the optimization problem for systems that require a concurrent architectural optimization and scheduling is defined. An efficient optimization flow that separates the architecture optimization and scheduling with an appropriate feedback method is proposed. This optimization is using conflict refinement which constitutes a novel approach for multi-layered design space exploration problems that consider architecture optimization and scheduling concurrently.

3.1 Problem Definition

The concurrent optimization of the architecture and scheduling is a challenging *design space exploration* task. While the architectural optimization comprises the optimization of the resource allocation and configuration, task mapping, and message routing, the scheduling has to be performed such that all applications satisfy their end-to-end latency constraints. For time-triggered systems, the scheduling is defined by process task offsets and appropriate bus schedules. The determination of these offsets and schedules is a complex design task due to the large search space.

The design space exploration task is formulated as the following multi-objective optimization problem:

DEFINITION 1 (DESIGN SPACE EXPLORATION).
optimize $f(x)$
subject to:
x is a feasible *architecture*
there exists a feasible *schedule* y for *architecture* x

In real-world problems, the objective function f consists of multiple functions (sometimes also including non-linear calculations). In general, the principal design objectives depend on the architecture x while the schedule y has to be determined such that all deadline constraints are satisfied. The proposed optimization scheme in this paper allows a primary optimization of the objectives f in terms of the architecture x and for each x a secondary optimization of the schedule y in terms of a single objective, e.g., the minimization of the end-to-end application latencies or the minimization of used communication slots.

3.2 Optimization Flow

The proposed optimization scheme is based on the separation of the architecture optimization problem C and scheduling problem C_x for each architecture x. An efficient feedback between the scheduling and architecture optimization is incorporated by using *conflict refinement* and *justification deduction* when no feasible schedule exists for a given architecture x. This avoids the formulation of a large common optimization problem that comprises both the architecture and scheduling problem which becomes too complex to be solved efficiently.

The proposed optimization flow is illustrated in Figure 2. First, an architecture x is determined from an architecture specification that is formulated as a set of constraints C. If no feasible architecture x exists (denoted as $\forall x : x \nvDash C$)[1], the optimization is stopped, otherwise the optimization is continued. Based on a feasible architecture x (denoted as $x \vDash C$), a scheduling problem C_x is defined that is specifically formulated for the architecture x. The scheduling problem C_x is used to determine a schedule y.

In case no feasible schedule y exists (denoted as $\forall y : y \nvDash C_x$), a conflict refinement on C_x is carried out to determine the reason for the conflict, i.e., a subset of constraints $\tilde{C}_x \subseteq C_x$. This reduced set of constraints $\tilde{C}_x$ is known as Irreducible Infeasible Subsets (IIS) [9] or Minimally Unsatisfiable Subformula (MUS) [16], respectively. For all $\tilde{c} \in \tilde{C}_x$, the reduced set of constraints $\tilde{C}_x \backslash \{\tilde{c}\}$ is satisfiable while $\tilde{C}_x$ is unsatisfiable. A conflict refinement might be done using several techniques, see [9, 16], and is part of each modern ILP solver [10].

Based on $\tilde{C}_x$, a justification c (an additional constraint) for the architecture constraints C is determined and added to the architecture model. The justification ensures that the architecture x as well as all other architectures that contain the design decisions that prohibit a feasible schedule are pruned in the ongoing optimization. Since the conflict refinement finds a minimal reason for the deadline violation in the scheduling model C_x, often a large part of the search space C might be pruned by adding the constraint c.

In case a feasible architecture x and a feasible schedule y are found, an appropriate optimization strategy is carried out. Note that determining the schedule y can also be defined as an optimization problem such that for a given architecture x the optimal schedule y is determined. If the optimization problem is linear, i.e., all objective functions are linear, an exclusion of all sub-optimal solutions is done as follows:

$$C = C \cup \bigcup_i (f_i(x) < v_i + M \cdot \mathbf{z_i}) \cup \sum_i (\mathbf{z_i} \leq n - 1) \quad (1)$$

This additional constraint ensures that dominated solutions are excluded from the search space. Here, v_i is the

[1] $x \vDash C$ is read as x *satisfies* C. $x \nvDash C$ is read as x *does not satisfy* C.

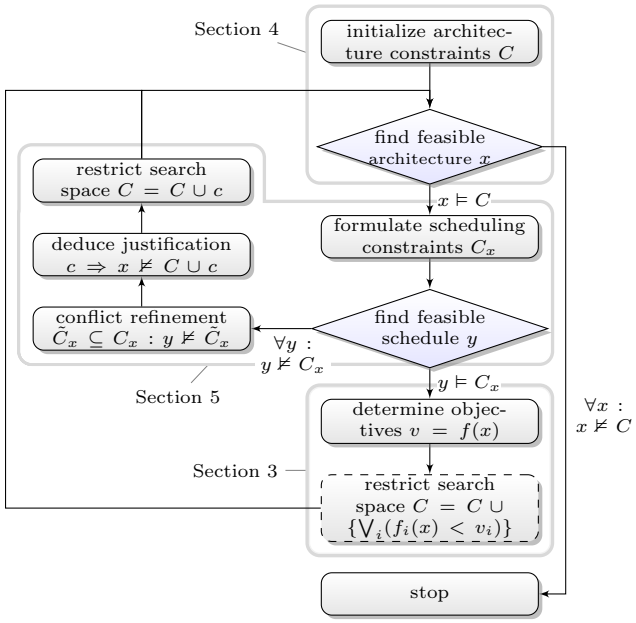

Figure 2: The proposed optimization flow. An architecture x and schedule y are determined separately. In case a feasible schedule y exists, an optimization is performed. Otherwise a conflict refinement is carried out and the search space for the architecture is efficiently pruned. The figure denotes in which sections the specific parts of the optimization flow are discussed in this paper.

objective value of the current objective value determined by f_i for an architecture x. The switch variables $\mathbf{z_i}$ ensure that at least one of n objectives is better than the value of the current obtained architecture x. Here, the constant M represents a very large number such that all constraints except one might be relaxed.

In case the problem is non-linear, a heuristic search as presented in [12] might be applied to optimize the architecture x. This is achieved by an appropriate randomized variation of x, guided by a heuristic to optimize multiple objectives.

4. ARCHITECTURE MODEL

In the following, the model for the architecture exploration and an encoding of the architecture constraints C are presented. First, the exploration model is introduced and the basic constraints are presented. In order to enable different configuration of ECUs, additional configuration parameters are introduced. Moreover, to exclude a large fraction of architectures that do not permit a feasible schedule from the search space, additional utilization constraints and path delay constraints are proposed.

4.1 Exploration Model

The architectural exploration is defined by a *specification*, consisting of a set of *applications* and the target *topology*. From this specification, an architecture x is defined by the allocation of target resources, mapping of process tasks, and routing of messages. An example of a specification and architecture x is given in Figure 3.

In the following, the exploration model is defined formally. The specification consists of a *topology graph* G_R and an *application graph* G_T :

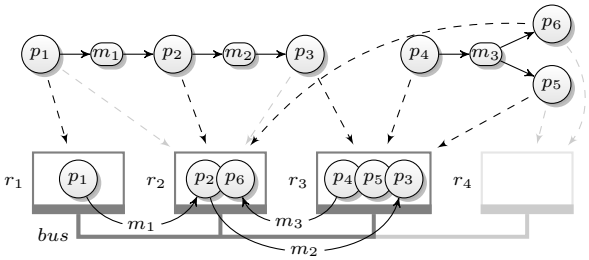

Figure 3: Illustration of a architecture specification and one possible architecture implementation. In the implementation, two applications are mapped to a topology with three allocated ECUs.

- The topology graph $G_R(R, E_R)$ consists of a set of resources R such as ECUs, buses, or gateways. The undirected edges E_R indicate that two resources are connected, i.e., a communication is possible.
- The application graph $G_T(T, E_T)$ is a directed acyclic graph and consists of a set of tasks $T = T_p \cup T_m$ which are either processes (T_p) or messages (T_m). The directed edges in E_T depict data-dependencies, i.e., sending of messages or receiving of messages, respectively. For each $t \in T$, the resources $R_t \subseteq R$ define the possible target resources of task t, i.e., where t might be mapped to or routed, respectively.

Given the exploration model, an architecture x is defined by the allocation of resources in the topology graph and the mapping of the application graph by an appropriate selection of target resources for each task:

- The allocation is an induced subgraph $G_R^x \subseteq G_R$ that consists of all resources that are used in the architecture implementation.
- The application graph is mapped to the allocated resources as follows: Each process task $t \in T_p$ is mapped to exactly one resource $R_t^x \in R_t$. Each message $t \in T_m$ is mapped to a routing tree on the set of allocated resources $R_t^x \subseteq R_t$. The mapping of processes and routing of messages has to be carried out such that all data-dependencies are satisfied, i.e., sent messages start at the resource of the sending task and received messages pass the resource of the receiving tasks, respectively.

4.2 Encoding C

Basic constraints. In the following, a set of linear constraints C is defined such that a solution $x \vDash C$ corresponds to a *feasible* architecture, i.e, the allocation of resources, the mapping of processes, and the routing of messages fulfill the data-dependencies. The symbolic encoding comprises the following binary variables:

- $\mathbf{r} \in \{0, 1\}$ - 1 if resource $r \in R$ is allocated, 0 otherwise
- $\mathbf{t_r} \in \{0, 1\}$ - 1 if task $t \in T$ and is mapped to resource $r \in R_t$, 0 otherwise
- $\mathbf{t_{r,n}} \in \{0, 1\}$ - 1 if message $t \in T_m$ is routed on resource $r \in R_t$ in communication step $n \in \mathbb{N}$, 0 otherwise

The linear constraints are formulated as follows:
$\forall t \in T_p$:

$$\sum_{r \in R_p} \mathbf{t_r} = 1 \tag{2}$$

$\forall t \in T_m$:

$$\sum_{r \in R_t} \mathbf{t_{r,0}} = 1 \tag{3}$$

$\forall t \in T_m, \tilde{t}$ with $(\tilde{t}, t) \in E_T, r \in R_t$:

$$\mathbf{t_r} - \tilde{\mathbf{t}}_{\mathbf{r,0}} = 0 \tag{4}$$

$\forall t \in T_p, \tilde{t}$ with $(\tilde{t}, t) \in E_T, r \in R_t$:

$$\tilde{\mathbf{t}}_{\mathbf{r}} - \mathbf{t_r} \geq 0 \tag{5}$$

$\forall t \in T_m, r \in R_t$:

$$\sum_{i=1}^{n} \mathbf{t_{r,i}} \leq 1 \tag{6}$$

$$\sum_{i=1}^{n} \mathbf{t_{r,i}} - \mathbf{t_r} \geq 0 \tag{7}$$

$\forall t \in T_m, r \in R_t, i = \{1, ..., n\}$:

$$\mathbf{t_r} - \mathbf{t_{r,i}} \geq 0 \tag{8}$$

$\forall t \in T_m, r \in R_t, i = \{1, ..., n-1\}$:

$$-\mathbf{t_{r,i+1}} + \sum_{\tilde{r} \in R_t \wedge e=(\tilde{r},r) \in E_R} \mathbf{t_{\tilde{r},i}} \geq 0 \tag{9}$$

$\forall t \in T, r \in R_t$:

$$\mathbf{r} - \mathbf{t_r} \geq 0 \tag{10}$$

$\forall r \in R$:

$$-\mathbf{r} + \sum_{t \in T \wedge r \in R_t} \mathbf{t_r} \geq 0 \tag{11}$$

Constraint (2) ensures that each process task is mapped to exactly one resource. Constraints (3) and (4) imply that each message has exactly one source that equals the used resource of the preceding process. Analogously for each process, the received messages have to be routed on the corresponding resources as stated in Constraint (5). Constraint (6) ensures that a message can pass a resource at most once such that no routing cycles exist. A message has to be existent in one communication step on a resource in order to be correctly routed on this resource as implied by Constraints (7) and (8). Constraint (9) states that a communication is only possible between adjacent resources. Constraint (10) implies that processes or messages are using allocated resources only. On the other hand, Constraint (11) states that a resource is only allocated if at least one task is mapped to or routed on this resource.

Configuration constraints. The architecture exploration might be extended by additional parameters. In our experimental results, the resources can have different configurations (V_r) in terms of available flash memory and RAM. For this purpose, binary variables $\mathbf{r_v} \in \{0, 1\}$ are introduced to indicate whether resource r is in configuration $v \in V_r$. Additionally, a load function l and capacity function c for each dimension is defined. Here, $l : T \times R \to \mathbb{R}$ defines the load that is generated by task t on resource r and $c : R \times V_r \to \mathbb{R}$ defines the capacity of the resource r in configuration v.
$\forall r \in R$:

$$\sum_{t \in T} l(t, r) \cdot \mathbf{t_r} \leq \sum_{v \in V_r} c(r, v) \cdot \mathbf{r_v} \tag{12}$$

$$\sum_{v \in V_r} \mathbf{r_v} = 1 \tag{13}$$

Constraint (12) defines the capacity constraint. Constraint (13) states that each resource is in exactly one configuration state.

Utilization constraints. In order to avoid unsatisfiable schedules if the load on a resource is above 1, the following constraint is added:
$\forall r \in R$:

$$\sum_{t \in T} \frac{e_t(r)}{h_t} \cdot \mathbf{t_r} \leq 1 \tag{14}$$

This constraint ensures that the sum of processing load from each process t on a resource r is below 1. Here, $e_t(r)$ is the worst-case execution time of task t on resource r and h_t is the period of task t.

Path delay constraints. It is further necessary to exclude those architectures where the execution time of tasks along a path $\pi \in \Pi$ in the application graph G_T exceeds the deadline $\tau_\pi \in \mathbb{R}$. Since the introduced model allows multi-cast communication, a consideration of each separate communication path becomes necessary. For this purpose, the introduction of the following additional variables is required:

- $\mathbf{t_r^{\tilde{t}}} \in \{0, 1\}$ - 1 if message $t \in T_m$ and is routed to the task $\tilde{t} \in T_p$ over the resource $r \in R_t$, 0 otherwise
- $\mathbf{t_{r,n}^{\tilde{t}}} \in \{0, 1\}$ - 1 if message $t \in T_m$ is routed to the task $\tilde{t} \in T_p$ on resource $r \in R_t$ in communication step $n \in \mathbb{N}$, 0 otherwise

The constraints are defined as follows:
$\forall t \in T_m, (t, \tilde{t}) \in E_T, r \in R_t$:

$$\mathbf{t_{r,0}} = \mathbf{t_{r,0}^{\tilde{t}}} \tag{15}$$

$\forall t \in T_m, (t, \tilde{t}) \in E_T, i = \{1, ..., n\}$:

$$\sum_{r \in R_t} \mathbf{t_{r,i}^{\tilde{t}}} \leq 1 \tag{16}$$

$\forall t \in T_m, (t, \tilde{t}) \in E_T, r \in R_t, i = \{1, ..., n\}$:

$$\mathbf{t_r^{\tilde{t}}} - \mathbf{t_{r,i}^{\tilde{t}}} \geq 0 \tag{17}$$

$$\mathbf{t_{r,i}} - \mathbf{t_{r,i}^{\tilde{t}}} \geq 0 \tag{18}$$

$\forall t \in T_m, (t, \tilde{t}) \in E_T, r \in R_t, i = \{1, ..., n-1\}$:

$$\sum_{\tilde{r} \in R_t, (r,\tilde{r}) \in E_R} \mathbf{t_{\tilde{r},i+1}^{\tilde{t}}} + \tilde{\mathbf{t}}_{\mathbf{r}} - \mathbf{t_{r,i}^{\tilde{t}}} \geq 0 \tag{19}$$

The Constraints (15)-(19) define the routing of messages $t \in T_m$ from the sender to a dedicated receiver task $t \in T_p$ by setting the $\mathbf{t_r^{\tilde{t}}}$ variables. Constraint (15) defines that each separate routing path starts at the same source resource as the routing of the corresponding message. Constraint (16) ensures that the routing path from t to $\tilde{t}$ is a path, i.e., not a tree. Here, a message is routed at most on one single resource at a communication step. Constraint (17) sets the $\mathbf{t_r^{\tilde{t}}}$ variables while Constraint (18) ensures that the separate routing paths are only mapped on the original routings of the messages. Constraint (19) states that a message is either forwarded to an adjacent resource or it is terminated at the target resource of the receiver task $\tilde{t}$.

With the additional variables, each path of the application is constrained as follows:

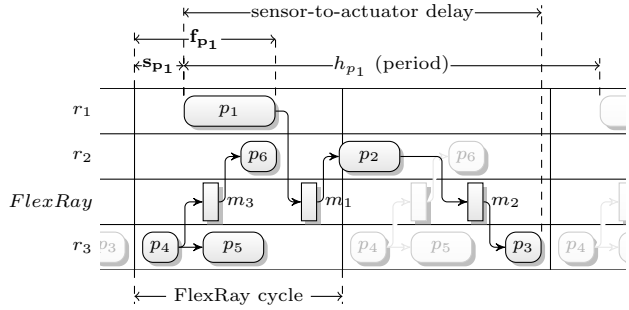

Figure 4: Illustration of a time-triggered schedule for the architecture in Figure 3. For each task, a start and finish time has to be defined. The tasks have to be executed with their period and end-to-end deadlines constraints have to be satisfied.

$\forall \pi \in \Pi$:

$$\sum_{t \in \pi \cap T_p, r \in R_t} e_t(r) \cdot \mathbf{t_r} + \sum_{(t,\tilde{t}) \in \pi, t \in T_m, r \in R_t} e_t(r) \cdot \mathbf{t_r^{\tilde{t}}} \leq \tau_\pi \quad (20)$$

Here, each process task in T_p and each message in T_m along the path π is considered. $e_t(r)$ is the execution time or transmission time, respectively, of a task t on the resource r. Thus, all architectures are excluded where the execution times along a path exceed the maximal delay.

5. SCHEDULING MODEL

In the following, the model for the scheduling is presented. Given an architecture x, the encoding of C_x is proposed for time-triggered systems using a synchronization via the FlexRay bus. Moreover, a hierarchical approach is proposed that improves the runtime of the conflict refinement. Finally, the justification deduction from a unsatisfiable scheduling problem is presented that prunes the infeasible search space of the architectural optimization problem. For this purpose, a fast conflict refinement strategy is proposed, using an elastic and deletion filter with domain knowledge to group the constraints that affect same tasks.

Many real-world applications have strict end-to-end timing delays. In this paper, it is assumed that each application and task $t \in T$ is executed with a fixed period $h_t \in \mathbb{R}$. Moreover, for each task t, a worst-case execution time e_t is given for the current task mapping. These upper bounds for execution times might be estimated using approaches like presented in [6]. A schedule has to be obtained that executes the tasks with the given period and satisfies all deadline constraints. The end-to-end timing constraints might be defined on each path $\pi \in \Pi$ in the application graph G_T by a deadline $\tau_\pi \in \mathbb{R}$.

5.1 Time-triggered Scheduling C_x

The time-triggered scheduling in this paper is assuming that all ECUs are using a non-preemptive operating system as well as the FlexRay static segment for the communication. The schedule y is determined by the offset/start time of each task in the system. For the architecture in Figure 3, an appropriate scheduling is illustrated in Figure 4.

For the determination of a feasible schedule y that considers all platform constraints, we propose an ILP formulation that determines the following variables:

- $\mathbf{s_t} \in \mathbb{R}_{[0,h_t]}$ - start time of task $t \in T$
- $\mathbf{f_t} \in \mathbb{R}_{[0,h_t]}$ - finish time of task $t \in T$

In the following the separate ILP formulation for the components are presented. First, the scheduling constraints of the ECUs using an eCos-based operating system without preemption are defined. The static segment of a FlexRay bus is used for the bus system. Finally, the end-to-end timing constraints are defined.

ECU Scheduling. The tasks on each ECU have to be scheduled in compliance with the used operating system scheduler. Here, the processes $T_r = \{t | r \in R_t^x, t \in T_p\}$ are scheduled on each ECU where $r \in R_t^x$ determines the target resource for each process task in the architecture x.

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

- $\mathbf{of_t} \in \{0,1\}$ - offset for finish time $t \in T_p$
- $\mathbf{y_{t,\tilde{t}}^{i,j}} \in \{0,1\}$ - 1 if job i of task $t \in T_p$ finished before job j of task $\tilde{t} \in T_p$

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

$$\mathbf{f_t} + h_t \cdot \mathbf{of_t} = \mathbf{s_t} + e_t \quad (21)$$

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

$$i \cdot h_t + \mathbf{s_t} + e_t \leq j \cdot h_{\tilde{t}} + \mathbf{s_{\tilde{t}}} + 2 \cdot H_r \cdot (1 - \mathbf{y_{t,\tilde{t}}^{i,j}}) \quad (22)$$

$$j \cdot h_{\tilde{t}} + \mathbf{s_{\tilde{t}}} + e_{\tilde{t}} \leq i \cdot h_t + \mathbf{s_t} + 2 \cdot H_r \cdot \mathbf{y_{t,\tilde{t}}^{i,j}} \quad (23)$$

Constraint (21) determines the finish time of each task. In case the process finishes in the next cycle, the finish time is smaller than the start time. In this case, the variable $\mathbf{of_t}$ becomes 1 to fulfill Constraint (21). Constraints (22) and (23) ensure that two tasks never preempt each other within two hyper-periods ($2 \cdot H_r$ with $H_r = \underset{t \in T_r}{lcm}(h_t)$). The variable $\mathbf{y_{t,\tilde{t}}^{i,j}}$ is used for switching, i.e., one of the Constraints (22) or (23) is trivially satisfied depending on $\mathbf{y_{t,\tilde{t}}^{i,j}}$.

FlexRay Scheduling. The FlexRay bus is used as a central communication bus in the proposed architecture. This bus system enables a synchronization of the ECUs and, thus, a fully time-triggered system. For message scheduling, the static segment of FlexRay is used. The static segment of FlexRay is organized in $n_{fx} \in \mathbb{N}$ static slots with each slot having a duration of $e_{fx} = e_t \in \mathbb{R}$ and available payload of $l_{fx} \in \mathbb{N}$ bytes. The static segment is transmitted in the beginning of each FlexRay cycle which has a duration of $h_{fx} \in \mathbb{R}$. A message is defined by the repetition that is deduced from the period $r_t = \frac{h_t}{h_{fx}} \in \mathbb{N}$, the execution time that equals the slot duration, and the size in bytes $l_t \in \mathbb{N}$.

For each message, a slot and base cycle have to be determined. From this value, the assignments of slots to ECUs is deduced implicitly. The slot and base cycle are encoded in the binary variable $[\mathbf{s}, \mathbf{b}]_\mathbf{t}$ such that the following variables are required:

- $[\mathbf{s}, \mathbf{b}]_\mathbf{t} \in \{0,1\}$ - 1 if message $t \in T_m$ is sent in slot $s \in \{0, .., n_{fx} - 1\}$ at base cycle $b \in \{0, .., r_t - 1\}$, 0 otherwise
- $\mathbf{s_r} \in \{0,1\}$ - becomes 1 if slot s is assigned to ECU r and 0 otherwise

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

$$\sum_{s\in\{0,..,n_{fx}-1\}} \sum_{b\in\{0,..,r_t-1\}} [\mathbf{s},\mathbf{b}]_\mathbf{t} = 1 \quad (24)$$

$$\mathbf{s_t} = \sum_{s\in\{0,..,n_{fx}-1\}} \sum_{b\in\{0,..,r_t-1\}} (s\cdot e_{fx} + b\cdot h_{fx})\cdot [\mathbf{s},\mathbf{b}]_\mathbf{t} \quad (25)$$

$$\mathbf{f_t} = \mathbf{s_t} + e_{fx} \quad (26)$$

$\forall s \in \{0,..,n_{fx}-1\}, b \in \{0,..,\frac{\frac{lcm_{t\in T_m}(h_t)}{h_{fx}}}{h_{fx}} - 1\}$:

$$\sum_{t\in T_m} l_t \cdot [\mathbf{s},\mathbf{b}\%\mathbf{r_t}]_\mathbf{t} \le l_{fx} \quad (27)$$

$\forall s \in \{0,..,n_{fx}-1\}$:

$$\sum_{r\in R} \mathbf{s_r} \le 1 \quad (28)$$

$\forall t \in T_m, s \in \{0,..,n_{fx}-1\}, b \in \{0,..,r_t-1\}, r \in R_t^x$:

$$[\mathbf{s},\mathbf{b}]_\mathbf{t} \le \mathbf{s_r} \quad (29)$$

Constraint (24) states that each message is scheduled in exactly one slot at a specific base cycle. Constraint (25) determines the start time of a message transmission based on the slot and base cycle. Constraint (26) defines the finish time of a message transmission depending on the duration of the transmission of a static slot. Constraint (27) ensures that the capacity of a slot is not exceeded. Constraint (28) states that a slot can be assigned to at most one ECU. Constraint (29) ensures that if a message is sent in a specific slot, the slot is assigned to the sending ECU.

Sensor-to-Actuator Delay. 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 and transmission times, respectively, of all tasks as well as the waiting time between the data-dependent tasks, respectively. To constrain the end-to-end delay, the following variables are required:

- $\mathbf{w_{t,\tilde{t}}} \in \mathbb{R}_{[0,h_t]}$ - 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 constraints are defined as follows:
$\forall \pi \in \Pi, (t,\tilde{t}) \in \pi$:

$$\mathbf{w_{t,\tilde{t}}} = \mathbf{s_{\tilde{t}}} - \mathbf{f_t} + h_t \cdot \mathbf{ow_{t,\tilde{t}}} \quad (30)$$

$\forall \pi \in \Pi$:

$$\tau_\pi \ge \sum_{t\in\pi} e_t + \sum_{(t,\tilde{t})\in\pi} \mathbf{w_{t,\tilde{t}}} \quad (31)$$

Constraint (30) determines the waiting time between data-dependent tasks. The binary variable $\mathbf{ow_{t,\tilde{t}}}$ ensures that the waiting time is always within the predefined bounds, i.e., not negative. Constraint (31) 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.

5.2 Hierarchical Scheduling

To reduce the time to perform the scheduling, a hierarchical approach is proposed. Subproblems of the complete scheduling problem are solved iteratively to search for trivially unsatisfiable sub-formulas of C_x. For this purpose, domain knowledge is used as proposed in the following. If one of these subproblems is already unsatisfiable, the scheduling can be stopped and conflict refinement on the sub-formula can be performed more efficiently. The iterative search is defined as follows:

1. Schedule each ECU $r \in R$ separately.
2. Schedule each application (weakly connected component in the application graph G_T) separately.
3. Schedule each cluster (set of weakly connected components in the application graph G_T) separately.
4. Schedule the whole system.

A cluster $D \in \mathcal{D}$ defines a subset of tasks ($D \subseteq T$). For two clusters $D, \tilde{D} \in \mathcal{D}$, none of the process tasks from different clusters are mapped to the same ECU:

$$(\bigcup_{t\in D\cap T_p} R_t^x) \cap (\bigcup_{\tilde{t}\in \tilde{D}\cap T_p} R_{\tilde{t}}^x) = \{\} \quad (32)$$

Moreover, each cluster D defines a set of weakly connected components in the application graph G_T.

In order to minimize the time to obtain a schedule for an entire system, a combination of cluster schedules is proposed. This approach is applied if each cluster has a feasible schedule. By definition, the scheduling of each cluster on the ECUs does not influence the ECU scheduling in another cluster. Therefore, the task in combining cluster schedules is to find a feasible bus schedule. Each cluster D is using the the slots $S_D = \{s | t \in T_m, [\mathbf{s},\mathbf{b}]_\mathbf{t} = 1, s = \{0,..,n_{fx}-1\}, b = \{0,..,r_t-1\}\}$. In order to combine the schedules of all clusters, it has to be ensured that a slot is never used by two clusters. To avoid conflicting slots, the schedule of each cluster can be shifted by h_{fx}/e_{fx} steps of the size of one slot e_{fx}. Thus, if a schedule is shifted by i slots, the required slots are $S_{D,i} = \{(s+i)\%(h_{fx}/e_{fx}) | s \in S_D\}$. In the following, an ILP formulation is given that determines the shifting of each cluster that allows to combine cluster schedules. The ILP determines the binary offset variables:

- $\mathbf{o_{D,i}} \in \{0,1\}$ - 1 if cluster D is scheduled with offset $i \cdot e_{fx}$, 0 otherwise

The constraints are formulated as follows:
$\forall D \in \mathcal{D}$:

$$\sum_{i=0}^{\frac{h_{fx}}{e_{fx}}-1} \mathbf{o_{D,i}} = 1 \quad (33)$$

$\forall D, \tilde{D} \in \mathcal{D}, i,j \in \{0,..,\frac{h_{fx}}{e_{fx}}-1\}, S_{D,i} \cap S_{\tilde{D},j} \neq \{\}$:

$$\mathbf{o_{D,i}} + \mathbf{o_{\tilde{D},j}} \le 1 \quad (34)$$

$\forall D \in \mathcal{D}, i \in \{0,..,\frac{h_{fx}}{e_{fx}}-1\} \exists s \in S_{D,i} : s \ge n_{fx}$:

$$\mathbf{o_{D,i}} = 0 \quad (35)$$

Constraint (33) states that each cluster has exactly one offset. Constraint (34) ensures that offsets for two clusters are not possible if there is at least one common slot that would have to be equal for both clusters. Constraint (35) sets each offset to 0 for each cluster if a slot would not be mapped to the static segment which is in the beginning of the FlexRay cycle and has a duration of $n_{fx} \cdot e_{fx} \le h_{fx}$.

5.3 Justification Deduction

Given an unsatisfiable scheduling formula C_x, a conflict refinement is applied to determine the subset $\tilde{C}_x \subseteq C_x$. Using the hierarchical approach, the set of unsatisfiable scheduling constraints from C_x might be already significantly minimized. From the reduced scheduling formula $\tilde{C}_x$, a justification c (an additional constraint) for the architectural optimization problem C has to be deduced. This is done by analyzing which tasks are part of $\tilde{C}_x$. The tasks that are considered are defined as

$$T_c = \{t | t \in T, \tilde{c} \in \tilde{C}_x, \mathbf{s_t} \in \tilde{c}\}. \tag{36}$$

Thus, the additional constraint c that extends the architecture problem C is defined as follows:

$$\sum_{t \in T_c} \sum_{r \in R_t^x} (1 - \mathbf{t_r}) \geq 1 \tag{37}$$

This constraint ensures that at least one mapping or routing decision that caused the conflict $\tilde{C}_x$ is changed in the ongoing architectural optimization.

5.4 Conflict Refinement

In general it is possible to use the default conflict refinement of a state-of-the-art ILP solver. However, we suggest to use a group-based approach to significantly speed-up the conflict refinement. In the following it is shown that a fine-grained conflict refinement over all constraints in C_x is not necessary to determine the smallest set T_c as defined in Equation (36).

In the following, it is outlined how to define the groups of constraints that will be used for the problem-specific conflict refinement. Let a function *map* define the mapping in Equation (36) such that $T_c = map(\tilde{C}_x)$. For a given $\tilde{C}_x$, T_c is invariant to constraint $\tilde{c} \in \tilde{C}_x$ if $map(\tilde{C}_x) = map(\tilde{C}_x \backslash \{\tilde{c}\})$. This means that removing the constraint $\tilde{c}$ does not change the justification. A set of constraints C_t for each task $t \in T$ is defined as follows: It holds $\tilde{c} \in C_t$ if and only if $\mathbf{s_t} \in \tilde{c}$, i.e., each constraint $\tilde{c} \in C_t$ influences the starting time of task t. Let the groups be defined by $\mathcal{C}_T = \{C_t | t \in T\}$. It holds that $map(\tilde{C}_x) = \{t | C_t \in \mathcal{C}_T, C_t \cap \tilde{C}_x \neq \{\}\}$. Thus, it is sufficient to define the Irreducible Inconsistent Set (IIS) over the groups $\mathcal{C}_T$ instead of over all constraints in C_x. Since the cardinality of the $\mathcal{C}_T$ is significantly smaller than the number of constraints in C_x, this reduces the complexity of the IIS determination.

In the following, an approach to determine the IIS based on the set of constraint groups $\mathcal{C}_T$ is proposed. First, a deletion filter is presented that is capable of determining this IIS in an iterative manner. In order to quickly reduce the set of relevant constraints, additionally an elastic filter is presented.

Deletion filter. The deletion filter removes iteratively sets of constraints, ensuring that the remaining constraint are still unsatisfiable. It is presented in Algorithm 1.

The algorithm starts with the initial assumption that the IIS are all constraints in C_x. The algorithm removes the groups of constraints that affect one task iteratively (line 2). If the reduced set of constraints remains unsatisfiable (line 3), the group of constraints is permanently removed from the irreducible set $\tilde{C}_x$ (line 4), otherwise the group is not removed from the IIS $\tilde{C}_x$.

Algorithm 1 Deletion filter for conflict refinement based on the set of constraint groups $\mathcal{C}_T$. Given a set of constraints C_x, the IIS of constraints with respect to the groups $\mathcal{C}_T$ is defined as $\tilde{C}_x$.

1: $\tilde{C}_x = C_x$
2: **for** $C_t \in \mathcal{C}_T$ **do**
3: **if** $\forall x : x \not\models (\tilde{C}_x \backslash C_t)$ **then**
4: $\tilde{C}_x = \tilde{C}_x \backslash C_t$
5: **end if**
6: **end for**

In contrast to the IIS deletion approach known from literature that requires $|C_x|$ iterations, the proposed approach requires only $|T|$ iterations. In the proposed design space exploration, this approach significantly reduces the runtime since the number of tasks is much smaller than the number of constraints.

Elastic filter. The elastic filter quickly identifies a subset of constraints that is unsatisfiable. However, this subset does not have to be necessarily irreducible such that an additional application of the deletion filter becomes necessary afterwards. The elastic filter that considers the sets $\mathcal{C}_T$ is presented in Algorithm 2.

Algorithm 2 Elastic filter for conflict refinement based on the set of constraint groups $\mathcal{C}_T$. Given a set of constraints C_x, the IIS of constraints with respect to the groups $\mathcal{C}_T$ is defined as $\tilde{C}_x$.

1: $C'_x = \{c' |$ create elastic constraint c' for $c \in C_x$
2: with variable $\mathbf{e_c}\}$
3: **for** $t \in T, c \in C_t$ **do**
4: $C'_x = C'_x \cup \{\mathbf{e_t} \geq \mathbf{e_c}\}$
5: **end for**
6: $T' = T$
7: **while** $\exists x : x \vDash C'_x$ **do**
8: min $\sum_{t \in T'} \mathbf{e_t}$
9: $T' = T' \backslash \{t | \mathbf{e_t} > 0\}$
10: **for** $c \in C_x$ with $\forall C_t \cap \{c\} \neq \{\} : t \in T \backslash T'$ **do**
11: $C'_x = C'_x \cup \{\mathbf{e_c} = 0\}$
12: **end for**
13: **end while**
14: $\tilde{C}_x = C_x \backslash \bigcup_{t \in T'} C_t$

The conflict refinement based on an elastic filter requires to add an elastic variable $\mathbf{e_c}$ to each constraint c (line 1). Here, each linear constraint c is transformed to the form $g(x) \geq b$ and extended to $g(x) \geq b - \mathbf{e_c}$ with $\mathbf{e_c} \geq 0$. Thus, the constraint c is made elastic with the variable $\mathbf{e_c}$. In order to make equality-constraints elastic, they have to be converted to two corresponding $\geq$-constraints ($g(x) = b$ equals $g(x) \geq b$ and $-g(x) \geq -b$). In the next step (line 3), the elastic variables $\mathbf{e_c}$ are constrained by variables $\mathbf{e_t}$ which are defined for each $t \in T$ or $C_t \in \mathcal{C}_T$, respectively. The elastic constraints and the constraints for the bounds $\mathbf{e_t}$ are defined as C'_x. It is checked iteratively whether the constraints C'_x are satisfiable (line 7). If this is this case, a feasible solution is searched that minimizes the bounds $\mathbf{e_t}$ (line 8), i.e., a solution that tries to tighten as many elastic constraints as possible. In case the solution does require a variable $\mathbf{e_t}$ to be greater than 0, the corresponding tasks t are removed from

ECU	MHz	flash [kB] / RAM [kB]
MPC560xB (e200z0)	64	256/24, 384/28, 512/32, 768/64, 1024/80, 1536/96
MPC564xB (e200z4)	120	1536/128, 2048/160, 3072/192
MPC560xP (e200z0)	64	192/12, 256/20, 384/36, 512/40
S12XFxxx (S12(X))	50	128/16, 256/20, 384/24, 512/32
MPC560xS (e200z0)	64	256/24, 512/48, 1024/48
MPC5645S (e200z4)	125	2048/64
MPC564xF (e200z4)	150	2048/128, 3072/192, 4096/192
MPC567xF (e200z7)	200	3072/192, 4096/256

Table 1: Used ECUs for the experimental results. Each ECU might be configured with a specified flash memory size and RAM.

the set T' (line 9). In the following, all elastic variables $\mathbf{e_c}$ are set to 0 where the corresponding constraint c does only contain $\mathbf{s_t}$ variables for those tasks t that have been removed from T' (line 10-11). This ensures that these constraints are tightened in the next iteration. This approach is carried out until there exists no feasible solution and the algorithm is stopped. The subset of unsatisfiable constraints $\tilde{C}_x$ is defined by reducing the set C_x by the constraints C_t where the corresponding tasks $t \in T'$ did not need to be tightened (line 14). As mentioned before, the application of the deletion filter on the reduced set $\tilde{C}_x$ is necessary to prove or define the IIS, respectively.

6. EXPERIMENTAL RESULTS

In the following, experimental results based on a case study give evidence of the applicability of the proposed methodology. The case study models a realistic problem from the automotive domain. All experiments were carried out on an Intel Xeon CPU @3.20 GHz (QuadCore) with 12 GB RAM using the GUROBI ILP solver [10].

Case study. The case study is based on ECUs with 16-bit S12(X) microcontrollers and 32-bit PowerPC microcontrollers as given in Table 1. For the PowerPC architectures, three different cores are considered: e200z0, e200z4, e200z7. Each ECU has a specific clock speed and might be configured with a given flash memory size and RAM. For the FlexRay bus, a configuration with $n_{fx} = 60$ static slots with a duration of $e_{fx} = 0.0666$ms is chosen such that the length of the static segment is 4ms. The cycle duration of the FlexRay bus is $h_{fx} = 5$ms.

For the proposed case study, two linear objectives were considered: The overall flash size and overall RAM size. This is a common approach since the memory size is often an indicator for the monetary costs of a system in the automotive domain where high quantities are produced and delivered. Moreover, memory components for vehicles have to fulfill strict requirements regarding heat, cold, and vibration tolerances and are therefore significantly more expensive than consumer electronics.

For the time-triggered case study, a FlexRay-based system with ten ECUs is used as outlined in Table 2. Three body applications are mapped to one MPC564xB and one MPC560xB. Four chassis applications are mapped to two S12XFxxx and one MPC560xP. Four information applications are mapped to one MPC560xS and one MPC564xS. Three applications for hybrid and electric auxiliaries are mapped to two MPC567xF and one MPC564xF. The applications consist overall of 66 process tasks and 43 messages. The detailed number of process tasks and messages as well as the periods and deadlines, respectively, are given in Table 3. The mappings of the process tasks are in total 111, resulting in a very large search space for the architectural exploration with $1.98 \cdot 10^{13}$ possible task mappings. The periods for the applications range between 5ms and 80ms, while the deadlines equal the periods.

cluster	ECU	flash [kB] / RAM [kB]	
		opt1	opt2
body	MPC560xB	512/32	512/32
	MPC564xB	1536/128	1536/128
chassis	MPC560xP	256/20	192/12
	S12XFxxx	256/20	384/24
	S12XFxxx	256/20	256/20
information	MPC560xS	512/48	512/48
	MPC5645S	2048/64	2048/64
electric	MPC567xF	3072/192	3072/192
	MPC567xF	3072/192	4096/256
	MPC564xF	3072/192	2048/128

Table 2: Information about the ECUs of the case study and the optimal solution. The case study is using four clusters and ten ECUs. Two optimal results were obtained by the proposed optimization approach.

cluster	tasks	messages	period [ms]/deadline [ms]
body	4	1	10.0
	5	4	10.0
	5	3	20.0
chassis	6	5	20.0
	4	2	20.0
	4	3	20.0
	4	4	20.0
information	4	2	80.0
	7	4	80.0
	4	2	40.0
	5	5	40.0
electric	6	3	10.0
	4	2	5.0
	4	3	5.0

Table 3: Information about the applications that are used in the case study. Given is the number of process tasks and messages for each application as well as the period and deadline, respectively.

Results. The results of the iterative optimization process using the proposed framework are illustrated in Figure 5. The optimal results with feasible schedules were obtained after 77 iterations and 602 seconds. Finally, two optimal results were obtained as outlined in Table 2. The first optimal solution (opt1) requires 14592 kB flash memory and 908 kB RAM. The second optimal solution (opt2) requires 14656 kB flash memory and 904 kB RAM. Note that both objectives are strongly correlated and, therefore, the number of optimal solutions is low.

Within the optimization process, 38 architectures were obtained that allow a feasible scheduling while 39 architectures prohibited a feasible scheduling. However, the first feasible solution was obtained after 19 iterations. Until then, the conflict refinement and justification effectively removed infeasible solutions. The small size of the justification constraints ($|c|$) indicates that a very large part of the architectural design space is pruned. The results show that the proposed approach is capable of optimizing even large problems efficiently and is applicable to realistic problems from the automotive domain.

For 10000 randomly generated architectures for the given case study, all of these architectures prohibit a feasible schedul-

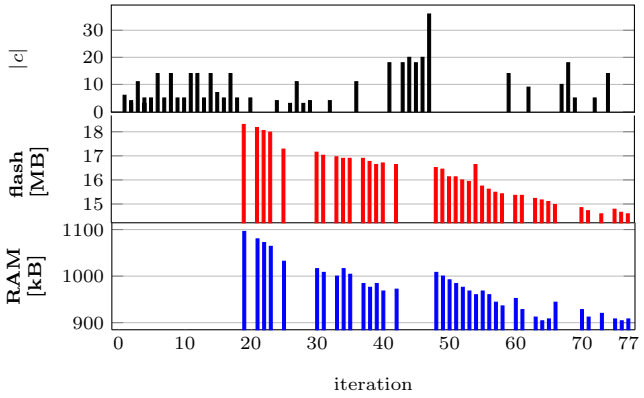

Figure 5: Illustration of the optimization process for the case study. The optimal results were obtained after 77 iterations. Illustrated is the size of the justifications c in terms of variables in the constraint ($|c|$), the flash size, and the RAM size.

ing. Thus, with known multi-step approaches from literature, the case study could not be optimized.

Scalability. Compared to other case studies in other concurrent optimization approaches such as [18, 19], the proposed case study is significantly larger. In particular, the number of tasks and messages in the used case study is very high compared to the number of ECUs. This leads to a very large search space for the scheduling. Nevertheless, the optimization runtime is very low with approximately only 10 minutes.

The scalability of the proposed approach depends on the scalability of the architecture exploration, the scheduling, and the combination of both, i.e., the conflict refinement and justification deduction. The architecture exploration is based on [14] which has been shown to be capable of handling very large problems. The architecture model is further extended by several constraints such as utilization and path delay constraints to restrict the search space. Moreover, the hierarchical scheduling approach allows the separate scheduling of different clusters, leading to a very scalable approach. Finally, the group-based conflict refinement shows a very good behavior for also very large problems.

Moreover, very often complex systems like automotive systems can be clustered into subsystems which can be designed separately. Thus, the proposed approach can be used iteratively for each subsystem as it is shown in the proposed case study.

7. CONCLUSION

This paper proposes a concurrent architecture and scheduling optimization for time-triggered automotive systems based on conflict refinement. In an iterative process, for each architecture, a time-triggered scheduling is carried out. In case no feasible schedule exists for a given architecture, a conflict refinement and justification deduction are performed to find the minimal architectural decision that prohibits a feasible schedule. This approach is incorporated into a global optimization framework. In particular, for systems with stringent timing constraints, the proposed approach is capable of finding optimal architectures in a short amount of time. The proposed approach is applicable beyond the automotive domain where complex optimization problems might be separated into subproblems accordingly to the proposed approach.

8. REFERENCES

[1] AUTOSAR. AUTomotive Open System ARchitecture. http://www.autosar.org.

[2] CAN. Controller Area Network. http://www.can.bosch.com/.

[3] S. Chakraborty, S. Kunzli, and L. Thiele. A General Framework for Analysing System Properties in Platform-based Embedded System Designs. In *Proc. of DATE 2003*, pages 190–195, 2003.

[4] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Period Optimization for Hard Real-time Distributed Automotive Systems. In *Proc. of DAC 2007*, pages 278–283, 2007.

[5] eCos. embedded Configurable operating system. http://ecos.sourceware.org/.

[6] H. Falk. WCET-aware Register Allocation Based on Graph Coloring. In *Proc. of DAC 2009*, pages 726–731, 2009.

[7] FlexRay Consortium. FlexRay Communications Systems - Protocol Specification. http://www.flexray.com.

[8] A. Gerstlauer, J. Peng, D. Shin, D. Gajski, A. Nakamura, D. Araki, and Y. Nishihara. Specify-Explore-Refine (SER): From Specification to Implementation. In *Proc. of DAC 2008*, pages 586–591, 2008.

[9] O. Guieu and J. W. Chinneck. Analyzing Infeasible Mixed-Integer and Integer Linear Programs. *INFORMS Journal on Computing*, 11:63–77, 1999.

[10] Gurobi Optimizer. Gurobi 4.6. http://www.gurobi.com.

[11] A. Hamann, M. Jersak, K. Richter, and R. Ernst. Design Space Exploration and System Optimization with SymTA/S–Symbolic Timing Analysis for Systems. In *Proc. of RTSS 2004*, pages 469–478, 2004.

[12] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich. SAT-Decoding in Evolutionary Algorithms for Discrete Constrained Optimization Problems. In *Proc. of CEC 2007*, pages 935–942, 2007.

[13] M. Lukasiewycz, R. Schneider, D. Goswammi, and S. Chakraborty. Modular Scheduling of Distributed Heterogeneous Time-triggered Automotive Systems. In *Proc. of the ASP-DAC 2012*, pages 665–670, 2012.

[14] M. Lukasiewycz, M. Streubühr, M. Glaß, C. Haubelt, and J. Teich. Combined System Synthesis and Communication Architecture Exploration for MPSoCs. In *Proc. of the DATE 2009*, pages 472–477, 2009.

[15] B. Meyer and D. Thomas. Simultaneous Synthesis of Buses, Data mapping and Memory Allocation for MPSoC. In *Proc. of CODES+ISSS 2007*, pages 3–8, 2007.

[16] Y. Oh, M. Mneimneh, Z. Andraus, K. Sakallah, and I. Markov. Amuse: a minimally-unsatisfiable subformula extractor. In *Proc. of DAC 2004*, pages 518–523, 2004.

[17] OSEK. OSEK VDX Portal. http://www.osek-vdx.org/.

[18] F. Reimann, M. Glaß, C. Haubelt, M. Eberl, and J. Teich. Improving Platform-based System Synthesis by Satisfiability Modulo Theories Solving. In *Proc. of CODES+ISSS 2010*, pages 135–144, 2010.

[19] F. Reimann, M. Lukasiewycz, M. Glass, C. Haubelt, and J. Teich. Symbolic System Synthesis in the Presence of Stringent Real-time Constraints. In *Proc. of DAC 2011*, pages 393–398, 2011.

[20] K. Richter, D. Ziegenbein, M. Jersak, and R. Ernst. Model Composition for Scheduling Analysis in Platform Design. In *Proc. of the DAC 2002*, pages 287–292, 2002.

[21] White Paper. Ethernet Time Synchronization, 2008. http://www.broadcom.com/collateral/wp/StrataXGSIV-WP100-R.pdf.

[22] 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.