Integrated Scheduling and Synthesis of Control Applications on Distributed Embedded Systems

Samii, Soheil; Cervin, Anton; Eles, Petru; Peng, Zebo

2009

Integrated Scheduling and Synthesis of Control Applications on Distributed Embedded Systems

Soheil Samii[1], Anton Cervin[2], Petru Eles[1], Zebo Peng[1]
[1]Department of Computer and Information Science
Linköping University, Sweden
{sohsa,petel,zpe}@ida.liu.se
[2]Department of Automatic Control
Lund University, Sweden
anton@control.lth.se

Abstract

Many embedded control systems comprise several control loops that are closed over a network of computation nodes. In such systems, complex timing behavior and communication lead to delay and jitter, which both degrade the performance of each control loop and must be considered during the controller synthesis. Also, the control performance should be taken into account during system scheduling. The contribution of this paper is a control–scheduling co-design method that integrates controller design with both static and priority-based scheduling of the tasks and messages, and in which the overall control performance is optimized.

1. Introduction and Related Work

The design of embedded control systems involves two main activities: synthesis of the controllers, and implementation of the control applications on a given execution platform. In the controller synthesis, given the plant to be controlled, a sampling period and a control law are chosen. It is also common to account for a constant sampling–actuation delay during the control-law computation. The synthesized controllers are implemented as a set of periodic tasks that read sensors, compute control signals, and write to actuators; the tasks can also implement other operations (e.g., signal processing). The platform, on which the applications execute, very often comprises a set of computation nodes that communicate on one or several buses; such distributed execution platforms are very common in, for example, automotive and avionics systems. The tasks and messages are scheduled on the nodes and the bus, respectively, either offline (static-cyclic scheduling [6]) or online (e.g., based on priorities [8]).

In addition to the computation delay of a control application itself, resource sharing and communication contribute to the delay in the control loop. It is well-known that not only the average delay degrades the control performance, but also the variance of the delay [15]—also called jitter. To achieve a good performance of distributed embedded control systems, it is thus important to consider the system timing during controller synthesis, and to consider the control performance during system scheduling. Such control–scheduling co-design problems [16] have become important research directions.

Seto et al. [14] studied mono-processor systems that run several control tasks. They solved the problem of optimal period assignment to each controller, with timing constraints given by two common scheduling policies: rate-monotonic and earliest-deadline-first scheduling [8]. The optimization goal is to maximize the performance of the running controllers. However, their approach does not take into account the delays in the control loop. Ben Gaid et al. [2] considered static scheduling of control signals, given one single control loop, which is closed over a communication channel, and the sampling period of the controller. The plant to be controlled, with given initial state, is assumed to be noise free. The result of the optimization is a finite sequence of control signals and start times for their transmissions over the communication channel to the actuator. Di Natale and Stankovic [4] proposed a simulated annealing-based approach that, given the application periods, constructs static schedules that minimize the jitter in distributed embedded systems with precedence and timing constraints. They did not, however, consider the impact of the schedules on the control performance.

The contribution of this paper is the formulation and solution of a control–scheduling co-design problem, where the goal is to optimize the overall performance of several control loops. Given is a set of plants with disturbance and measurement noise. As part of the optimization, we synthesize a controller (sampling period and control law) for each plant. Further, considering both static-cyclic scheduling and priority-based scheduling, we schedule the execution and communication of the control applications on the given distributed execution platform, such that the overall performance of the control loops is optimized.

2. System Model

Given is a set of plants $\mathbf{P}$, indexed by the set $\mathcal{I}_{\mathbf{P}}$, for which controllers and their implementation are to be synthesized. Each plant P_i ($i \in \mathcal{I}_{\mathbf{P}}$) is described by a continuous-time linear model [17]

$$d\boldsymbol{x}_i/dt = A_i\boldsymbol{x}_i + B_i\boldsymbol{u}_i + \boldsymbol{v}_i,\ \boldsymbol{y}_i = C_i\boldsymbol{x}_i + \boldsymbol{e}_i, \qquad (1)$$

where the vectors $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the plant state and controlled input, respectively, and the vector $\boldsymbol{v}_i$ models plant disturbance as a continuous-time white-noise process with intensity R_{1i}. The continuous-time output $\boldsymbol{y}_i$ is measured and sampled periodically and is used to produce the control signal $\boldsymbol{u}_i$. The measurement noise $\boldsymbol{e}_i$ is modeled as a discrete-time white-noise process with variance R_{2i}. The control signal is updated at discrete time-instants t_k and is held constant between two updates (zero-order hold [17]). The control signal can thus be viewed as a sequence $\{\boldsymbol{u}_{i,k}\}$, where the index k is used to denote updates (i.e., $\boldsymbol{u}_i(t) = \boldsymbol{u}_{i,k}$ for $t_k \leqslant t < t_{k+1}$). As an example, let us consider a set of two inverted pendulums $\mathbf{P} = \{P_1, P_2\}$. Each pendulum P_i ($i \in \mathcal{I}_{\mathbf{P}} = \{1, 2\}$) is modeled according to Equation 1, with $A_i = \begin{bmatrix} 0 & 1 \\ g/l_i & 0 \end{bmatrix}$, $B_i = \begin{bmatrix} 0 & g/l_i \end{bmatrix}^{\mathsf{T}}$, and $C_i = \begin{bmatrix} 1 & 0 \end{bmatrix}$, where $g \approx 9.81$ m/s^2 and l_i are the gravitational constant and length of pendulum P_i, respectively ($l_1 = 0.2$ m and $l_2 = 0.1$ m). For the plant disturbance and measurement noise, respectively, we have $R_{1i} = B_i B_i^{\mathsf{T}}$ and $R_{2i} = 0.1$.

Each plant has a controller that runs as an application mapped to an execution platform. The platform consists of a set of computation nodes $\mathbf{N}$, indexed by $\mathcal{I}_{\mathbf{N}}$, that are connected by a communication controller to a bus with a given communication protocol. Figure 1 shows two nodes ($\mathbf{N} = \{N_1, N_2\}$) that are connected to a bus (the communication controllers are denoted CC). On the execution platform runs a set of applications $\mathbf{\Lambda}$, indexed by the set $\mathcal{I}_{\mathbf{\Lambda}}$ (the set of applications may include other applications than the control applications—thus, $\mathcal{I}_{\mathbf{\Lambda}} \supseteq \mathcal{I}_{\mathbf{P}}$). An application $\Lambda_i \in \mathbf{\Lambda}$ ($i \in \mathcal{I}_{\mathbf{\Lambda}}$) is modeled as a directed acyclic graph $(\mathbf{T}_i, \mathbf{\Gamma}_i)$, where the nodes $\mathbf{T}_i$, indexed by $\mathcal{I}_i$, represent computation tasks and the edges $\mathbf{\Gamma}_i \subset \mathbf{T}_i \times \mathbf{T}_i$ represent data dependen-

cies between the tasks. In Figure 1, we show two applications $\mathbf{\Lambda} = \{\Lambda_1, \Lambda_2\}$, which are controllers for the two pendulums P_1 and P_2 in the example before. For $i \in \mathcal{I}_{\mathbf{\Lambda}}$, the task set of application Λ_i is $\mathbf{T}_i = \{\tau_{is}, \tau_{ic}, \tau_{ia}\}$ ($\mathcal{I}_i = \{s, c, a\}$). Changing back to the general discussion, for each $i \in \mathcal{I}_{\mathbf{P}}$, application Λ_i is the controller for plant P_i. For each such application, we denote with τ_{ia} ($a \in \mathcal{I}_i$) the actuator task in the controller for plant P_i.

An application $\Lambda_i \in \mathbf{\Lambda}$ has a release period h_i. At time $(q-1)h_i$, a job of each task in the application is released for execution. Job q of task τ_{ij} is denoted $\tau_{ij}^{(q)}$ and is released at time $(q-1)h_i$. For a message $\gamma_{ijk} = (\tau_{ij}, \tau_{ik}) \in \mathbf{\Gamma}_i$, the message instance produced by job $\tau_{ij}^{(q)}$ is denoted $\gamma_{ijk}^{(q)}$. An edge $\gamma_{ijk} = (\tau_{ij}, \tau_{ik}) \in \mathbf{\Gamma}_i$ indicates that the earliest start time of a job $\tau_{ik}^{(q)}$ is when $\tau_{ij}^{(q)}$ has completed its execution and the produced data (i.e., $\gamma_{ijk}^{(q)}$) has been communicated to $\tau_{ik}^{(q)}$. We also define the hyperperiod $h_{\mathbf{\Lambda}}$ as the least common multiple of the periods $\{h_i : i \in \mathcal{I}_{\mathbf{\Lambda}}\}$. Further, a task can have a deadline, which means that any job of that task must finish within a given time relative to its release. Control applications do not typically have hard timing constraints, but instead the goal is to achieve a good quality of control.

Each task is mapped to a node. The mapping is given by a function $\mathsf{map} : \bigcup_{i \in \mathcal{I}_{\mathbf{\Lambda}}} \mathbf{T}_i \longrightarrow \mathbf{N}$. Let us also introduce the function $\mathsf{map}^* : \mathbf{N} \longrightarrow 2^{\bigcup_{i \in \mathcal{I}_{\mathbf{\Lambda}}} \mathbf{T}_i}$ that, given a node N_d, gives the set of tasks that are mapped to N_d—thus, $\mathsf{map}^*(N_d) = \{\tau_{ij} \in \bigcup_{i \in \mathcal{I}_{\mathbf{\Lambda}}} \mathbf{T}_i : \mathsf{map}(\tau_{ij}) = N_d\}$. A message between tasks mapped to different nodes is sent on the bus; thus, the set of messages that are communicated on the bus is $\mathbf{\Gamma}_{\text{bus}} = \{\gamma_{ijk} = (\tau_{ij}, \tau_{ik}) \in \mathbf{\Gamma}_i : \mathsf{map}(\tau_{ij}) \neq \mathsf{map}(\tau_{ik}), i \in \mathcal{I}_{\mathbf{\Lambda}}\}$. For a message instance $\gamma_{ijk}^{(q)}$, we denote with c_{ijk} the communication time when there are no conflicts on the bus. Given a mapping of the tasks to the nodes, for each task, we have a specification of possible execution times. We model the execution time of task τ_{ij} as a stochastic variable c_{ij} with probability function $\xi_{c_{ij}}$. The execution time is bounded by given best-case and worst-case execution times, denoted c_{ij}^{bc} and c_{ij}^{wc}, respectively. In Figure 1, the execution times (constant in this example) and communication times for the tasks and messages are given in milliseconds inside parentheses.

3. Control Quality and Synthesis

We measure the quality of a controller Λ_i for a plant P_i (Equation 1) with a quadratic cost [17]

$$J_i = \lim_{T \to \infty} \frac{1}{T} \mathrm{E}\left\{ \int_0^T \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix}^{\mathsf{T}} Q_i \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix} dt \right\}. \qquad (2)$$

The weight matrix Q_i, which is given by the designer, is a positive semi-definite matrix with weights for the magnitude of the plant states and the control signals ($\mathrm{E}\{\cdot\}$ denotes the expected value of a stochastic variable).

For a given sampling period h_i and a given, constant sensor–actuator delay δ_i^{sa} (i.e., the time between sampling the output $\boldsymbol{y}_i$ and updating the controlled input $\boldsymbol{u}_i$), it is possible to find the control law $\boldsymbol{u}_i$ that minimizes the cost J_i [17]. The quality of a controller is degraded (its cost J_i is increased) if the sensor–actuator delay is different from what was assumed during the control-law synthesis, or if this delay is not constant (i.e., there is jitter). Assuming that the sensor–actuator delay is represented as a stochastic variable Δ_i^{sa} with probability function $\xi_{\Delta_i}^{\text{sa}}$, the cost J_i can be computed with MATLAB and the Jitterbug toolbox [7].

4. Motivational Example

In Section 2, we introduced an example in which two inverted pendulums are controlled over a network of two computation nodes. Each application Λ_i in Figure 1 consists of three tasks, denoted τ_{is}, τ_{ic}, and τ_{ia}, respectively. All time quantities are given in milliseconds throughout this section. We use static-cyclic scheduling of the tasks and messages. The weight matrix of each inverted pendulum is $Q_i = \mathrm{diag}\left(C_i^{\mathsf{T}} C_i, 0.002\right)$.

First, let us assign the periods of the controllers as $h_1 = 20$ and $h_2 = 30$. Let us also consider that each control law $\boldsymbol{u}_i$ is computed for the chosen period h_i and a constant delay equal to the sum of task execution times (i.e., $c_{is} + c_{ic} + c_{ia}$). Thus, control law $\boldsymbol{u}_1$ is computed for the period 20 and the constant delay 9, whereas $\boldsymbol{u}_2$ is computed for the period 30 and the constant delay 13. We have computed the individual controller costs, given that the sampling–actuation delays during execution are exactly those that were assumed during the controller design. We obtained the costs $J_1 = 0.9$ and $J_2 = 2.4$; the total cost is $J_{\text{tot}} = J_1 + J_2 = 3.3$. However, it is often not possible to schedule the task executions and message transmissions in a way that leads to the delay characteristics assumed during controller design. One system schedule is depicted in Figure 2. We show the schedule with three rows for node N_1, the bus, and node N_2, respectively. The boxes depict the task executions and message transmissions. The grey boxes show the execution of control application Λ_1, whereas the white boxes show the execution of Λ_2. Each box is labeled with an index that specifies the corresponding task or message, and with a number that specifies the job or message instance. For example, the white box labeled $a(2)$ shows the execution of job $\tau_{2a}^{(2)}$ of the actuator task τ_{2a}. The job starts and finishes at times 50 and 54, respectively. The grey box labeled $sc(1)$ shows the first message $\gamma_{1sc}^{(1)}$ between the sensor and controller task of Λ_1. The schedule period is 60.

The outputs of the plants are sampled periodically without jitter (e.g., by some dedicated hardware mechanism that stores the sampled data in buffers, which are read by the sensor tasks). Let us now study the sampling–actuation delay Δ_i^{sa} of the two control loops. In the schedule in Figure 2, we have three instances of Λ_1. The three actuations $\tau_{1a}^{(1)}$, $\tau_{1a}^{(2)}$, and $\tau_{1a}^{(3)}$ finish at times 32, 49, and 54, respectively. By considering the sampling period 20, we obtain the sampling–actuation delays 32, 29, and 14. Each of these delays occur with the same frequency during execution. This means that the nonzero function values of the probability function are $\xi_{\Delta_1}^{\text{sa}}(14) = \xi_{\Delta_1}^{\text{sa}}(29) = \xi_{\Delta_1}^{\text{sa}}(32) = 1/3$. Thus, as a result of the implementation in Figure 2, the actual delay is different from 9, which is the delay assumed during controller synthesis, and, moreover, it is not constant. By using the Jitterbug toolbox and providing as input the probability function $\xi_{\Delta_1}^{\text{sa}}$, we obtained a much higher cost $J_1 = 4.2$ of the implemented controller. Similarly, the two instances of application Λ_2 have the delays 44 and 24—that is, $\xi_{\Delta_2}^{\text{sa}}(24) = \xi_{\Delta_2}^{\text{sa}}(44) = 1/2$. The corresponding cost is $J_2 = 6.4$. The total cost of the whole system is $J_{\text{tot}} = 10.6$ and has increased significantly (from 3.3) as a result of the implementation.

To obtain a better control performance, it is important to reduce the delay and jitter. Let us study the schedule in Figure 3, without changing the periods and the control laws. The sensor–actuator delay is now 14 for all the three instances of Λ_1 (i.e., $\xi_{\Delta_1}^{\text{sa}}(14) = 1$). Similarly, we obtain $\xi_{\Delta_2}^{\text{sa}}(18) = \xi_{\Delta_2}^{\text{sa}}(24) = 1/2$ (the second control loop has a

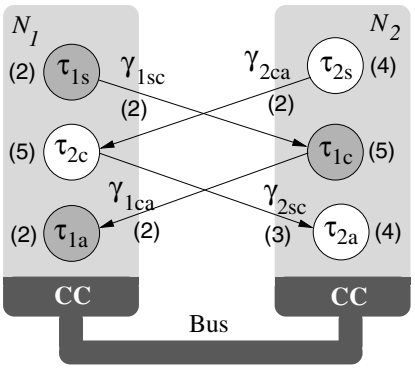

Figure 1. Example with two applications on a two-node platform

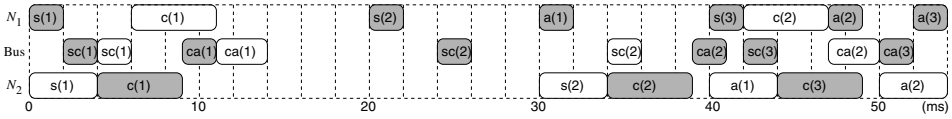

Figure 2. Schedule for $h_1 = 20$ ms and $h_2 = 30$ ms

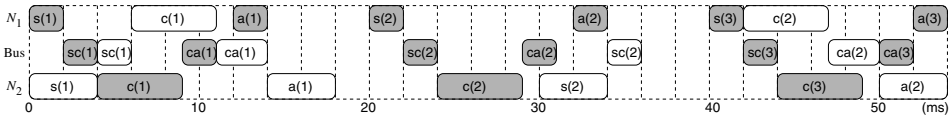

Figure 3. Optimized schedule for $h_1 = 20$ ms and $h_2 = 30$ ms

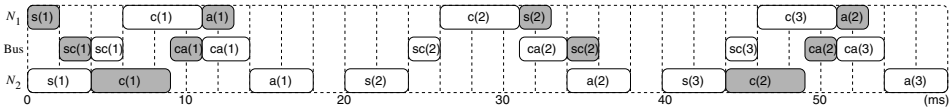

Figure 4. Optimized schedule for $h_1 = 30$ ms and $h_2 = 20$ ms

smaller delay with less jitter than in Figure 2). Considering the new schedule, the costs are $J_1 = 1.1$ and $J_2 = 5.6$; the total cost is thus $J_{\text{tot}} = 6.7$. We note that the performance is improved if the tasks and messages are properly scheduled to decrease the delay and jitter. With the same schedule (Figure 3), we can further improve the control performance by recomputing the control laws. Therefore, let us assume that the control law for P_1 is recomputed for the delay 14, which is the constant sensor–actuator delay in the schedule. The control law for P_2 is recomputed for the average sensor–actuator delay $\mathrm{E}\{\Delta_2^{\text{sa}}\} = 21$. With the new controllers, the costs of the implementation are $J_1 = 1.0$ and $J_2 = 3.7$; the total cost is $J_{\text{tot}} = 4.7$, which indicates the performance improvement that is achieved by taking the schedule into account during controller design.

As a last step, we show that the overall performance can be further improved by another selection of the sampling periods. Let us change the periods of the two control applications to $h_1 = 30$ and $h_2 = 20$. Figure 4 shows a schedule with two instances of Λ_1 and three instances of Λ_2 (the period of the schedule is 60). The delays in the first control loop are 13 and 23—thus, $\xi^{\text{sa}}_{\Delta_1}(13) = \xi^{\text{sa}}_{\Delta_1}(23) = 1/2$. This means that the first control loop has some jitter, which is not the case in the schedule in Figure 3. We have designed the control law for the constant delay $\mathrm{E}\{\Delta_1^{\text{sa}}\} = 18$. The delay in the second control loop is 14 (constant), for which we designed the control law. The evaluation resulted in the costs $J_1 = 1.3$ and $J_2 = 2.1$—thus, $J_{\text{tot}} = 3.4$. The example in this section demonstrates that a good selection of controller periods, combined with integrated scheduling and controller design, is important to design high-performance embedded control systems.

5. Problem Formulation

The inputs to the control–scheduling co-design problem are

- a set of plants $\mathbf{P}$ to be controlled,
- a set of applications $\mathbf{\Lambda}$ among which a subset of them are the controllers for the plants ($\mathcal{I}_{\mathbf{P}} \subseteq \mathcal{I}_{\mathbf{\Lambda}}$),
- a set of available sampling periods $\mathbf{H}_i$ ($i \in \mathcal{I}_{\mathbf{P}}$) for each control application Λ_i and the release period for each of the other applications,
- deadlines of a subset of the tasks (possibly no deadlines),
- a set of computation nodes $\mathbf{N}$ connected to a bus,
- a scheduling policy for the tasks and messages,
- a mapping function map of the whole task set, and
- execution-time distributions of the tasks and communication times of messages.

The outputs related to the controller synthesis are the period $h_i \in \mathbf{H}_i$ and the control law $\boldsymbol{u}_i$ for each plant P_i. The outputs related to the scheduling depends on the scheduling policy. For static-cyclic scheduling, the output is a schedule table with start times of the job executions and the communications on the bus. In the case of priority-based scheduling, the output is a priority for each task and message. In both cases, it must be guaranteed that task deadlines are met at runtime. The cost function to be minimized is a weighted sum $\sum_{i \in \mathcal{I}_{\mathbf{P}}} w_i J_i$ of the individual controller costs J_i (Equation 2). The weights w_i and Q_i are given as inputs by the designer.

6. Scheduling and Synthesis Approach

6.1. Overall Solution

Figure 5 illustrates the overall approach. The dashed box in the figure shows the portion of the flowchart that is specific for static-cyclic scheduling (Section 6.2). If priority-based scheduling is used, this box is replaced by the flowchart in Figure 6 (Section 6.3). In the outer loop, we iterate over different assignments of the controller periods. In each iteration, we choose a period for each control application in the set of available periods; thus, we choose a vector $\boldsymbol{h} = (h^{(1)}, \ldots, h^{(|\mathcal{I}_{\mathbf{P}}|)}) \in \mathbf{H}_{\sigma_{\mathbf{P}}(1)} \times \cdots \times \mathbf{H}_{\sigma_{\mathbf{P}}(|\mathcal{I}_{\mathbf{P}}|)} = \prod_{i=1}^{|\mathcal{I}_{\mathbf{P}}|} \mathbf{H}_{\sigma_{\mathbf{P}}(i)}$, where $\sigma_{\mathbf{P}} : \{1, \ldots, |\mathcal{I}_{\mathbf{P}}|\} \longrightarrow \mathcal{I}_{\mathbf{P}}$ is any bijection and $\prod$ denotes the Cartesian product of sets. The period of controller Λ_i ($i \in \mathcal{I}_{\mathbf{P}}$) is thus $h_i = h^{\left(\sigma_{\mathbf{P}}^{-1}(i)\right)}$. For such a period assignment $\boldsymbol{h}$, we perform the following steps (the dashed rectangle in Figure 5): schedule the whole system (Sections 6.2 and 6.3), synthesize the control law for each plant, and compute the cost of each control loop, obtaining a final cost $J_{\boldsymbol{h}}$ of the period assignment $\boldsymbol{h}$. The last two steps are described in the continuation of this section. In the end of this section, we describe the period-exploration process. Let us first, however, introduce the stochastic variable Δ_i^{sa} for the sampling–actuation delay of controller Λ_i ($i \in \mathcal{I}_{\mathbf{P}}$). The probability function $\xi^{\text{sa}}_{\Delta_i}$ of this delay is determined by the execution-time distributions $\xi_{c_{ij}}$ of the tasks and the scheduling of the whole system. In Sections 6.2 and 6.3, we elaborate on how to derive $\xi^{\text{sa}}_{\Delta_i}$ for the case of static and priority-based scheduling, respectively.

Given is an assignment of controller periods $\boldsymbol{h} = (h^{(1)}, \ldots, h^{(|\mathcal{I}_{\mathbf{P}}|)})$, where $h^{(i)}$ is the period of controller $\Lambda_{\sigma_{\mathbf{P}}(i)}$, we shall synthesize the control laws. From the task and message scheduling, we obtain the sampling–actuation delay Δ_i^{sa} of each controller Λ_i. Each control law $\boldsymbol{u}_i$ is chosen to minimize the cost J_i for the sampling period $h^{\left(\sigma_{\mathbf{P}}^{-1}(i)\right)}$ and a constant sampling–actuation delay $\delta_i^{\text{sa}} = \mathrm{E}\{\Delta_i^{\text{sa}}\}$ (the expected value of the delay). This controller synthesis is based on stan-

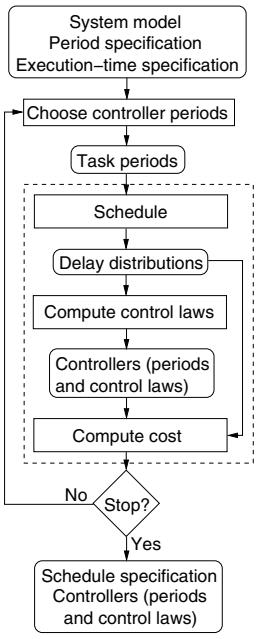

Figure 5. Overall approach with static-cyclic scheduling

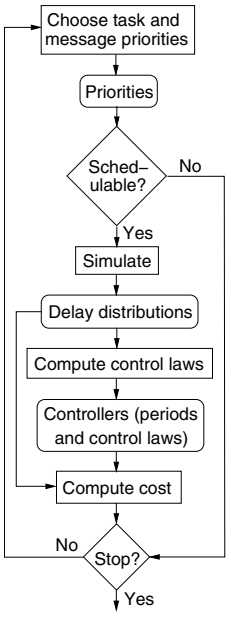

Figure 6. The priority-based scheduling approach

dard control-theory [17], provided in the Jitterbug toolbox [7]. For each plant P_i ($i \in \mathcal{I}_{\mathbf{P}}$), we now have a controller Λ_i with a period h_i and a control law $\boldsymbol{u}_i$ synthesized for a constant delay $\mathrm{E}\{\Delta_i^{\mathrm{sa}}\}$. The actual implementation, however, results in a nonconstant sensor–actuator delay. This delay is modeled as a stochastic variable Δ_i^{sa} for which the probability function $\xi_{\Delta_i}^{\mathrm{sa}}$ is given by the system schedule (Sections 6.2 and 6.3). We compute the cost J_i of each control loop based on Δ_i^{sa}. The total cost of the periods $\boldsymbol{h}$ is $J_{\boldsymbol{h}} = \sum_{i \in \mathcal{I}_{\mathbf{P}}} w_i J_i$.

The period-exploration process is based on a genetic algorithm [12]. An initial population $\boldsymbol{\Psi}_1 \subset \prod_{i=1}^{|\mathcal{I}_{\mathbf{P}}|} \mathbf{H}_{\sigma_{\mathbf{P}}(i)}$ is generated randomly. At each iteration $k > 0$, the cost $J_{\boldsymbol{h}}$ of each member $\boldsymbol{h} \in \boldsymbol{\Psi}_k$ in the population is computed by performing three steps: system scheduling, control-law synthesis and cost computation. Using the crossover and mutation operators [12] on the current population $\boldsymbol{\Psi}_k \subset \prod_{i=1}^{|\mathcal{I}_{\mathbf{P}}|} \mathbf{H}_{\sigma_{\mathbf{P}}(i)}$, we generate a set of offsprings $\boldsymbol{\Psi}_k^{\mathrm{offspr}} \subset \prod_{i=1}^{|\mathcal{I}_{\mathbf{P}}|} \mathbf{H}_{\sigma_{\mathbf{P}}(i)}$. We continue by generating randomly a subset $\boldsymbol{\Psi}_k' \subset \boldsymbol{\Psi}_k$, for which $|\boldsymbol{\Psi}_k'| = \left|\boldsymbol{\Psi}_k^{\mathrm{offspr}}\right|$, with members that will be replaced by the generated offsprings; the probability for a member $\boldsymbol{h} \in \boldsymbol{\Psi}_k$ to be included in $\boldsymbol{\Psi}_k'$ is larger the smaller its cost $J_{\boldsymbol{h}}$ is. The population to be evaluated in the next iteration is $\boldsymbol{\Psi}_{k+1} = (\boldsymbol{\Psi}_k \setminus \boldsymbol{\Psi}_k') \cup \boldsymbol{\Psi}_k^{\mathrm{offspr}}$.

We have tuned the parameters of the genetic algorithm experimentally. The population size is constant ($|\boldsymbol{\Psi}_k| = |\boldsymbol{\Psi}_{k+1}|$) and is chosen to be $2\,|\mathcal{I}_{\mathbf{P}}| \max_{i \in \mathcal{I}_{\mathbf{P}}} |\mathbf{H}_i|$. The number of offsprings that are generated in each iteration is $\left|\boldsymbol{\Psi}_k^{\mathrm{offspr}}\right| = 0.25\,|\boldsymbol{\Psi}_k|$, whereas the mutation probability is 0.1. The exploration terminates when the average cost J_{avg} of the current population is sufficiently close to the cost J_{min} of the current best solution: In our implementation, we stop the period exploration process when $J_{\mathrm{avg}} < 1.05 J_{\mathrm{min}}$. In the following two subsections, we shall focus on static-cyclic and priority-based scheduling separately. In both cases, we consider the period h_i of each application Λ_i given ($i \in \mathcal{I}_{\boldsymbol{\Lambda}}$); that is, a period assignment $\boldsymbol{h} \in \prod_{i=1}^{|\mathcal{I}_{\mathbf{P}}|} \mathbf{H}_{\sigma_{\mathbf{P}}(i)}$ is considered.

6.2. Static-Cyclic Scheduling

Given the periods $\boldsymbol{h}$ and the hyperperiod $h_{\boldsymbol{\Lambda}}$, let $\boldsymbol{\Theta}_{N_d} = \bigcup_{\tau_{ij} \in \mathsf{map}^*(N_d)} \left\{\tau_{ij}^{(q)} : q = 1, \ldots, h_{\boldsymbol{\Lambda}}/h_i\right\}$ be the set of jobs that are released for execution on node N_d ($d \in \mathcal{I}_{\mathbf{N}}$) in the time interval $[0, h_{\boldsymbol{\Lambda}}[$. Let us also define the set of message instances $\boldsymbol{\Theta}_{\mathrm{bus}}$ that are communicated on the bus in the time interval $[0, h_{\boldsymbol{\Lambda}}[$ as $\boldsymbol{\Theta}_{\mathrm{bus}} = \bigcup_{\gamma_{ijk} \in \boldsymbol{\Gamma}_{\mathrm{bus}}} \left\{\gamma_{ijk}^{(q)} : q = 1, \ldots, h_{\boldsymbol{\Lambda}}/h_i\right\}$. A static-cyclic schedule $\boldsymbol{\Omega}$ is a set of schedules $\boldsymbol{\Omega} = \bigcup_{i \in \mathcal{I}_{\mathbf{N}}} \{\Omega_i\} \cup \{\Omega_{\mathrm{bus}}\}$ for each computation node and the bus. For each $d \in \mathcal{I}_{\mathbf{N}}$, the schedule for node N_d is an injective function $\Omega_d : \boldsymbol{\Theta}_{N_d} \longrightarrow [0, h_{\boldsymbol{\Lambda}}[$ that gives the start time of each job. The bus schedule is an injective function $\Omega_{\mathrm{bus}} : \boldsymbol{\Theta}_{\mathrm{bus}} \longrightarrow [0, h_{\boldsymbol{\Lambda}}[$ that gives the start time of each message instance. The schedule $\boldsymbol{\Omega}$ is periodic with the period $h_{\boldsymbol{\Lambda}}$. Let $q' = 1 + (q - 1 \bmod h_{\boldsymbol{\Lambda}}/h_i)$. Then, the periodicity of the schedule means that, for each $\tau_{ij} \in \mathsf{map}^*(N_d)$, the start time of job $\tau_{ij}^{(q)}$ is $\left\lfloor (q-1) \Big/ \frac{h_{\boldsymbol{\Lambda}}}{h_i} \right\rfloor h_{\boldsymbol{\Lambda}} + \Omega_d\left(\tau_{ij}^{(q')}\right)$, whereas, for each message $\gamma_{ijk} \in \boldsymbol{\Gamma}_{\mathrm{bus}}$, the start time of $\gamma_{ijk}^{(q)}$ is $\left\lfloor (q-1) \Big/ \frac{h_{\boldsymbol{\Lambda}}}{h_i} \right\rfloor h_{\boldsymbol{\Lambda}} + \Omega_{\mathrm{bus}}\left(\gamma_{ijk}^{(q')}\right)$.

Constructing the static schedule for a hyperperiod implies the determination of the start times of all jobs and message instances. The schedule must satisfy the precedence constraints, given by the data dependencies between tasks, and account for the worst-case execution times of tasks and the communication times of messages. Moreover, deadlines must be met, if imposed on certain tasks, and all executions and communications must finish before the hyperperiod. Finally, at any time instant, at most one job can execute on a given node, and at most one message can be transmitted on the bus.

Given a schedule $\boldsymbol{\Omega}$, we are interested in the sensor–actuator delay Δ_i^{sa} of each control application Λ_i ($i \in \mathcal{I}_{\mathbf{P}}$). The probability function of Δ_i^{sa} is determined by the start times of the actuator task τ_{ia} and the stochastic execution time c_{ia} with given probability function $\xi_{c_{ia}}$. Let $N_d = \mathsf{map}(\tau_{ia})$ denote the computation node for the actuator task of controller Λ_i. For $q = 1, \ldots, h_{\boldsymbol{\Lambda}}/h_i$, in the q^{th} instance of Λ_i, the sensors are read at time $(q-1)h_i$. The start time of the corresponding actuation is $\Omega_d\left(\tau_{ia}^{(q)}\right)$. Letting $\phi_{ia}^{(q)} = \Omega_d\left(\tau_{ia}^{(q)}\right) - (q-1)h_i$, the sampling–actuation delay in the q^{th} controller instance is distributed between a minimum and maximum delay given by $\phi_{ia}^{(q)} + c_{ia}^{\mathrm{bc}}$ and $\phi_{ia}^{(q)} + c_{ia}^{\mathrm{wc}}$, respectively. The probability function of the delay in the q^{th} instance is therefore $\xi_{\Delta_i}^{\mathrm{sa}(q)}(\delta) = \xi_{c_{ia}}\left(\delta - \phi_{ia}^{(q)}\right)$. Considering all jobs in the schedule, the probability function of the delay Δ_i^{sa} is $\xi_{\Delta_i}^{\mathrm{sa}}(\delta) = \frac{h_i}{h_{\boldsymbol{\Lambda}}} \sum_{q=1}^{h_{\boldsymbol{\Lambda}}/h_i} \xi_{\Delta_i}^{\mathrm{sa}(q)}(\delta)$.

Our goal is to find a schedule $\boldsymbol{\Omega}$ that minimizes the cost $\sum_{i \in \mathcal{I}_{\mathbf{P}}} w_i J_i$. In our approach, we use a constraint logic programming (CLP) formulation [1] of the static-scheduling problem. We have used the ECL^{i}PS^e solver (version 5.10) [1]. It is known that two timing parameters affect the control performance: the average delay and the jitter. During the construction of the schedule, we minimize therefore the quadratic cost $\sum_{i \in \mathcal{I}_{\mathbf{P}}} w_i \left(\alpha_i \mathrm{E}\left\{\Delta_i^{\mathrm{sa}}\right\}^2 + \beta_i \mathrm{D}\left\{\Delta_i^{\mathrm{sa}}\right\}^2\right)$, where α_i and β_i are designer inputs that specify the sensitivity of plant P_i to delay and jitter, respectively ($\mathrm{D}\{\cdot\}$ denotes the standard deviation of a stochastic variable); we have used $\alpha_i = \beta_i = 1$ in our experiments. The constraints in the CLP formulation are

given by the definition of a static-cyclic schedule. The CLP solver is a branch-and-bound search based on constraint propagation [1]. Because the scheduling problem is NP-complete, we cannot, for a large problem size, afford a complete search of the set of possible schedules. Therefore, we have configured the CLP solver to use limited-discrepancy search [1], which is an incomplete branch-and-bound search.

6.3. Priority-Based Scheduling

In this subsection, we consider preemptive scheduling of tasks and nonpreemptive scheduling of messages, both based on fixed priorities. The overall flowchart of the solution for the case of priority-based scheduling is obtained by replacing the dashed box in Figure 5 with the flowchart in Figure 6. Given from the outer loop in Figure 5 are the periods $\boldsymbol{h}$ of all applications. The goal is to minimize the overall cost by deciding task and message priorities, and by computing the control laws. In the outer loop in Figure 6, we iterate over different priority assignments, and for each assignment we compute the cost. In the continuation of this section, we describe the computation flow that leads to the cost of a priority assignment. Last, we describe the priority-exploration approach.

Let us define a priority assignment as a set $\boldsymbol{\rho} = \bigcup_{d\in\mathcal{I}_\mathbf{N}}\{\rho_d\} \cup \{\rho_{\text{bus}}\}$, where $\rho_d : \mathsf{map}^*(N_d) \longrightarrow \mathbb{N}$ and $\rho_{\text{bus}} : \mathbf{\Gamma}_{\text{bus}} \longrightarrow \mathbb{N}$ are injective functions that give the priorities of the tasks on node N_d ($d \in \mathcal{I}_\mathbf{N}$) and the messages on the bus, respectively (a larger value indicates a higher priority). Given a priority assignment $\boldsymbol{\rho}$, we are interested in the temporal behavior of the system. Given the periods and priorities, in the first step, we check whether the system is schedulable; that is, we run response-time analysis [9, 10, 11] to obtain the worst-case response time of each task. The system is schedulable if all worst-case response times exist and are smaller than or equal to the imposed task deadlines. Note that, in this step, we check the satisfaction of the imposed hard timing constraints by formal response-time analysis. In the next step, we compute the control laws based on delay distributions obtained with simulation.

Given the periods and priorities, we use our simulation environment for distributed real-time systems [13] to obtain an approximation $\widehat{\Delta}_i^{\text{sa}}$ of each sensor–actuator delay Δ_i^{sa}. The probability function of the discrete stochastic variable $\widehat{\Delta}_i^{\text{sa}}$, which approximates Δ_i^{sa}, is denoted $\widehat{\xi}_{\Delta_i}^{\text{sa}}$ and is an output of the simulation. During the simulation, we compute the average sensor–actuator delays periodically with the period $h_\mathbf{\Lambda}$. Let $\mathbf{\Delta}_i^{(k)}$ denote the set of sensor–actuator delays for application Λ_i ($i \in \mathcal{I}_\mathbf{P}$) that, during simulation, occur in the time interval $[0, kh_\mathbf{\Lambda}]$ ($k > 0$). Further, let $\eta_{\mathbf{\Delta}_i}^{(k)} : \mathbf{\Delta}_i^{(k)} \longrightarrow \mathbb{Z}^+$ be a function for which $\eta_{\mathbf{\Delta}_i}^{(k)}(\delta)$ is the number of times the delay δ occurred in the time interval $[0, kh_\mathbf{\Lambda}]$. The total number of delays for Λ_i in the simulated time interval is $\eta_{i,\text{tot}}^{(k)} = \sum_{\delta\in\mathbf{\Delta}_i^{(k)}} \eta_{\mathbf{\Delta}_i}^{(k)}(\delta)$. At times $kh_\mathbf{\Lambda}$ during simulation, we compute an average delay $\delta_{i,\text{avg}}^{(k)} = \sum_{\delta\in\mathbf{\Delta}_i^{(k)}} \delta \cdot \eta_{\mathbf{\Delta}_i}^{(k)}(\delta) \Big/ \eta_{i,\text{tot}}^{(k)}$ for each control application Λ_i. The simulation is terminated at the first simulated time-instant $k'h_\mathbf{\Lambda}$ ($k' > 1$) where the following condition is true: $\left|\delta_{i,\text{avg}}^{(k')} - \delta_{i,\text{avg}}^{(k'-1)}\right| \Big/ \delta_{i,\text{avg}}^{(k'-1)} < \zeta_{\text{sim}}$ for all $i \in \mathcal{I}_\mathbf{P}$. The parameter ζ_{sim} is given as an input; we have experimentally tuned this parameter to $\zeta_{\text{sim}} = 0.05$, which means that the simulation is stopped when the average delay has changed with less than 5 percent. After the simulated time $k'h_\mathbf{\Lambda}$, the approximation of each Δ_i^{sa} is given by the probability function $\widehat{\xi}_{\Delta_i}^{\text{sa}}(\delta) = \eta_{\mathbf{\Delta}_i}^{(k')}(\delta) \Big/ \eta_{i,\text{tot}}^{(k')}$. Given the approximate delay $\widehat{\Delta}_i^{\text{sa}}$ for each control loop, we proceed by computing the control law $\boldsymbol{u}_i$ for a constant delay $\text{E}\left\{\widehat{\Delta}_i^{\text{sa}}\right\}$. Finally, the controller cost J_i is computed, using the approximate probability function $\widehat{\xi}_{\Delta_i}^{\text{sa}}$. The cost of the priority assignment $\boldsymbol{\rho}$, given the periods $\boldsymbol{h}$, is $J_{\boldsymbol{\rho}|\boldsymbol{h}} = \sum_{i\in\mathcal{I}_\mathbf{P}} w_i J_i$.

The outer loop in Figure 6, which explores different priority assignments, is based on a genetic algorithm, similar to the period exploration in Section 6.1. We generate a population randomly and, in the iterations, we evaluate the cost of priority assignments (offsprings) that are generated with the crossover and mutation operators [12]. The population size is constant and equal to the number of tasks and messages in the system. The number of offsprings that are generated in each iteration is 25 percent of the population size, whereas mutation is applied with the probability 0.1. The exploration of priority assignments terminates when $J_{\text{avg}} < 1.1J_{\text{min}}$, where J_{avg} is the average cost of the current population and J_{min} is the cost of the current best priority assignment. The cost of the period assignment $\boldsymbol{h}$ is thus $J_\boldsymbol{h} = J_{\text{min}}$.

7. Experimental Results

We have run experiments to study the performance improvements that can be achieved with our control–scheduling codesign approach. We have defined a straightforward approach as a baseline for the comparison. In this approach, we design each controller Λ_i for a sampling period equal to the average of the set of available periods $\mathbf{H}_i$, and for a delay equal to the sum of average task execution times in the control application. For the implementation, we do the following steps:

1. Assign the period of each control application Λ_i to the smallest period in the set of available periods $\mathbf{H}_i$.
2. Schedule the system (depends on the scheduling policy).
   (a) Static-cyclic scheduling: Schedule the executions and communications as soon as possible, taking into account the schedule constraints.
   (b) Priority-based scheduling: Assign priorities rate-monotonically such that a task with a smaller period has a higher priority than a task with a larger period. The message priorities are assigned in the same way.
3. If the system is not schedulable, do the following steps:
   (a) If, for each application, the current period is the largest in the original set of available periods, the straightforward approach terminates.
   (b) For each of the control applications with highest utilization, if the current period is not the largest in the original set of available periods, then remove it from $\mathbf{H}_i$. Go to step 1.

Thus, the straightforward approach takes designed controllers and produces a schedulable implementation for as small controller periods as possible, but does not further consider the control quality.

To investigate the efficiency of period exploration and appropriate scheduling, we have defined two semi-straightforward approaches. For the first approach, straightforward period assignment, periods are assigned as in the straightforward approach. The scheduling, however, is performed according to our proposed approach, which integrates control-law synthesis. For the second approach, straightfor-

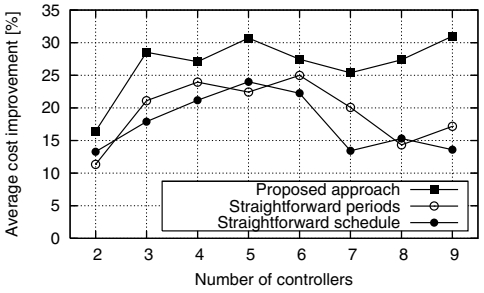

Figure 7. Improvements for static scheduling

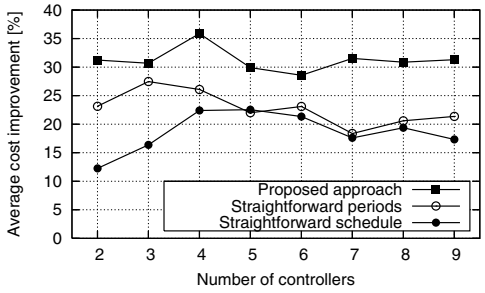

Figure 8. Improvements for priority scheduling

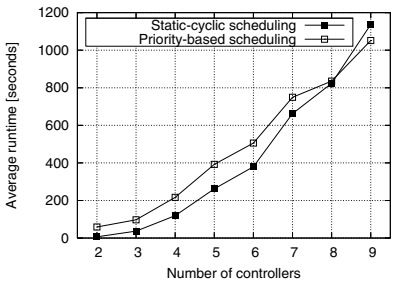

Figure 9. Runtimes for the optimization

ward scheduling, periods are chosen according to our proposed genetic algorithm-based approach, but the scheduling is done according to Step 2 in the straightforward approach.

For the evaluation, we created 130 benchmarks with varying number of plants (controllers). We used benchmarks with 2 to 9 plants that were chosen randomly from a database with inverted pendulums, ball and beam processes, DC servos, and harmonic oscillators [17]. For each plant, we generated a control application with 3 to 5 tasks with data dependencies. Thus, the number of tasks in our benchmark set varies from 6 to 45. The tasks were mapped randomly to platforms consisting of 2 to 7 computation nodes. For each controller, we generated 6 available periods, based on common rules of thumb [17]. Based on the average values of these periods, we generated the execution and communication times of the tasks and messages to achieve maximum node and bus utilizations between 40 and 80 percent. For the tasks, we considered uniform execution-time distributions.

For each benchmark, we have run our proposed approach for both static-cyclic scheduling and priority-based scheduling of the tasks and messages[1]. We have also run the two semi-straightforward approaches. For each of the three approaches, we obtained a final cost J_{approach}. We were interested in the relative cost-improvements $(J_{\text{SF}} - J_{\text{approach}})/\, J_{\text{SF}}$, where J_{SF} is the cost obtained with the straightforward approach. The average cost-improvements for static-cyclic scheduling and priority-based scheduling are depicted in Figures 7 and 8, respectively. In each figure, the vertical axis is the average cost improvement in percent for all benchmarks, with the number of controllers given on the horizontal axis. In Figure 7, for example, the average relative cost-improvements of the straightforward period-exploration and straightforward scheduling for 9 controllers are 17.2 and 13.6 percent, respectively. Our proposed approach has, for the same case, an average cost-improvement of 31.0 percent. For a small number of controllers, we note that the semi-straightforward approaches give improvements close to the improvements by the integrated approach. For larger number of controllers, however, the design space becomes larger, and thus the semi-straightforward approaches perform worse. The results show that it is important to combine period exploration with integrated scheduling and control-law synthesis to obtain high-quality solutions.

We have measured the runtimes for the proposed integrated approach; all experiments were run on a PC with a quad-core CPU running at frequency 2.2 GHz, 8 Gb of RAM, and running Linux. In Figure 9, we show, for both static-cyclic and priority-based scheduling, the average runtime in seconds as a function of the number of controllers. It can be seen that the complex optimization, involving period assignment, scheduling (priority assignment), controller design, and cost computation, can be run in our framework in less than 19 minutes for large systems of 9 controllers and 45 tasks.

[1]In the experiments, we have considered a TTP bus [6] for the static-cyclic scheduling case and a CAN bus [3] for the priority-based scheduling case. Our implementation also supports the FlexRay protocol [5].

8. Conclusions

Scheduling and communication in distributed embedded control systems lead to timing behaviors (delay and jitter) that degrade the overall control performance. In the context of both static-cyclic scheduling and priority-based scheduling, we have presented a control–scheduling co-design approach that integrates task and message scheduling with controller design (control-period exploration and control-law computation). The experimental results show that such an integrated design flow is essential to achieve high control-performance of control applications on distributed execution platforms.

References

[1] K. R. Apt and M. G. Wallace. *Constraint Logic Programming using ECL^{i}PS^e*. Cambridge University Press, 2007.

[2] M. M. Ben Gaid, A. Cela, and Y. Hamam. Optimal integrated control and scheduling of networked control systems with communication constraints: Application to a car suspension system. *IEEE Transactions on Control Systems Technology*, 14(4):776–787, 2006.

[3] R. Bosch GmbH. *CAN Specification Version 2.0*. 1991.

[4] M. Di Natale and J. Stankovic. Scheduling distributed real-time tasks with minimum jitter. *IEEE Transactions on Computers*, 49(4):303–316, 2000.

[5] FlexRay Consortium. *FlexRay Communications System. Protocol Specification Version 2.1*. 2005.

[6] H. Kopetz. *Real-Time Systems*. Kluwer Academic, 1997.

[7] B. Lincoln and A. Cervin. Jitterbug: A tool for analysis of real-time control performance. In *Proceedings of the 41st IEEE Conference on Decision and Control*, pp. 1319–1324, 2002.

[8] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):47–61, 1973.

[9] J. C. Palencia Gutiérrez and M. González Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pp. 26–37, 1998.

[10] P. Pop, P. Eles, Z. Peng, and T. Pop. Analysis and optimization of distributed real-time embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, 11(3):593–625, 2006.

[11] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei. Timing analysis of the FlexRay communication protocol. *Journal of Real-Time Systems*, 39(1–3):205–235, 2008.

[12] C. R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, 1993.

[13] S. Samii, S. Rafiliu, P. Eles, and Z. Peng. A simulation methodology for worst-case response time estimation of distributed real-time systems. In *Proceedings of the Design, Automation and Test in Europe Conference*, pp. 556–561, 2008.

[14] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin. On task schedulability in real-time control systems. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pp. 13–21, 1996.

[15] B. Wittenmark, J. Nilsson, and M. Törngren. Timing problems in real-time control systems. In *Proceedings of the American Control Conference*, pp. 2000–2004, 1995.

[16] K. E. Årzén, A. Cervin, J. Eker, and L. Sha. An introduction to control and scheduling co-design. In *Proceedings of the 39th IEEE Conference on Decision and Control*, pp. 4865–4870, 2000.

[17] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems*. Prentice Hall, 1997.