Design Optimization and Synthesis of FlexRay Parameters for Embedded Control Applications

Soheil Samii[1], Petru Eles[1], Zebo Peng[1], Anton Cervin[2]
[1]Department of Computer and Information Science
Linköping University, Sweden
[2]Department of Automatic Control
Lund University, Sweden

***Abstract*—FlexRay is a popular communication protocol in modern automotive systems with several computation nodes and communication units. The complex temporal behavior of such systems depends highly on the FlexRay configuration and influences the performance of running control applications. In our previous work, we presented a design framework for integrated scheduling and design of embedded control applications, where control quality is the optimization objective. This paper presents our extension to the design framework to handle FlexRay-based embedded control systems. Our contribution is a method for the decision of FlexRay parameters and optimization of control quality.**

***Keywords*-control performance, communication synthesis, design optimization, embedded control systems, FlexRay**

I. Introduction and Related Work

FlexRay [3] is a communication protocol developed for high-speed communication in automotive computer networks. The bus cycle of FlexRay comprises a static and a dynamic phase. The static phase is based on a classical TDMA scheme and offers predictable communication for safety-critical applications with strict timing constraints, whereas the dynamic phase implements flexible TDMA that is used by control applications. The main feature (related to temporal behavior) of the FlexRay protocol is the combination of classical TDMA with an adaptive communication scheme in the dynamic segment.

FlexRay-based systems exhibit complex temporal behavior [9], [4]. As opposed to safety-critical applications where worst-case response times are crucial, embedded control applications with periodic execution are sensitive to sampling–actuation delays of the controller executions and their jitter. The delay distribution is thus an important factor in the quality provided by embedded control applications. The selection of parameters for FlexRay communication impacts the temporal behavior of the applications and thus the control quality provided by embedded FlexRay-based control systems.

That delay and jitter have a negative impact on the control quality has been established as an important issue for embedded control systems [16], [17], [2], [7], [8]. Methods have been proposed to compute the control quality, given the delay characteristics of the control loop [5]. Integrated methods for system scheduling and control design (sampling-period selection and control-law synthesis) have also been proposed—however, mostly focusing on uniprocessor control systems. Sampling-period optimization for uniprocessor systems with multiple control loops and priority-based scheduling was first studied by Seto et al. [15] and later improved by Bini and Cervin [1] by considering control delays in the optimization. Rehbinder and Sanfridson proposed optimal control strategies for static scheduling of controllers on a uniprocessor system [11].

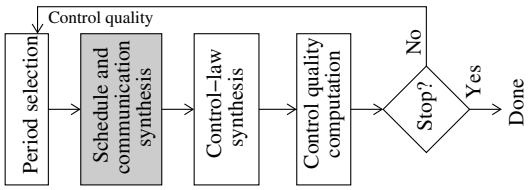

Figure 1. Overview of our design tool with three main components: (1) synthesis of controller periods, (2) schedule and communication synthesis, and (3) synthesis of control laws. The underlying optimization targets the overall control quality.

In our previous work, we proposed a framework [12] for scheduling and synthesis of multiple control applications on computer networks with TTP or CAN communication. Figure 1 shows a general overview of our design tool, featuring an optimization loop for the decision of sampling periods for each control application in the system. In each iteration, for a given assignment of periods, we synthesize execution schedules and communications. We presented solutions [12] for time-triggered systems with static schedule tables for task executions and message transmissions, as well as event-driven systems with priority-based CAN communication. The results of the scheduling and communication synthesis step are the delay characteristics of the control applications. These delays are given as inputs to the next step for control-law synthesis. The control laws determine how the control signals are computed based on the measured outputs of the controlled plants. The evaluation of a solution is based on the computation of the control quality, which is the optimization objective and which decides whether to terminate the optimization or not.

This paper addresses the synthesis step highlighted in gray in Figure 1 for embedded control systems with fixed-priority scheduling of computation tasks and communication of messages according to the FlexRay

protocol. To our knowledge, optimization of FlexRay parameters in the context of integrated control design and scheduling has not been addressed in literature. Specifically, our contribution is optimization of priorities and FlexRay frame identifiers, where control quality is the optimization objective. Our proposed solution has been integrated in our design tool and enables developers to design and optimize distributed embedded control systems that implement the FlexRay communication protocol.

II. System Model

A. Plant Model

Given is a set of plants $\mathbf{P}$, which are connected by sensors and actuators to a computation platform with several computation nodes on a FlexRay bus. Let us introduce the set $\mathcal{I}_{\mathbf{P}}$ to index the elements in $\mathbf{P}$. A plant $P_i \in \mathbf{P}$ is modeled as a linear state system

$$\dot{\boldsymbol{x}}_i = A_i \boldsymbol{x}_i + B_i \boldsymbol{u}_i + \boldsymbol{v}_i, \tag{1}$$

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 given random white-noise process. In typical applications, all components of the state $\boldsymbol{x}_i$ cannot be measured by the available sensors. The plant outputs $\boldsymbol{y}_i$ that can be measured are modeled as

$$\boldsymbol{y}_i = C_i \boldsymbol{x}_i + \boldsymbol{e}_i, \tag{2}$$

where the vector $\boldsymbol{y}_i$ is the output of plant P_i and $\boldsymbol{e}_i$ is measurement noise (e.g., due to imperfect sensors). The output $\boldsymbol{y}_i$ is measured and sampled periodically and is used by a controller to produce the control signal $\boldsymbol{u}_i$. The measurement noise $\boldsymbol{e}_i$ is modeled as white-noise process. The control problem is to control each plant in the presence of plant disturbances and measurement noise. The control signal is updated at discrete time instants and is held constant between updates [18]. The actual controller is a design objective and is an output of our design framework. 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 Equations 1 and 2, 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).

B. Platform Model

The execution platform consists of a set of computation nodes $\mathbf{N}$, indexed by $\mathcal{I}_{\mathbf{N}}$, that are connected by communication controllers to a bus. Figure 2 shows a system with two nodes (N_1 and N_2) connected by communication controllers (denoted CC in the figure) to a bus. The communication in the system is conducted according to the FlexRay protocol with a periodic bus cycle composed of a static and dynamic phase of given length. The static phase is divided into static slots of equal lengths. Each static slot is assigned one node; thus, a node has a set of static slots (possibly no slots assigned) and can send messages in the static phase only in those slots. Figure 4 shows an example of a FlexRay bus cycle with two static slots in the static phase.

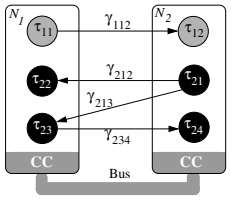

Figure 2. Example with two applications on two nodes. Tasks are depicted as circles, whereas messages between two tasks are depicted with arrows.

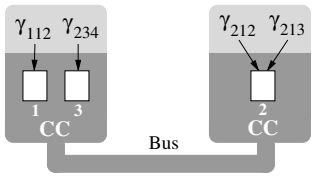

Figure 3. Dynamic-slot assignment. Messages are assigned frame identifiers that decides the waiting queue of the message and the dynamic slot used for its transmission.

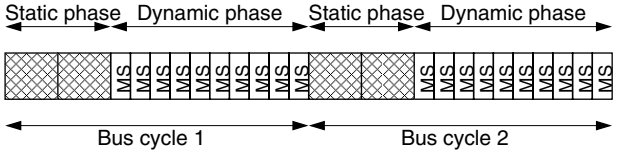

Figure 4. FlexRay bus cycle. The bus cycle is composed of a static TDMA phase and a dynamic phase in which communication is done in minislots.

The dynamic phase is divided into minislots. The bus cycle in Figure 4 has 10 equal-length minislots (denoted MS) in the dynamic phase. The communication in the dynamic phase is conducted in dynamic slots (as opposed to static slots of fixed size in the static phase). The slots in the dynamic phase are of varying size depending on whether messages are sent in certain minislots and the number of minislots that are needed to send a particular message. The maximum number of dynamic slots n_{DYN} is given as a part of the FlexRay configuration. Each dynamic slot is assigned one node; thus, given is a function $\mathsf{dyn} : \{1, \ldots, n_{\mathrm{DYN}}\} \longrightarrow \mathbf{N}$ that assigns a computation node to each dynamic slot. Let us also introduce the function $\mathsf{dyn}^* : \mathbf{N} \longrightarrow 2^{\{1,\ldots,n_{\mathrm{DYN}}\}}$ that gives the set of dynamic slots assigned to a node—thus, $\mathsf{dyn}^*(N_d) = \{\delta \in \{1, \ldots, n_{\mathrm{DYN}}\} : \mathsf{dyn}(\delta) = N_d\}$.

C. Application Model

Each plant is controlled by a set of software tasks that sample the plant outputs, compute control signals, and actuate them periodically. For each plant P_i, we have a control application $\Lambda_i = (\mathbf{T}_i, \mathbf{\Gamma}_i)$ modeled as a directed acyclic graph in which the vertices are tasks $\mathbf{T}_i$ and the edges $\mathbf{\Gamma}_i$ represent data dependencies between tasks. We denote the set of control applications with $\mathbf{\Lambda}$, which is indexed by $\mathcal{I}_{\mathbf{P}}$. In Figure 2, 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.

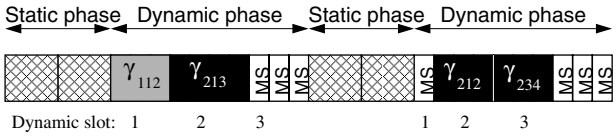

Figure 5. Communication in dynamic slots. Dynamic slots are varying in size depending on the size of the message to be sent. The length of a dynamic slot is equal to the length of a minislot if no message is sent.

The tasks of each application $\Lambda_i \in \mathbf{\Lambda}$ are released for execution periodically with the period h_i. An edge $\gamma_{ijk} = (\tau_{ij}, \tau_{ik}) \in \mathbf{\Gamma}_i$ models a message between two tasks. In Figure 2, the two applications Λ_1 and Λ_2 control the two pendulums P_1 and P_2, respectively. For example, task τ_{11} on node N_1 samples the pendulum position periodically and sends it on the FlexRay bus to task τ_{12}, which in turn computes and actuates the control signal.

Each task is mapped to a computation node in the network. The tasks on a node are scheduled according to given static priorities [6]. The mapping is given by a function $\mathsf{map} : \mathbf{T_\Lambda} \longrightarrow \mathbf{N}$, where $\mathbf{T_\Lambda} = \cup_{i \in \mathcal{I}_\mathbf{P}} \mathbf{T}_i$ is the set of all tasks in the system. 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 $\mathbf{\Gamma}_{\text{bus}} = \{\gamma_{ijk} = (\tau_{ij}, \tau_{ik}) \in \mathbf{\Gamma}_i : \mathsf{map}(\tau_{ij}) \neq \mathsf{map}(\tau_{ik}), i \in \mathcal{I}_\mathbf{P}\}$. The communication time of a message is given. The execution time of a task is typically not constant and is given as a distribution of execution times between with the best-case and worst-case execution times.

III. FlexRay Communication and Motivation

Let us consider message transmission in the dynamic phase of the FlexRay bus cycle (see Figure 4). The dynamic phase has a given length and consists of a given number of minislots (the length of a minislot is implicit). The maximum number of dynamic slots within one dynamic phase is n_{DYN}. Section II-B defines these parameters and the dynamic-slot mapping to computation nodes as a function dyn. In Figure 3, we show the communication subsystem of the example in Figure 2. The first node has two dynamic slots assigned (slots 1 and 3), whereas the second node can send messages only in the second dynamic slot. There are three dynamic slots in total. A node has in its communication controller a message queue for each assigned dynamic slot. The message queues are illustrated with white rectangles in Figure 3. When a task completes execution and produces a message to a task on another computation node, this message is transferred to the designated queue. The communication controller sends the message on the bus when the corresponding dynamic slot of the queue starts.

Each message $\gamma_{ijk} = (\tau_{ij}, \tau_{ik}) \in \mathbf{\Gamma}_{\text{bus}}$ on the bus is assigned a frame identifier f_{ijk}. The message is sent in the dynamic slot indicated by the assigned frame identifier. The frame identifier must be assigned a value that corresponds to a dynamic slot available to the sending computation node—thus, $f_{ijk} \in \mathsf{dyn}^*(\mathsf{map}(\tau_{ij}))$. Message γ_{112} in Figure 3 is assigned frame identifier 1 and is thus sent in the first dynamic slot. Messages that are assigned the same frame identifier are ordered by priorities in the corresponding message queue. Messages γ_{212} and γ_{213} have been assigned the same frame identifier. In this example, we consider that γ_{213} has the higher priority. The figure shows the messages γ_{112}, γ_{212}, γ_{213}, and γ_{234}. The arrows show the message queue (frame identifier) of each message.

Let us consider a runtime communication example. Task τ_{11} is executed on node N_1 and τ_{21} is executed on node N_2. The other tasks are waiting for messages. Both tasks execute and finish during the static phase. When the dynamic phase starts, messages γ_{112} (to task τ_{12}), γ_{212} (to task τ_{22}), and γ_{213} (to task τ_{23}) are in the corresponding message queues in the communication controllers and are ready to be transmitted. Message γ_{112} has frame identifier 1 and is therefore sent in the first dynamic phase as shown in Figure 5. The message occupies three minislots, which means that, in this bus cycle, the first dynamic slot comprises three minislots.

The second dynamic slot starts at the end of the third minislot. As Figure 3 shows, messages γ_{212} and γ_{213} have the frame identifier 2. Message γ_{213} has higher priority than γ_{212} and is therefore the message that is transmitted in the second dynamic slot. The length of the message is four minislots. When γ_{213} is received on N_1, task τ_{23} starts its execution. The third dynamic slot starts at the end of minislot 7. The only message that is assigned frame identifier 3 is γ_{234} on node N_1. In this first bus cycle, however, nothing is sent in the third dynamic slot, since τ_{23} did not yet finish execution (the slot comprises one minislot).

Let us consider that τ_{23} finishes its execution during the static phase of the second bus cycle. At the start of the second dynamic phase, messages γ_{212} and γ_{234} (from task τ_{23}) are ready for transmission. The first dynamic phase is one minislot, because no message with frame identifier 1 is available in the queue. Message γ_{212} is sent in the second dynamic phase and occupies 3 minislots. Message γ_{234} is transmitted in the third dynamic slot. A message is transmitted only if sufficient number of minislots remain until the end of the dynamic phase (the message is delayed to the next bus cycle otherwise).

The delay characteristics depend highly on the assignment of frame identifiers to the messages. This in turn impacts the achievable performance of the running control applications. Figure 6 shows two histograms of control delays, corresponding to two different assignments of frame identifiers. On the horizontal axis, we depict the control delay, whereas on the vertical axis, we show the number of times a certain control delay occurred, relative

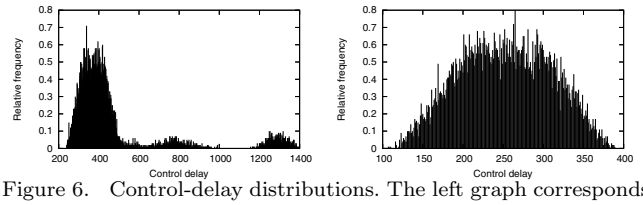

Figure 6. Control-delay distributions. The left graph corresponds to an assignment of FlexRay frame identifiers that lead to large delays and variations. Another assignment leads to smaller delays and variations in the right graph, leading to better control quality.

to the total number of delays observed during simulation of an example system. Each graph represents the control-delay distribution of a certain control application. The average delay for the left graph is 477, whereas for the right graph it is 249. In addition, we observe that the left graph shows larger delay variations, compared to the right graph. The frame identifiers that were used in the simulations corresponding to the right graph thus give better control quality, because the delays are smaller in terms of average delay and their variance. The sensitivity to delays and their amount of variation depends on the dynamics (Equation 1) of the plant under control.

IV. Control Quality

Let us consider a control application Λ_i of a plant P_i with given period h_i and control law $\boldsymbol{u}_i$. We measure the quality of controller Λ_i with a quadratic cost

$$J_i = \lim_{T\to\infty} \frac{1}{T} \mathrm{E}\left\{ \int_0^T \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix}^{\mathsf{T}} Q_i \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix} dt \right\}, \qquad (3)$$

which is a very common performance metric in control-systems literature [18]. The cost penalizes variations in the plant state $\boldsymbol{x}_i$. The cost includes a penalizing term for the control signal $\boldsymbol{u}_i$ since this influences the energy spent for actuation. The weight matrix Q_i is given by the designer and is used to indicate importance of having small variations in the plant state relative to the penalty of control actuations. The matrix Q_i is also used to indicate importance of a certain control application relative to other control applications in the system. Important to notice is that a smaller value of the cost J_i indicates a higher control quality.

The cost J_i depends on the sampling period h_i of control application Λ_i and its control law $\boldsymbol{u}_i$. More important in the context of this paper, J_i depends on the delays (computation and communication delays) between sampling and actuation. Given a controller and its delay characteristics, we compute the cost J_i with the Jitterbug toolbox [5].

V. Problem Formulation

A. Problem Inputs and Decision Variables

The inputs to our design-optimization problem are a set of physical plants $\mathbf{P}$ that are connected through sensors and actuators to a FlexRay network with a set of computation nodes $\mathbf{N}$. Given is also a set of control applications $\boldsymbol{\Lambda}$ and their mapping onto the computation nodes, all according to our description in Section II. Regarding the bus configuration, we consider the static and dynamic phases to have given lengths. The dynamic-slot assignment $\mathsf{dyn} : \{1, \ldots, n_{\mathrm{DYN}}\} \longrightarrow \mathbf{N}$ to computation nodes is given according to Section II-B.

The decision variables (outputs) of our problem are the period h_i and control-law $\boldsymbol{u}_i$ of each control application Λ_i, as well as a priority for each task and FlexRay frame identifiers and priorities of each message in the system. The periods and control laws are optimized according to our current framework [12]. The integration of FlexRay-parameter optimization is significant to achieve high control quality and is the contribution of this paper. Let us define what constitutes a solution in terms of priorities and frame identifiers. We shall define three parts: (1) the task priorities $\boldsymbol{\rho}$, (2) the frame identifiers $\boldsymbol{f}$ for messages, and (3) the message priorities $\boldsymbol{\omega}$. To define the three parts, we shall consider given an arbitrary bijection $\sigma_{\mathbf{T}} : \{1, \ldots, |\mathbf{T}_{\boldsymbol{\Lambda}}|\} \longrightarrow \mathbf{T}_{\boldsymbol{\Lambda}}$ that gives a total order of the set of all tasks $\mathbf{T}_{\boldsymbol{\Lambda}}$. The first part $\boldsymbol{\rho}$ (the task priorities) is a tuple $(\rho_1, \ldots, \rho_{|\mathbf{T}_{\boldsymbol{\Lambda}}|})$ where each $\rho_k \in \mathbb{N}$ is a natural number representing the priority of task $\sigma_{\mathbf{T}}(k)$. The tasks have unique priorities (i.e., $\rho_k \neq \rho_l$ for $k \neq l$).

To define the two remaining parts of a solution, consider any bijection $\sigma_{\boldsymbol{\Gamma}} : \{1, \ldots, |\boldsymbol{\Gamma}_{\mathrm{bus}}|\} \longrightarrow \boldsymbol{\Gamma}_{\mathrm{bus}}$ that orders the set of all messages $\boldsymbol{\Gamma}_{\mathrm{bus}}$ in the system. The second part $\boldsymbol{f}$ (the frame identifiers) assigns a frame identifier to each message and is a tuple $\boldsymbol{f} = (f_1, \ldots, f_{|\boldsymbol{\Gamma}_{\mathrm{bus}}|})$, where $f_k \in \mathbb{N}$ is the frame identifier of message $\sigma_{\boldsymbol{\Gamma}}(k)$. The frame identifiers must be assigned according to the dynamic-slot assignments to computation nodes. This means that the frame identifier f_k of message $\sigma_{\boldsymbol{\Gamma}}(k)$ must be assigned a value which corresponds to the frame identifier of the transmitting node in the network. Specifically, considering that τ_{ij} is the task that sends the message, we have $f_k \in \mathsf{dyn}^*(\mathsf{map}(\tau_{ij}))$. The third part $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_{|\boldsymbol{\Gamma}_{\mathrm{bus}}|})$ of the solution comprises the message priorities and are unique among messages that are assigned the same frame identifier. The message priorities are merely used to prioritize the transmission of messages with the same frame identifier.

B. Optimization Objective

Our objective is to maximize the control quality of all control applications in the system. We shall formulate the optimization objective in terms of the control cost defined in Equation 3. The cost to be minimized is the cumulative control cost of all control applications

$$J = \sum_{i \in \mathcal{I}_{\mathbf{P}}} J_i. \qquad (4)$$

Note that a smaller value of J indicates a higher overall control quality. Importance of a certain control application relative to other control applications is included implicitly in Equation 4 since this is specified by the designer through the weight matrix Q_i in Equation 3.

VI. Optimization Approach

Figure 7 shows a flowchart of our optimization approach. This flowchart, which is specific for FlexRay-based systems, corresponds to the gray box in our design framework in Figure 1. Thus, in this section, we consider the period h_i of each control application Λ_i given. Our optimization approach for the exploration of priorities and frame identifiers belongs to the class of stochastic optimization techniques and is based on genetic algorithms [10]. The gray rectangles in the flowchart show the different steps of our approach. The white rectangles with rounded corners show the inputs and results of each step. Initially, we generate randomly a population of solutions (priorities and frame identifiers) with the characteristics defined in Section V-A. Each member in the population thus gives an assignment of task priorities as well as frame identifiers and priorities of all messages. Our genetic algorithm-based solution is iterative and in each iteration it evaluates every member of the population (the dashed rectangle in Figure 7) in terms of the cost J defined in Equation 4. After this step, a new population is generated with the crossover and mutation operators for genetic algorithms. The crossover operator is used to combine two solutions into better solutions, based on the computed costs. The mutation operator is used to achieve diversification in the exploration phase, in order to avoid situations where the algorithm gets stuck in local optima. Members are chosen for crossover and mutation with certain probabilities. These probabilities for genetic algorithms are to be tuned experimentally (the crossover probability is usually significantly larger than the mutation probability). For the experiments presented in the next section, we have used 0.65 and 0.1 for the crossover and mutation probabilities, respectively. Crossover and mutation operators have been used in our previous work [14] to optimize average response times of real-time systems with FlexRay communication. After the crossover and mutation step, we have a new population that is evaluated in the next iteration. At each iteration during optimization, we keep track of the solution with the current best cost. The stopping condition that we have used in our implementation and experiments is that the optimization stops when the current best solution has been unchanged since five consecutive iterations of the genetic algorithm.

Let us consider a certain iteration and discuss how we evaluate each member (solution candidate) of a population. The evaluation of a member is shown in Figure 7 as

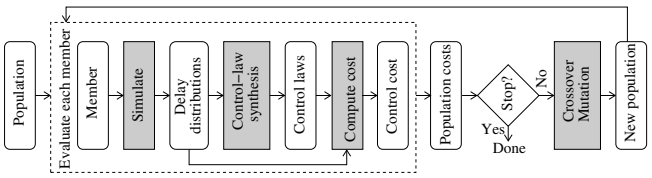

Figure 7. Our proposed optimization approach. The genetic algorithm for the assignment of priorities and frame identifiers is iterative and evaluates members (solution candidates) through simulations followed by computations of control quality.

the steps inside the dashed rectangle. The first step is to obtain the delay distribution of each control application, considering the priorities and frame identifiers given by the member (solution candidate). These distributions are obtained through simulation by configuring our in-house simulator [13] for distributed real-time systems with the periods of each control application Λ_i and the priorities and frame identifiers given by the member. Examples of possible outputs (delay distributions) of this simulation are shown in Figure 6. The next step is to synthesize the control law $\boldsymbol{u}_i$ of each application Λ_i. We use the Jitterbug toolbox [5] to synthesize the control law. The obtained control law and the delay distributions for the simulations are used to compute the cost J_i (Equation 3) of each application Λ_i (smaller J_i indicates higher control quality), leading to the final cost J of the evaluated member.

VII. Experimental Results

We performed experiments on 120 benchmarks to study the quality improvement that can be achieved by optimizing priorities and frame identifiers of tasks and FlexRay messages. The benchmarks have been created with an in-house tool [12] and comprise networks of 2 to 6 computation nodes, and 2 to 9 plants to be controlled from a database of inverted pendulums, ball and beam processes, DC servos, and harmonic oscillators [18]. Such plants are representatives of control problems and are used extensively for experimental evaluation in the real-time control systems domain. For each control application, we considered 3 to 5 tasks with data dependencies. The size of our benchmarks vary from 6 to 45 tasks and 4 to 36 messages on the FlexRay bus.

As a baseline of comparison, we defined a straightforward approach to the assignment of priorities and frame identifiers. Note that no other methods for control-quality optimization of FlexRay parameters has been proposed in literature. The straightforward approach is based on a rate-monotonic assignment of priorities and frame identifiers. This means that tasks in control applications with small periods are assigned high priorities, and vice versa. Similarly, messages that belong to control applications with small periods are assigned small frame identifiers from the set of frame identifiers available on the corresponding computation node. This is

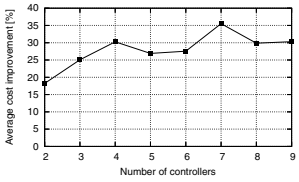

Figure 8. Cost improvements of our approach. The average improvement is 29.0 percent.

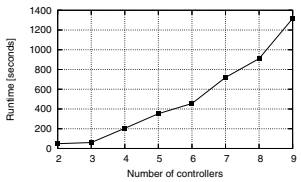

Figure 9. Optimization time. Large systems are synthesized in less than 25 minutes.

a typical design method that would be used by real-time control system developers in case no suitable, systematic optimization method is available. For each benchmark, we run our design tool with the proposed optimization approach in Section VI as well as our tool with the defined straightforward approach. We were interested in the relative cost-improvement $(J_{\mathrm{SF}} - J_{\mathrm{OPT}})/J_{\mathrm{SF}}$ of our approach, where J_{OPT} and J_{SF}, respectively, are the costs of the solutions obtained with our proposed optimization approach and the straightforward approach (we remind that a smaller cost indicates higher control quality). Figure 8 shows on the vertical axis the average cost improvements in percent for the benchmarks with the number of control applications on the horizontal axis. For example, the cost-improvement that can be achieved with our proposed approach is 31.3 percent for systems with 9 control applications. The average cost improvement considering all benchmarks is 29.0 percent.

Figure 9 shows the runtime of the proposed approach. All experiments were performed on a Linux PC with CPU frequency 2.2 Ghz and 8 Gb of main memory. We note that large FlexRay-based embedded control systems with 9 control applications, 45 tasks, and 36 messages can be synthesized in our design framework (period and control-law synthesis and optimization of priorities and frame identifiers) in less than 25 minutes to obtain design solutions that provide high control quality.

VIII. Conclusions

FlexRay is a communication protocol that is in use in many embedded control systems. The complex timing of such systems is influenced by the selection of FlexRay parameters and is a major factor in the overall control quality provided by the system. We presented a method for selection of frame identifiers and control-quality optimization for FlexRay-based control applications.

References

[1] E. Bini and A. Cervin. Delay-aware period assignment in control systems. In *Proceedings of the* 29^{th} *IEEE Real-Time Systems Symposium*, pp. 291–300, 2008.

[2] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. E. Årzén. How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. *IEEE Control Systems Magazine*, 23(3):16–30, 2003.

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

[4] A. Hagiescu, U. D. Bordoloi, S. Chakraborty, P. Sampath, P. V. V. Ganesan, and S. Ramesh. Performance analysis of FlexRay-based ECU networks. In *Proceedings of the* 44^{th} *Design Automation Conference*, pp. 284–289, 2007.

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

[6] 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.

[7] P. Martí, J. Fuertes, G. Fohler, and K. Ramamritham. Jitter compensation for real-time control systems. In *Proceedings of Real-Time Systems Symposium*, pp. 39–48, 2001.

[8] P. Martí, J. Yépez, M. Velasco, R. Villà, and J. Fuertes. Managing quality-of-control in network-based control systems by controller and message scheduling co-design. *IEEE Transactions on Industrial Electronics*, 51(6):1159–1167, 2004.

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

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

[11] H. Rehbinder and M. Sanfridson. Integration of off-line scheduling and optimal control. In *Proceedings of the* 12^{th} *Euromicro Conference on Real-Time Systems*, pp. 137–143, 2000.

[12] S. Samii, A. Cervin, P. Eles, and Z. Peng. Integrated scheduling and synthesis of control applications on distributed embedded systems. In *Proceedings of the Design, Automation and Test in Europe Conference*, pp. 57–62, 2009.

[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] S. Samii, Y. Yin, Z. Peng, P. Eles, and Y. Zhang. Immune genetic algorithms for optimization of task priorities and FlexRay frame identifiers. In *Proceedings of the* 15^{th} *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 486–493, 2009.

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

[16] 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.

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

[18] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems*. Prentice Hall, 3 edition, 1997.