

# Co-design of Cyber-Physical Systems via Controllers with Flexible Delay Constraints

Dip Goswami, Reinhard Schneider and Samarjit Chakraborty  
 Institute for Real-Time Computer Systems, TU Munich, Germany  
 (dip.goswami@tum.de, reinhard.schneider@rcs.ei.tum.de and samarjit@tum.de)

**Abstract—** In this paper, we consider a cyber-physical architecture where control applications are divided into multiple tasks, spatially distributed over various processing units that communicate via a shared bus. While control signals are exchanged over the communication bus, they have to wait for bus access and therefore experience a delay. We propose certain (co-)design guidelines for (i) the communication schedule, and (ii) the controller, such that stability of the control applications is guaranteed for more flexible communication delay constraints than what has been studied before. We illustrate the applicability of our design approach using the FlexRay dynamic segment as the communication medium for the processing units.

## I. INTRODUCTION

Systems with tight conjoining of and coordination between computational (cyber) units and physical resources are generally referred as *cyber-physical systems* [1], [2]. Regulation of the interacting dynamics of the computational and the physical parts of such systems necessitate careful co-design of the overall architecture. In this paper, we address design considerations to facilitate the interaction between control applications, that run on spatially distributed processing entities, and the scheduling algorithms for managing these entities.

Our work may be referred to as *control/communication architecture co-design*. In particular, our results show that it is possible to suitably design control applications that can work under more flexible communication delay constraints, rather than requiring all control messages to meet specified deadlines, which has been the case so far. The architecture design problem now boils down to designing the communication schedule that meets such flexible delay constraints. This allows for a wider range of schedules and communication architectures, compared to what is permissible with strict delay constraints. We quantify the notion of *flexibility* and show how control applications may incorporate this notion. Apart from our specific technical contributions which involve both controller design and (FlexRay) schedule synthesis, our work illustrates the importance and advantages of architecture/algorithm *co-design* in the context of cyber-physical systems.

The formal problem statement is formulated in Section II. The proposed design guidelines and the significance of the proposed methodology are described in Section III. In the subsequent Section IV, we illustrate our design proposal from the perspective of the communication architecture and the stability of control applications. The proposed design methodology is applicable to any cyber-physical control application. However, the significance of our approach is more prominent when the delay in the communication bus is time-varying. Hence, we illustrate the applicability of our framework considering the FlexRay *dynamic* segment as the communication medium. We developed a FlexRay control co-simulation framework, that is described in Section V, in order to simulate communication delays for different bus schedules and to analyze control stability. Finally, our simulation results are discussed in Section VI.

## II. PROBLEM FORMULATION

In this research, we consider a discrete-time control system of the form shown in (1).

$$x[k+1] = Ax[k] + Bu[k] \quad (1)$$

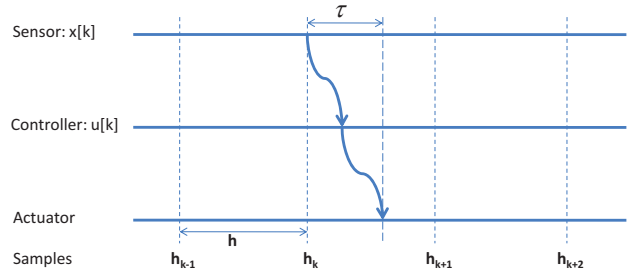


Fig. 1. Timing Diagram.

where  $x[k]$  is the  $n \times 1$  vector for *state variables* and  $u[k]$  is the *control input* to the system.  $A$  is a  $n \times n$  system matrix,  $B$  is a  $n \times 1$  vector and we assume that  $(A,B)$  is a *controllable* pair. The sampling time of the controller is a constant  $h$ . Such systems usually have two physical components: *actuators* (or plants)<sup>1</sup> and *sensors*. A *controller* is an algorithm to compute  $u[k]$  such that the states  $x[k]$  behave according to the designer's requirement. We consider a distributed architecture where the actuators and the sensors are spatially distributed and reside at different processing units (PUs) connected via a communication bus. The controller algorithm (or controller tasks) may run on the sensor or the actuator or on some other PU.

We assume the states  $x[k]$  are *measurable* and that their values are measured by the sensors. A *state-feedback controller*  $u[k]$  (as per (2)) utilizes the *feedback signals*  $x[k]$ . The whole purpose of any control design is to find suitable values of the controller gains  $K$  such that  $x[k]$  follows the designer's requirement. Another way to describe the same control problem is that we choose  $K$  such that the closed-loop dynamics  $(A + BK)$  is *stable* (i.e., all the Eigen values of  $(A + BK)$  are within the *unit circle*).

$$\begin{aligned} u[k] &= K_1 x_1[k] + K_2 x_2[k] \cdots K_n x_n[k] \\ \Rightarrow u[k] &= Kx[k] \end{aligned} \quad (2)$$

where  $K = [K_1 \ K_2 \ \cdots \ K_n]$  is known as *state-feedback gains*.

Now, we come to the issues related to a distributed control architecture. The control input  $u[k]$  in (2) at the  $k^{\text{th}}$  sampling instant (or  $t = h_k$ ) utilizes the values of the states exactly at  $t = h_k$ . However, we can see from Fig. 1 that both, the sensor data processing and the control input computation, consume a finite amount of time  $\tau$  before the control input is applied to the actuator. The value of  $\tau$  may be ignored when the actuator, the sensors and the controllers reside in the same task within a PU. However, in the case of distributed architectures,  $\tau$  can be arbitrarily large. In this work, we are interested in investigating design (or rather co-design) considerations for the communication schedule parameters and the controller gains such that  $x[k] \rightarrow 0$  as  $k \rightarrow \infty$  (i.e., *asymptotic stability*) for a range of values of  $\tau$ .

<sup>1</sup>In this work, we do not differentiate between actuators and plants. Because, the control inputs are directly applied to the actuators (e.g., the wheel of a car) and actuators may or may not be connected to some bigger plant (e.g., the car).

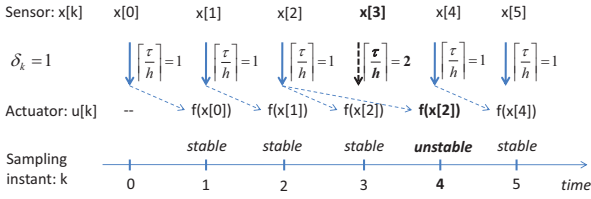


Fig. 2. Stable and unstable samples.

### A. Related Work

Effect of such communication delay, i.e.,  $\tau$ , on the stability of control applications is studied in [3], [4], [6] from a networked control systems perspective. One of the major challenges in modeling such communication delay in control theory is that the control signals experience time-varying delay in many industrial communication protocols (e.g., FlexRay, CAN, etc). In order to deal with such time-variation in the feedback loop, one noticeable design restriction found in the literature [4], [6] is that the maximum message delay should be less than one sample interval of the discrete-time control application under consideration (i.e.,  $0 < \tau_{max} < h$ ). This design restriction was relaxed in [3], [7] where the delay is upper-bounded by a certain value which can be more than one sample. In our proposed design method, such a restriction on the maximum delay is not imposed. Another noticeable consideration in these control-theoretic approaches is that the probability of occurrence of sensor-to-actuator delay is uniformly distributed between their minimum and maximum values. Moreover, it is assumed that the communication schedules are given and designer has no control over such communication schedules. Because of such restrictions, controller design often becomes overly conservative and fails to identify feasible designs (or are only applicable to stable plants or plants with marginal instability, as in the example provided in [7]). In contrast to such approaches, we study a co-design problem where the designer can adjust both the control application as well as the communication schedule. Some of previous work in this direction may be found in [2], [5] where control applications are stabilized using relatively simple assumptions on the communication bus. However, most industrial communication buses have more complex and unpredictable temporal behavior and the design methodology should be able to adapt to such complex behaviors. In this paper, we have illustrated the applicability of our design methodology using the FlexRay dynamic segment as the communicating medium.

## III. MAIN RESULTS

Suppose the controller gains  $K_\delta$  achieve asymptotic stability of the system (1) with the control law  $u[k] = K_\delta x[k - \delta_k]$ , i.e., the feedback signals are delayed by  $\delta_k$  samples. Hence, the system (1) will be asymptotically stable if all the feedback signals  $x[k]$ ,  $\forall k$  are delayed by  $(\delta_k - 1)h < \tau < \delta_k h$ , i.e.,  $\lceil \frac{\tau}{h} \rceil = \delta_k$  samples. However, there are sampling instants where  $\lceil \frac{\tau}{h} \rceil \neq \delta_k$ . The sampling instants with  $\lceil \frac{\tau}{h} \rceil = \delta_k$  are referred as *stable* samples and those with  $\lceil \frac{\tau}{h} \rceil \neq \delta_k$  are referred as *unstable* samples. We have explained this idea with an example in Fig. 2 where the actuator utilizes  $u[4]=f(x[2])$ , i.e., the feedback signal is two samples old. Now, we propose to design the communication schedule and the controller according to the following guidelines:

- 1) The communication bus parameters are chosen such that *most* of the samples are stable. We say that *most* of the samples are stable if the ratio between the total number of stable samples  $\mu_s$  and unstable samples  $\mu_u$  over  $(\mu_s + \mu_u)$  (which is sufficiently large) samples is lower-bounded by a *sufficiently* large positive integer  $M$ , i.e.,  $(\frac{\mu_s}{\mu_u} \geq M)$ . Hence, there might be unstable samples with arbitrarily large amount of delay (i.e.,  $\tau \gg \delta_k h$ ) and also samples with  $\tau \ll \delta_k h$ . However, their total number is upper bounded over certain time intervals.

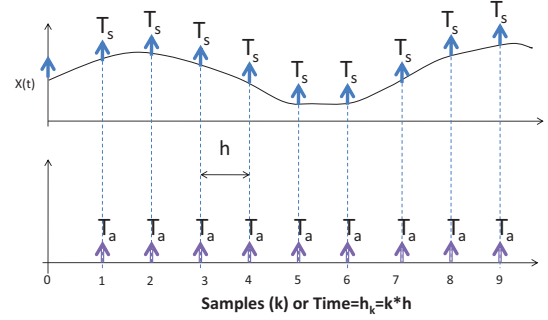


Fig. 3. Triggering of tasks  $T_s$  and  $T_a$ .

- 2) Controller gains  $K_\delta$  are designed to achieve asymptotic stability of the system (1) with the control law  $u[k] = K_\delta x[k - \delta_k]$ .

Therefore, we propose to design the communication schedule such that *most* of the sample delays vary within a range of one sample  $(\delta_k - 1)h < \tau < \delta_k h$  and design the controller gains based on such a range of delays (i.e.,  $\delta_k$  sample delays). Our claim is that the controller gains  $K_\delta$  (in (1)) stabilize even in the presence of certain samples for which  $\lceil \frac{\tau}{h} \rceil \neq \delta_k$  (proof is provided in the following sections). Note that our focus is not on designing controller gains or on synthesizing network schedules alone, rather we propose a design guideline for both of them which brings the designer the following advantages.

### A. Significance of our results

- The above design method does not require any knowledge of the worst-case delay experienced by the feedback signals. The analysis to find the worst-case delay for common industrial communication protocols such as CAN and FlexRay is relatively complex and computationally expensive. Moreover, controller design based on worst-case delays often leads to a pessimistic design.
- As we can see from [3], [4], [6], controller design based on time-varying delay often requires that the sensor-to-actuator delay *uniformly* varies between the best-case and the worst-case delays. Hence, it is assumed that the worst-case sensor-to-actuator delay might occur as frequently as any other possible sensor-to-actuator delays. However, this is not a realistic assumption for most of the commonly found communication buses. Rather, it is more meaningful to choose scheduler parameters that ensure, e.g., that the fraction of messages that experience the worst-case delay is bounded by a predefined constant. We follow this approach; our design method is based on a range of delay values that are *most likely* to occur (rather than *bounding* the worst-case delay). Hence, it is less pessimistic than previous approaches.

## IV. CONTROL/SCHEDULER CO-DESIGN

We start with the distributed control architecture described in Section II. The plant is represented by continuous time states  $x(t)$  and the task  $T_s$  reads the continuous states  $x(t)$  at the sampling times  $t = h_k$  with  $k = \{0, 1, 2, \dots\}$ , and  $h_{k+1} - h_k = h$ . The sensor signal at  $t = h_k$  is represented by  $x[k]$ . The task  $T_c$  processes  $x[k]$  (or computes  $u[k]$ ), sends the output to the task  $T_a$ , which applies the control input to the actuator/system/plant. The communication among the tasks  $T_a$ ,  $T_c$  and  $T_s$  can either be via the communication bus or directly, depending on the task mapping. We propose to design the network such that the tasks  $T_c$  and  $T_a$  are triggered synchronously (Fig. 3). In other words, both the tasks  $T_c$  and  $T_a$  are triggered at  $t = h_k$  with  $k = \{0, 1, 2, \dots\}$ . In the following, we illustrate the effects of such a design choice. Due to communication

delay,  $x[k]$  gets delayed by  $\tau$  time units to reach the task  $T_a$  (Fig. 1). However, the earliest time at which task  $T_a$  triggers is  $t = h_{k+1}$ , i.e., at the next sampling instance. Therefore, the effective sensor-to-actuator delay is given by  $\delta_k = \lceil \frac{\tau}{h} \rceil$  samples. The task  $T_a$  applies control input  $u[k] = f(x[k - \delta_k])$  to the actuator. Therefore, any delay between  $0 < \tau < h$  is effectively one sample delay with the above architecture.

### A. Control Scheme

Given the above architecture, we assume that we have a network schedule and controller gains as per the design guidelines mentioned in Section III. We apply the control input  $u[k]$  according to (3). Hence, we apply the control input to the plant for the stable samples and we do not apply any control input for the unstable samples.

$$\begin{aligned} u[k] &= K_\delta x[k - \delta_k], \forall \text{ stable samples} \\ &= 0, \forall \text{ unstable samples} \end{aligned} \quad (3)$$

### B. Asymptotic Stability of Control Applications

We consider the system (1) and the controller (3). For the stable samples, the delayed control inputs are applied. Due to delay in the control input, certain additional (or augmented) states appear in the closed-loop system. The design of stable controller gain for such augmented system is not the focus of this work. Hence, we illustrate the idea with a small example for easier understanding of the presented design guidelines. Consider a first order system  $x[k+1] = 1.1x[k] + u[k]$ . Let us consider  $u[k] = -0.2x[k-1]$ , i.e., the input signal is delayed by one sample. When  $u[k]$  is applied, the augmented system is as (4). The new state  $x_0[k]$  is because of the delay in the input signal. We denote the augmented states as  $\tilde{x}[k] = \begin{bmatrix} x_0[k] \\ x[k] \end{bmatrix}$  and the augmented system matrix as  $A_a$ . Hence,  $\tilde{x}[k+1] = A_a \tilde{x}[k] + BK_\delta \tilde{x}[k] = (A_a + BK_\delta) \tilde{x}[k]$ . We denote the closed-loop system matrix as  $A_{cl} = (A_a + BK_\delta)$ . In this example,  $A_a = \begin{bmatrix} 0 & 1 \\ 0 & 1.1 \end{bmatrix}$  and  $A_{cl} = \begin{bmatrix} 0 & 1 \\ -0.2 & 1.1 \end{bmatrix}$ .

$$\begin{aligned} x_0[k+1] &= x[k] \\ x[k+1] &= -0.2x_0[k] + 1.1x[k]. \end{aligned} \quad (4)$$

As the control input with  $K_\delta$  stabilizes (see (1)),  $A_{cl}$  is stable (in this example, the Eigen values of  $A_{cl}$  are 0.23 and 0.87). On the other hand, for unstable samples,  $u[k] = 0$ . Therefore, the closed-loop system matrix is  $A_a$  for the unstable samples. Often, the original system matrix  $A$  (before augmentation) or  $A_a$  is unstable with some of the Eigen values being outside the unit circle. Moreover,  $A_a$  is unstable if the original system matrix  $A$  is unstable (it can be verified from the above example). Hence, the overall system has some stable (stable samples) and unstable (unstable samples) phases. We prove the stability of the overall system with the help of the Lyapunov theorem [8].

For Lyapunov-based asymptotic stability, one needs to show the existence of a Lyapunov function  $V(x[k])$  such that (a)  $V(0)=0$ , (b)  $V(x[k])>0, \forall k$ , and (c)  $V(x[k+1])-V(x[k])<0, \forall k$  [8]. Basically, the criterion (c) says that if the system energy function  $V(x[k])$  decreases monotonically then the system (or equilibrium point) is asymptotically stable. For the stable samples, the closed-loop system dynamics is given by  $\tilde{x}[k+1] = A_{cl} \tilde{x}[k]$  and  $A_{cl}$  is stable. We assume that the Lyapunov function for the overall system is  $V(\tilde{x}[k]) = \tilde{x}[k]^T P \tilde{x}[k]$  where  $P$  is a positive definite matrix (which is one of most commonly used Lyapunov functions for linear systems). The criterion (a) and (b) hold for the proposed Lyapunov function. We have to show that criteria (c) also holds in order to prove that the overall system with stable and unstable phases is asymptotically stable. Towards this, we have equation (5) (this is the discrete-time version of the criterion presented in [9]). As the

system is ‘‘stable’’ at the stable samples, there exist some solution of the positive definite matrix  $P$  satisfying  $A_{cl}^T P A_{cl} - P = -I$  [9] at the stable samples. Therefore,  $V(\tilde{x}[k+1]) - V(\tilde{x}[k]) < 0$  holds at the stable samples and the energy function  $V(\tilde{x}[k])$  decreases at these samples. However,  $V(\tilde{x}[k])$  might increase at the unstable samples if  $A_a$  is unstable as the closed-loop system is  $\tilde{x}[k+1] = A_a \tilde{x}[k]$  for the unstable samples. The idea is to show that the total decrease in energy at the stable samples is more than total increase in energy at the unstable samples. In that case,  $V(\tilde{x}[k])$  decreases over time and the overall system is asymptotically stable in the Lyapunov sense.

$$V(\tilde{x}[k+1]) - V(\tilde{x}[k]) = \tilde{x}[k]^T (A_{cl}^T P A_{cl} - P) \tilde{x}[k]. \quad (5)$$

Towards this, we choose communication schedules such that the total number of stable ( $\mu_s$ ) and unstable ( $\mu_u$ ) samples over a time interval is upper-bounded by a sufficiently large positive integer  $M$ . Decrease in energy in the  $k^{th}$  stable sample is given by  $V(\tilde{x}[k+1]) - V(\tilde{x}[k]) = -\tilde{x}[k]^T I \tilde{x}[k]$  (using (5) and  $A_{cl}^T P A_{cl} - P = -I$ ). Now, if we start with  $k = 0$  and  $\mu_s$  stable samples, total decrease in energy is given by (using  $\tilde{x}[k+1] = A_{cl} \tilde{x}[k]$ ),

$$\begin{aligned} \Delta E_{stable} &= - \sum_{k=0}^{\mu_s-1} \tilde{x}[k]^T I \tilde{x}[k] = -\tilde{x}[0]^T (I + A_{cl}^T A_{cl} + (A_{cl}^T A_{cl})^2 \\ &\quad + (A_{cl}^T A_{cl})^3 \dots + (A_{cl}^T A_{cl})^{\mu_s-1}) \tilde{x}[0]. \end{aligned} \quad (6)$$

In the unstable samples, total energy increase is given by (as  $\tilde{x}[k+1] = A_a \tilde{x}[k]$  for the unstable samples),

$$\begin{aligned} \Delta E_{unstable} &= \tilde{x}[\mu_s + \mu_u]^T P \tilde{x}[\mu_s + \mu_u] - \tilde{x}[\mu_s]^T P \tilde{x}[\mu_s] \\ &= \tilde{x}[\mu_s]^T (A_a^{\mu_u})^T P (A_a^{\mu_u}) \tilde{x}[\mu_s] - \tilde{x}[\mu_s]^T P \tilde{x}[\mu_s] \\ &= \tilde{x}[\mu_s]^T ((A_a^{\mu_u})^T P (A_a^{\mu_u}) - P) \tilde{x}[\mu_s] \\ &= \tilde{x}[0]^T ((A_{cl}^{\mu_s})^T ((A_a^{\mu_u})^T P (A_a^{\mu_u}) - P) A_{cl}^{\mu_s}) \tilde{x}[0] \end{aligned} \quad (7)$$

The total change in energy in every  $(\mu_s + \mu_u)$  samples is  $\Delta E = (\Delta E_{stable} + \Delta E_{unstable})$ . It can be noticed that  $\Delta E < 0$  if condition in (8) holds for some positive definite matrix  $P$ .

$$\begin{aligned} (I + A_{cl}^T A_{cl} + (A_{cl}^T A_{cl})^2 + (A_{cl}^T A_{cl})^3 \dots + (A_{cl}^T A_{cl})^{(\mu_s-1)} \\ - ((A_{cl}^{\mu_s})^T ((A_a^{\mu_u})^T P (A_a^{\mu_u}) - P) A_{cl}^{\mu_s}) > 0. \end{aligned} \quad (8)$$

Hence, if condition (8) holds for every  $(\mu_s + \mu_u)$  samples, the overall system with stable and unstable samples (or phases) is asymptotically stable in the sense of Lyapunov. In this research, we find the value of  $\frac{\mu_s}{\mu_u} = M$  either experimentally or by simulation (as described in the Section V).

The above proof holds when  $\mu_u$  unstable samples occur after  $\mu_s$  stable samples. However, the proof can easily be extended to any other orderings of unstable samples. The idea is to show that the system is asymptotically stable as long the total change in energy is negative over every  $(\mu_s + \mu_u)$  samples for any order of occurrence of stable and unstable samples.

## V. FLEXRAY CONTROL CO-SIMULATION FRAMEWORK

### A. FlexRay Protocol

The FlexRay communication protocol [11] is organized as a periodic sequence of communication cycles. Each cycle is of fixed length  $gdCycle$  and is indexed by a cycle counter that is incremented from 0 to 63 after which the counter is reset to 0. This

communication pattern that is repeated periodically is known as the 64-cycle matrix. Further, every cycle consist of (i) a mandatory static segment (ST), (ii) an optional dynamic segment (DYN), and (iii) a segment for clock synchronization which is referred to as Network Idle Time (NIT). In the following we will discuss the communication specification of the DYN segment of FlexRay.

**FlexRay dynamic segment:** The DYN segment is partitioned into equal-length *minislots* that are indexed by a *minislot counter* which starts counting from 1 up to  $n$  *minislots* in every cycle. Additionally, a *slot counter* counts the communication slots that indicate time windows for admissible message transmissions. Each FlexRay message  $m_i$  is assigned a static schedule  $(S_i, B_i, R_i)$  for uniquely specified transmission points. A message  $m_i$  can successfully be transmitted via the DYN segment if the following requirements are satisfied:

- the assigned slot number  $S_i \in S_{DYN}$  is equal to the current *slot counter* value of node  $N$ , where  $S_{DYN}$  is the set of available slot numbers in the DYN segment,
- the actual communication cycle is element of the *set of feasible cycles*  $\gamma_n \in \Gamma_i$  where  $\gamma_n = (B_i + n \times R_i) \bmod 64$  with  $n \in [0, 1, 2, \dots]$ ,  $R_i = 2^r$  for  $r \in [0..6]$  and  $B_i < R_i$ ,
- the *minislot counter* must not exceed the specified value of  $p_{LatestTx}$  of node  $N$ .

The base cycle  $B_i$  indicates the first cycle within the 64-cycle matrix and the cycle repetition rate  $R_i$  indicates the number of cycles that elapse between two consecutive allowable message transmissions. The parameter  $p_{LatestTx}$  denotes the highest *minislot counter* value for which a message transmission is allowed to begin for a certain node  $N$ . If a message  $m_i$  is ready for transmission and the above defined conditions are fulfilled then the *minislot counter* is incremented by multiple minislots (according to the message size  $c_i$ ) during the transmission of  $m_i$  whereas the *slot counter* holds the value of  $S_i$ . After successful transmission of  $m_i$  the *slot counter* is incremented with the next minislot. If the *minislot counter* exceeded  $p_{LatestTx}$  before  $m_i$  got access to the FlexRay bus, the message has to wait for the next admissible cycle  $\gamma_n \in \Gamma_i$  to be transmitted in slot  $S_i$ . In that case only one minislot is consumed. If no message matches a dedicated *slot counter* and *cycle counter* or there is no message ready for transmission, one minislot is consumed as well, while the *slot counter* is incremented.

**Message displacement:** To illustrate the communication paradigm of the DYN segment consider the example in Fig. 4. Let the cycle length be  $gdCycle = 5ms$ , the number of ST slots  $s = 4$  and the number of minislots in the DYN segment  $n = 8$ . The *slot counter* in the DYN segment starts counting communication slots from  $s + 1 = 5$ . Further, let four messages  $m_1, m_2, m_3$  and  $m_4$  be scheduled according to their tuples of the form  $(S_i, B_i, R_i)$ , e.g.,  $(5, 0, 4)$ ,  $(7, 0, 2)$ ,  $(7, 1, 2)$ ,  $(9, 0, 4)$ . The message sizes  $c_i$  are denoted in terms of minislots by  $c_1 = 3$  and  $c_2 = c_3 = c_4 = 2$ . Further, we consider the last minislot where a transmission may start for all messages as  $p_{LatestTx} = 7$ .

In the following, we are especially interested in the delay of message  $m_4$ . In cycle 0,  $m_1$  is not transmitted, i.e., one minislot is consumed, whereas  $m_2$  and  $m_4$  are being transmitted on the bus in their assigned slots  $S_2 = 7$  and  $S_4 = 9$  consuming 2 minislots each. At slot  $S_4 = 9$  the *minislot counter* value is equal to 6 which is smaller than the specified  $p_{LatestTx}$  value of 7. Consequently,  $m_4$  is allowed to be transmitted in that cycle. However, in cycle 4, message  $m_1$  is additionally transmitted to  $m_2$  which increases the workload on the bus by another  $c_1 - 1$  minislots. Thus, at slot  $S_4 = 9$  the *minislot counter* exceeds the specified value of  $p_{LatestTx}$  which results in a displacement of  $m_4$ , i.e.,  $m_4$  can not be transmitted in the current cycle and will be *displaced* to its next feasible cycle  $\gamma_n \in \Gamma_4$ , e.g., cycle 8. Therefore,  $m_4$  can experience delay of several cycle lengths, i.e., several sample delays, depending on the number of consecutive displacements. Note, that a transmission of  $m_3$  will never affect the transmission of  $m_4$  as  $m_3$ 's schedule only allocates

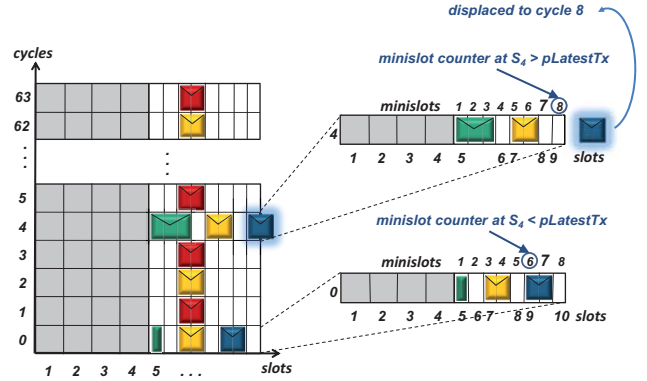


Fig. 4. FlexRay schedule.

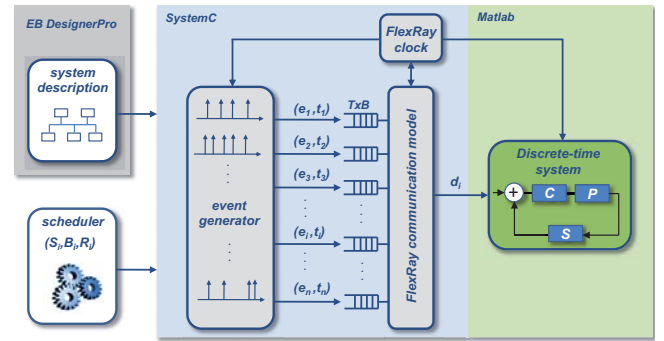


Fig. 5. FlexRay Control Co-Simulation Framework.

odd cycles in the 64-cycle matrix, i.e.,  $\Gamma_3 = [1, 3, 5, \dots, 63]$ , whereas  $m_4$  is assigned even cycles  $\Gamma_4 = [0, 4, 8, \dots, 60]$ .

## B. Simulator

**System Description:** We use the Elektrobit (EB) tresos Designer Pro tool [10] to determine the FlexRay bus configuration parameters such as  $gdCycle$ ,  $p_{LatestTx}$ , the proportion of the ST and DYN segment and other protocol related parameters. Additionally, message properties and schedule parameters of existing messages are imported to the simulation framework.

**Scheduler:** Our scheduler (see Fig. 5) synthesizes all *feasible* message schedules  $(S_i, B_i, R_i)$  for a message  $m_i$  that serves as an input to the simulator. With *feasible* message schedules we mean schedules that satisfy the FlexRay protocol (as previously discussed). In order to avoid buffer overflows in the FlexRay transmit buffers, we upper bound the admissible repetition rates by  $R_{max,i} = \min(2^{\lfloor \log_2(\frac{p_i}{gdCycle}) \rfloor}, 64)$ .

The simulation framework as depicted in Fig. 5 is made up of two main modules: the *FlexRay event simulator* in order to simulate communication delays, and the *discrete-time system model* to simulate the discrete-time system. The FlexRay simulator is implemented in SystemC and consists of several submodules: (i) the *FlexRay clock* provides the *FlexRay communication model* with the actual *slot counter*, *minislot counter* and *cycle counter* values, (ii) an *event generator* generates input event streams based on the system description and (iii) the *FlexRay communication model* implements the FlexRay specification and computes the message delays for the transmitted event streams. Further, the message delays serve as an input to the *discrete-time system model* in order to compute sensor-to-actuator delay and simulate the stability of the

system. In the following, we explain these modules in more detail.

**FlexRay clock:** At  $t = 0$ , the FlexRay clock is initialized with *cycle counter* equal to 0 and *slot counter* equal to 1. On starting the simulator the *cycle counter* is incremented from 0 to 63 after which it is reset to 0 again. During every cycle the *slot counter* is incremented according to event transmissions until the *minislot counter* value is equal to the last minislot of the DYN segment. Further, the FlexRay clock synchronizes the *event generator* with the global FlexRay time base.

**Event generator:** Every event  $e_i \in E$  is characterized by the tuple  $(S_i, B_i, R_i, p_i, o_i, [r_{min,i}, r_{max,i}], c_i)$  where  $S_i$  is the dynamic communication slot,  $B_i$  is the base cycle and  $R_i$  denotes the repetition rate. An event activation is specified by a period  $p_i$  and an initial offset  $o_i$  where  $o_i$  determines the point in time at which the first instance of  $e_i$  is triggered. The  $k^{th}$  instance  $e_i^k$  will be generated at  $t_i^k = o_i + k \times p_i$  and arrives at the transmit buffer (TxB) at  $t_i^* = t_i^k + r_i^*$  where  $r_i^* \in [r_{min,i}, r_{max,i}]$  denotes a response time that is randomly distributed between a lower and an upper bound  $r_{min,i} > 0$ ,  $r_{max,i} \geq r_{min,i}$  respectively. The message size  $c_i$  is denoted in terms of minislots.

**FlexRay communication model:** Further, every event  $e_i \in E$  is statically assigned a transmit buffer that is configured with a filter mask according to the assigned schedule parameters  $S_i, B_i$  and  $R_i$ . An event  $e_i$  generated at time stamp  $t_i$  is stored in the assigned buffer where it waits until it gets access on the bus. Several instances of  $e_i = \langle e_i^1, e_i^2, \dots, e_i^n \rangle$  are stored in a first-in-first-out (FIFO) order. The transmit buffer of any event  $e_i$  is requested for transmission on the FlexRay bus if (i) the *slot counter* of the FlexRay clock matches the slot  $S_i$ , (ii) the actual *cycle counter* matches  $\gamma_n \in \Gamma_i$  and (iii) the *minislot counter* is within the specified  $pLatestTx$  bound. In case the buffer is filled, the event at the head of the FIFO is removed and gets transmitted on the FlexRay bus. After successful transmission at time stamp  $t_i'$  the *slot counter* is incremented by  $c_i$  minislots. If the buffer is empty or no buffer is requested for transmission the *slot counter* is incremented with the next minislot. The delay  $d_i^k$  of message  $m_i$  at sample  $k$  is computed as the deviation of the event transmission time stamp  $t_i^{k'}$ , i.e., arrival of  $e_i^k$  at the receiver, to the event generation time stamp  $t_i^k$ , formally written as  $d_i^k = t_i^{k'} - t_i^k$ . Once the simulation has finished all delays  $d_i = \langle d_i^1, d_i^2, \dots, d_i^n \rangle$  among all event instances  $e_i = \langle e_i^1, e_i^2, \dots, e_i^n \rangle$  are passed to the *discrete-time system model* for further processing.

**Discrete-Time System Model:** The *discrete-time system model* is implemented in Matlab as a discrete time control system of the form (1). We consider a distributed control architecture as depicted in Fig. 6 with sampling period  $h$ . The sensor task  $T_s$  is triggered with offset  $o_s$ , the controller task  $T_c$  is processed on the same processor with offset  $o_c > o_s + r_{max,s}$  where  $r_{max,s}$  is the worst-case response time of  $T_s$ . Subsequently, the controller output is packetized in message  $m_c$  that is transmitted via the DYN segment of FlexRay with period  $p = h$  and time delay  $d$ . The actuator task  $T_a$  is triggered with an offset  $o_a = o_s$  and performs the input to the plant. The total end-to-end delay along the control path is computed as per  $\lceil \frac{\tau}{h} \rceil = \lceil \frac{o_c - o_s + d_c}{h} \rceil$ . Finally, the stability properties are observed for every feasible schedule that is synthesized according to the FlexRay protocol.

## VI. RESULTS

**System description:** The FlexRay configuration parameters have been specified using the EB Designer Pro tool. The cycle length is set to  $gdCycle = 5ms$  with ST segment of length  $2ms$  and 10 static slots. The rest of the cycle has been distributed to the DYN segment and NIT. Further, the DYN segment consists of 60 minislots where the duration of one minislot is  $0.05ms$ . The value  $pLatestTx$  was

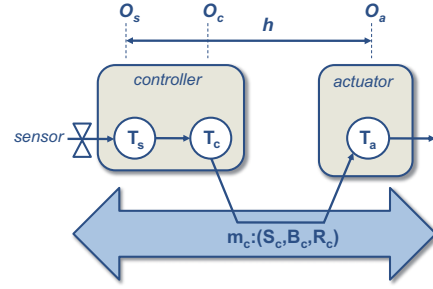


Fig. 6. Distributed control system under simulation.

TABLE I  
EXISTING FLEXRAY SCHEDULES.

index $i$	$(S_i, B_i, R_i)$	$o_i$ in ms	$p_i$ in ms	$[r_{min,i}, r_{max,i}]$	$c_i$
1	(11, 0, 2)	1	20	[0.7,2.7]	4
2	(12, 0, 2)	2	20	[0.7,2.7]	4
3	(14, 2, 4)	2	40	[1.7,6.7]	3
4	(16, 0, 4)	3	50	[1.7,6.7]	3
5	(17, 7, 8)	12	100	[1.7,6.7]	3
6	(17, 4, 8)	14	100	[1.7,6.66]	3
7	(20, 2, 4)	18	80	[2.7,10.7]	5
8	(25, 0, 2)	0.1	20	[0.7,2.7]	4
9	(25, 1, 4)	17	100	[6.7,26.7]	6
10	(28, 0, 4)	21	60	[2.8]	4
11	(30, 1, 2)	11	50	[1.7,6.7]	5
12	(33, 0, 2)	28	20	[1.4]	4
13	(34, 0, 1)	2	20	[0.7,2.7]	4
14	(38, 0, 2)	30	40	[1.3,5.3]	5

set to 50 for all messages in the network, i.e., the last minislot where a message transmission may begin is when *minislot counter* is equal to  $pLatestTx$ . In order to provide results of practical relevance we consider an existing FlexRay network with several messages being mapped on the DYN segment. The message properties are depicted in Table I, i.e., schedule parameters  $S_i, B_i, R_i$ , task offsets  $o_i$  and periods  $p_i$ , uniformly distributed response times between  $[r_{min,i}, r_{max,i}]$  and message sizes  $c_i$  in minislots. We simulate a distributed controller architecture as depicted in Fig. 6 with the following system properties:

- discrete-time control system of form as shown in (1), with  $A = [0.4 \ 0.60 \ 0.7; -0.56 \ -0.9 \ -0.6; -3.6 \ -1.2 \ -2.8]$ , and  $B = [0.1; 0.7; 0.5]$ . It can be noticed that the Eigen values of the open-loop system matrix  $A$  are  $[-1.5717 \ -1.4 \ -0.3283]$ . As two of the Eigen values are outside the unit circle, the open-loop system (original plant) is highly unstable.
- controller gains  $K_\delta = [-1.8622 \ -0.2858 \ -1.0355]$  are designed to achieve asymptotic stability of the system with control law  $u[k] = K_\delta x[k - \delta_k]$ , with  $\delta_k = 1$
- the minimum ratio of stable and unstable samples for which the closed-loop system is asymptotically stable with the above controller gains is experimentally found to be  $M_{ref} = 52$
- sampling period  $h = 40ms$
- task offsets  $o_s = o_a = 0.1ms$ ,  $o_c = 0.4ms > o_s + r_{max,s}$
- response time  $r_c^*$  for controller task  $T_c$  is uniformly distributed between  $[1.3ms, 5.3ms]$
- FlexRay bus schedules  $(S_c, B_c, R_c)$  for controller message  $m_c$  generated by the *scheduler*

For the purpose of our experiments we carried out 120 simulations with a different schedules  $(S_c, B_c, R_c)$  for  $m_c$  at each simulation run. The size of the message  $m_c$  is  $c_{15} = 4$  minislots. The period of the controller task  $T_c$  is same as the sampling interval  $h$ , i.e.,  $p_{15} = 40ms$ . The simulation time was set to  $100sec$  which corresponds to 2500 generated samples at a sampling period  $h = 40ms$ . During each simulation we plotted the distribution of sensor-to-actuator delay  $\tau$  and analyzed the asymptotic stability of the discrete-time system. We illustrate our observations for three example schedules that have been synthesized for  $m_c$ .

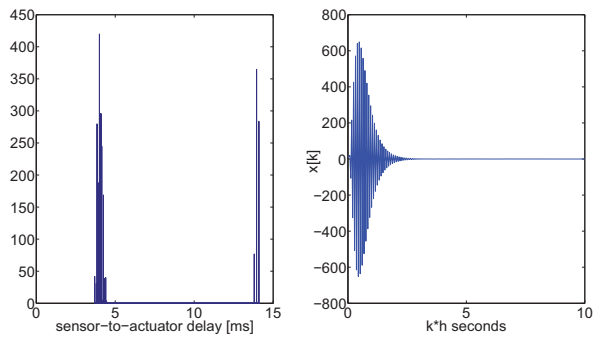


Fig. 7. Delay distribution for  $m_c : (40, 0, 2)$  and the corresponding  $x_1[k]$ .

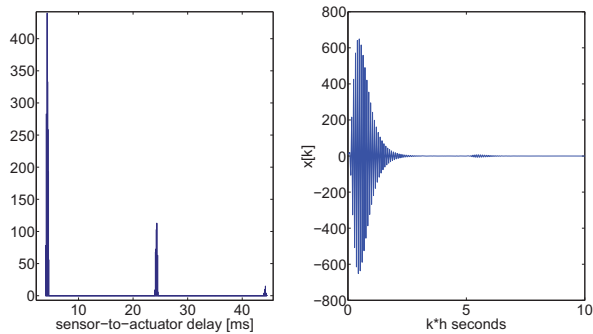


Fig. 8. Delay distribution for  $m_c : (45, 0, 4)$  and the corresponding  $x_1[k]$ .

**Example 1:** The controller message  $m_c$  has been assigned the schedule  $(40, 0, 2)$ . Fig. 7 illustrates the delay distribution for the sensor-to-actuator delay. The delay is distributed between  $\tau_{min} = 3.7ms$  and  $\tau_{max} = 14.1ms$  which corresponds to a constant sample delay of  $\lceil \frac{\tau}{h} \rceil = \delta_k, \forall k$ , i.e., all sample instances have been *stable*. The delays lie in two distinct regions. The distance between the two regions is around  $R_c \cdot gdCycle$  which is the time a message must wait in case (i) the message just misses its slot, and (ii) the message got *displaced* (as described in Section V). As expected, the system is asymptotically stable as  $x[k] \rightarrow 0$  for  $k \rightarrow \infty$  (see Fig. 7). As the time-variation of all the state variables  $x[k]$  are very similar, we show only  $x_1[k]$  to explain our results. We can notice some initial oscillation in  $x_1[k]$  because the original system is highly unstable. Nevertheless, the controller gains are able to achieve asymptotic stability.

**Example 2:** Here,  $m_c$  has been assigned the schedule  $(45, 0, 4)$ , i.e., a slot with higher slot number  $S_c$  and repetition rate  $R_c$  than in example 1 (which means lower priority). The corresponding delay distribution is depicted in Fig. 8. The sensor-to-actuator delay varies between  $\tau_{min} = 3.95ms$  and  $\tau_{max} = 44.5ms$ , i.e., there exists a sample  $k$  for which  $\lceil \frac{\tau}{h} \rceil > 1$ . In this example 36 *unstable* samples could be observed, i.e.,  $M = 70 > M_{ref}$ . Hence, the system is still stable as  $x[k] \rightarrow 0$  for  $k \rightarrow \infty$  (see Fig. 8).

**Example 3:** Finally, we observe the case where the number of *unstable* samples is high, e.g.,  $M = 11 < M_{ref}$ . This is depicted in Fig. 9 where  $m_c$  is assigned the schedule  $(48, 0, 4)$ . We observed 243 *unstable* samples where the sensor-to-actuator delay was distributed between  $\tau_{min} = 4.1ms$  and  $\tau_{max} = 84.5ms$ , i.e., some of the *unstable* samples even suffer from a delay of  $\lceil \frac{\tau}{h} \rceil = 3$ . In that case, asymptotic stability could not be achieved (see Fig. 9).

## VII. CONCLUDING REMARKS

In this paper we have shown that by appropriately designing control algorithms, important control performance metrics such as stabil-

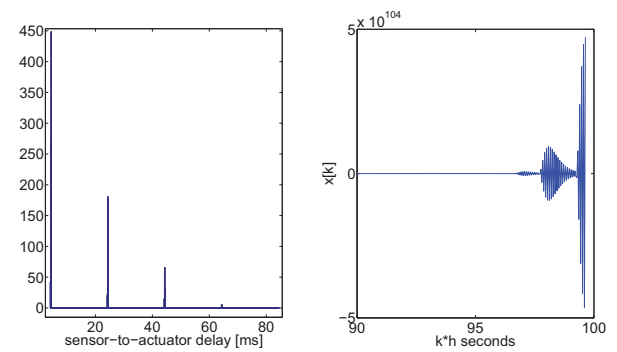


Fig. 9. System is unstable for  $m_c : (48, 0, 4)$  and the corresponding  $x_1[k]$ .

ity can still be guaranteed even when certain control messages violate their timing properties. This requires careful control/communication-architecture co-design, which was the focus of this paper. To illustrate our techniques, we chose the *dynamic segment* of a FlexRay bus as the communication medium. In contrast to current practice, where the time-triggered (and hence predictable) *static segment* of FlexRay is used for time-critical control algorithms, our work illustrates that careful system co-design allows for greater design flexibility. We believe that such “co-design” will be the hallmark of the cyber-physical systems design paradigm. As a part of future work, we plan to augment our methodology with a formal analysis of the dynamic segment of FlexRay.

## VIII. ACKNOWLEDGEMENT

This paper was written while the first author was an Alexander von Humboldt Research Fellow at TU Munich, Germany. The generous grant of the Alexander von Humboldt Foundation is gratefully acknowledged.

## REFERENCES

- [1] W. Wolf, “Cyber-physical Systems,” IEEE Computer, 42(3):88-89, March 2009.
- [2] F. Zhang, K. Szwaykowska, W. Wolf and V. Mooney, “Task Scheduling for Control Oriented Requirements for Cyber-Physical Systems,” pp. 47-56, Real-Time Systems Symposium 2008.
- [3] G. C. Walsh, H. Ye, and L. G. Bushnell, “Stability Analysis of Networked Control Systems,” IEEE Trans. on Control System Technology, pp. 438-446, vol. 10, no. 3, 2002.
- [4] M. E. M. B. 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 Trans. on Control System Technology, pp. 776-787, vol. 14, no. 4, 2006.
- [5] H. Voit, R. Schneider, D. Goswami, A. Annaswamy and S. Chakraborty, “Optimizing Hierarchical Schedules for Improved Control Performance,” International Symposium on Industrial Embedded Systems (SIES), Trento, Italy, 2010.
- [6] M. Yongguang, C. Wenying and L. Guangxiao, “Compensation of networked control systems with time-delay and data packet losses,” Chinese Control and Decision Conference, pp. 5609-5612, 2009.
- [7] X.M. Tang, J.S. Yu, “Stability Analysis for Discrete Time-Delay Systems,” Conference on Networked Computing and Advanced Information Management, 2008.
- [8] B. C. Kuo, “Automatic Control Systems,” Prentice Hall 1981.
- [9] R. Langerak and J. W. Polderman, “STools for Stability of Switching Linear Systems: Gain Automata and Delay Compensation,” IEEE Conference on Decision and Control and European Control Conference, 2005.
- [10] Elektrobit, “Elektrobit Tresos,” www.elektrobit.com.
- [11] FlexRay, “FlexRay Communications System Specifications,” www.flexray.com.