Re-engineering Cyber-Physical Control Applications for Hybrid Communication Protocols

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 multiple control applications are divided into multiple tasks, spatially distributed over various processing units that communicate over a bus implementing a *hybrid* communication protocol, i.e., a protocol with both *time-triggered* and *event-triggered* communication schedules (e.g., FlexRay). In spite of efficient utilization of communication bandwidth (BW), event-triggered protocols suffer from unpredictable temporal behavior, which is exactly the opposite in the case of their time-triggered counterparts. In the context of communication delays experienced by the control-related messages exchanged over the shared communication bus, we observe that a distributed control application is more prone to performance deterioration in *transient* phases compared to in the *steady-state*. We exploit this observation to re-engineer control applications to operate in two modes, in order to optimally exploit the *bi-modal* (time- and event-triggered) characteristics of the underlying communication medium. Depending on the *state* (transient or steady) of the system, both, the control inputs and the communication schedule are now switched. Using a FlexRay-based case study, we show that such a design provides a good trade-off between control performance and bus utilization.**

I. Introduction

Systems with tight conjoining of and coordination between computational (cyber) units and physical resources are generally referred as *cyber-physical systems* [1]. In the context of such systems, there have been quite a few recent research efforts addressing the *control/communication architecture co-design* problems aiming to improve the regulation of the interacting dynamics between the computational and the physical parts [2], [3], [4]. In this paper, we propose a *control/communication architecture co-design* framework specially targeted towards *hybrid* communication protocols like TTCAN [5] and FlexRay [6]. FlexRay is considered to be future de-facto standard for automotive communication systems. In this work, we therefore choose FlexRay as communication platform for the sake of illustration.

Hybrid communication protocols are a combination of the *time-triggered* and *event-triggered* paradigms. Communication activities in event-triggered protocols are triggered by the occurrence of specific events, whereas time-triggered protocols schedule communication activities at predetermined points in time, which are commonly referred to as *slots*. Both have their unique advantages and disadvantages. Applications solely communicating via either the time-triggered or the event-triggered schedules are likely to suffer from the shortcomings of the corresponding paradigms.

978-3-9810801-7-9/DATE11/©2011 EDAA

In this work, we aim to *re-engineer* control applications in order to optimally exploit the bi-modal characteristics (time- and event-triggered) of the underlying communication medium. When control-related signals are transmitted over the shared communication bus, they might experience a delay giving rise to *instability* and *performance degredation*. Although it is possible to avoid these by assigning time-triggered slots to all control-related signals, this suffers from inflexibility and poor communication BW utilization. The use of time-triggered communication becomes impractical and expensive especially when there are a large number of distributed control applications and each of them demands time-triggered communication slots. On the other hand, event-triggered communication schedules often result in poor control performance due to the unpredictable temporal behavior of control messages.

In this work, we propose to *switch* both the communication schedule and the control inputs in order to achieve a trade-off between control performance and bus BW utilization. Our scheme is based on the observation that communication delays during a *transient* phase cause more performance deterioration, compared to delays during the *steady* state of the controller. Therefore, we provide *shared* time-triggered communication slots (shared among multiple control applications) to the control messages during transient phases, and *switch* to event-triggered communication schedules once a steady state is reached. Such switching might intermittently happen throughout the operation if there are disturbances in the system. After the formal problem statement (Section II), we describe our scheme in Section III. Our co-simulation environment and simulation results are shown in Sections IV and V.

II. Problem formulation

Here, the control applications are partitioned (e.g, into sensor task T_s, actuator task T_a and controller task T_c) and mapped into multiple processing units (PUs). We consider a discrete-time control system of the form shown in (1).

$$x[k+1] \quad = \quad Ax[k] + Bu[k] \tag{1}$$

where $x[k]$ is the $n \times 1$ vector of *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. We consider a *state-feedback controller* $u[k] = Kx[k-\delta]$ where

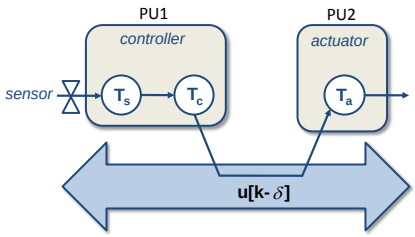

Fig. 1. The distributed control system.

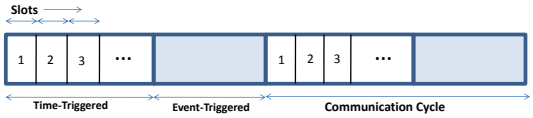

Fig. 2. Communication schedules: *time-triggered* and *event-triggered*.

K is the *state-feedback gain* [7] and δ is the communication delay. Designing a state-feedback controller $u[k]$ essentially boils down to a problem of finding suitable controller gains K such that $x[k]$ follows the designer's requirement. Typically, a designer needs to meet both *stability* and *performance* requirements of the closed-loop system. In this work, we have the following stability and performance considerations.

- We consider the notion of *asymptotic* stability, i.e., we want to ensure that $x[k] \to 0$ as $k \to \infty$.
- We consider the *settling time* - time interval to reach and remain (without external *disturbances*) within a specific error band of the reference signal - as the performance requirement. Moreover, we improve (minimize) the integral cost function of the input signal $u[k]$, i.e., $\sum_k u[k]^2$ and the integral cost function of the tracking error, i.e., $\sum_k x[k]^T x[k]$[1].

In context of the problem under consideration, we consider the communication protocol as shown in Fig. 2 where each communication cycle is divided into time-triggered and event-triggered segments. We classify communication schedules into two categories (Fig. 2): *time-triggered* (TT) and *event-triggered* (ET) schedules. First, the schedules which can ensure *zero delay* of the control-related messages are referred as *time-triggered* communication schedules[2]. For example, let us consider the *time-triggered* schedules where applications are allowed to send messages only at their assigned *slots* and the tasks are triggered *synchronously* with the bus, i.e., the messages are generated at the beginning of the corresponding slots and get transmitted immediately. On the other hand, the tasks are assigned *priorities* in order to arbitrate for the access to the *event-triggered* segment. In this case, the messages

[1]In this case, the tracking error is $x[k]$ as *reference* signal is zero.

[2]In this work, we neglect the effect of transmission time of messages and consider $\delta = 0$ in *time-triggered* communication. The *zero-delay* communication is as good as processing within a single PU except for the fact that the messages need finite time to be transmitted from one PU to another through the communication bus. Usually, the sampling times of mechanical or chemical systems are in the order of milliseconds (ms) whereas the transmission time is in the order of microsecond (μs) or nanosecond (ns).

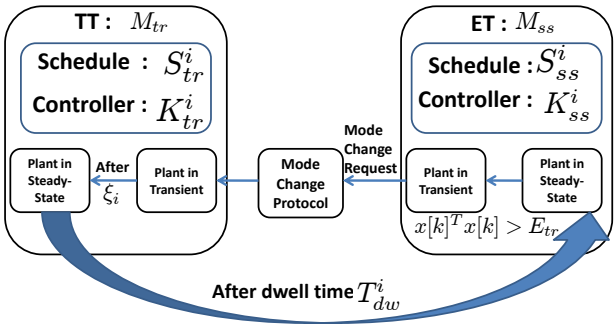

Fig. 3. Our Scheme.

might experience communication delay τ when the higher priority tasks access the *event-triggered* segment. In this work, we assume that the *event-triggered* communication schedules are chosen such that the communication delay τ is less than one sample interval, i.e., $0 < \tau \leq h$ for the control-related messages.

Moreover, we would like to categorize the state of the controller-plant setup into (i) *transient* (ii) *steady-state*. In the transient phase, relatively large variations in the states can be noticed. In this work, we quantify such phases by $x[k]^T x[k]$ which is an indicator of the system's energy level. $x[k]^T x[k] > E_{tr}$ indicates a transient state where E_{tr} indicates the *threshold*. A transient state can also be a result of *disturbances*. The state of the closed-loop dynamics with $x[k]^T x[k]$ being bounded by E_{tr} for reasonably long time period is referred as steady-state. In the following Sections, we provide a formal description of our scheme based on switching between control inputs and communication schedules.

Related Work: The stable switching schemes among multiple controllers is quite heavily studied research area [7], [8], [9]. There are several well-known approaches for synthesizing stable switching schemes (a brief survey is available in [7]). In these works, the communication bus is essentially modeled by one delay or a set of delays and controllers are synthesized in a generalized fashion without any regulation over the communication bus. Because of such restrictions, the controller synthesis often becomes overly conservative and fails to identify feasible solution for a wide class of systems. In contrast to these approaches, our scheme proposes a switching in both communication schedule and the controller. Such scheme essentially provides better control over the communication delay which helps to exploit the potential of the communication protocol in controller synthesis.

III. Overview of our Scheme

We consider an architecture consisting of n distributed control applications denoted by C_i for $i \in \{1...n\}$ (along with other applications). Each controller has two *modes* $M \in \{M_{ss}, M_{tr}\}$. M_{tr} and M_{ss} are the modes where we provide *TT* and *ET* communications for the control-related messages respectively.

Controllers: Each controller has a set of controller gains $K_i = \{K^i_{ss}, K^i_{tr}\}$ which are applied at the respective modes. The gains K^i_{tr} are designed to asymptotically stabilize C_i

with the control law $u[k] = K^i_{tr}x[k]$ and *TT* communication schedule. Using Ackermann's formula [10], one can *place* the *poles* of the closed-loop dynamics so as to meet the *stability* and *performance* requirements.

The gains K^i_{ss} are designed such that the plant is asymptotically stabilized with the control law $u[k] = K^i_{ss}x[k-1]$. Therefore, K^i_{ss} is designed to stabilize C_i with *one* sample delay in every control-related message (with *ET* communication). Due to one sample delay in the feedback loop in case of *ET* communication, the order of the *augmented* plant is one order higher than the original plant. Unlike the case of *TT* communication of control-related messages, there are some restrictions on the *pole-placement* of the augmented system (we do not provide the details of these design issues as they are not the focus of this work) and we cannot use Ackermann's formula. It is assumed that K^i_{ss} is designed considering those restrictions such that $u[k] = K^i_{ss}x[k-1]$ is a stabilizing control law for C_i.

Note that the control gains $K_i = \{K^i_{ss}, K^i_{tr}\}$ are such that $u[k] = K^i_{tr}x[k]$ and $u[k] = K^i_{ss}x[k-1]$ guarantee stability of control applications. However, there is no guarantee for stability with control laws $u[k] = K^i_{ss}x[k]$ and $u[k] = K^i_{tr}x[k-1]$.

Communication schedules: We have a *ET* communication schedule where each control application C_i has a *priority* S^i_{ss}. Therefore, the event-triggered segment of the communication cycle is shared by n control applications with n *priorities*. We have m shared *TT* communication slots such that $m < n$. The n control applications are divided into m *subgroups* such that $\sum_i n_i = n$ where n_i is the number of elements in the i^{th} subgroup. Each subgroup is assigned one *TT* communication slot S^i_{tr}.

Our Scheme: The basic idea of our scheme is shown in Fig. 3. A control application resides at M_{ss} when it is in steady-state. In M_{ss} the communication schedules and controller gains are S^i_{ss} and K^i_{ss} respectively. Due to *ET* communication, every control related message experiences one sample delay at all the times and $u[k] = K^i_{ss}x[k-1]$ stabilizes the application. The state of the application changes to transient when $x[k]$ becomes such that $x[k]^T x[k] > E_{tr}$ due to *disturbances* or certain initial conditions. The control application in transient state issues a mode change request $M_{ss} \rightarrow M_{tr}$. When the *mode change protocol* (discussed in the next paragraph) allows the application the mode change, the application goes to M_{tr}. In M_{tr} the communication schedules and controller gains are S^i_{tr} and K^i_{tr} respectively. With *TT* communication in M_{tr}, every control related message experiences zero communication delay at all the times and $u[k] = K^i_{tr}x[k]$ stabilizes the application with improved control performances. The application goes to steady-state after ξ_i time where ξ_i is the settling time (in terms of sample count) with control law $u[k] = K^i_{tr}x[k]$. However, the application is not allowed to change mode to $M_{tr} \rightarrow M_{ss}$ before T^i_{dw} time is elapsed. T^i_{dw} is the *dwell* time for the control application C_i and $T^i_{dw} > \xi_i$. After T^i_{dw} the application switches back to M_{ss} and remains in that mode until a new disturbance arrives.

We apply the scheme to every control application in a subgroup. In this case, multiple controller might request for mode changes from $M_{ss} \rightarrow M_{tr}$ at the same time. However, only one controller (in each subgroup) can be allowed to reside at M_{tr}. Hence, we adapt a *mode change protocol* for a systematic arbitration among the control applications within the subgroup.

Mode change protocol: Each control application in a subgroup is assigned a *priority*[3]. The *mode change protocol* allows the application with highest priority (among the applications requesting for a mode change) to change its mode. Once a higher priority application is in M_{tr}, it must reside at that mode for T^i_{dw} samples. Therefore, the lower priority applications must wait for T^i_{dw} samples. In order to minimize the waiting time of the lower priority applications, the applications are assigned priority with increasing order of their dwell time T^i_{dw}, i.e., the application with lowest T^i_{dw} is of highest priority and so on.

In the following we answer some of the relevant questions regarding the scheme:

- **What do we achieve?** In case of $m = n$, we have *TT* communication for all the control applications at all times. On the other hand, $m = 0$ indicates *ET* communication for all the applications at all times. We are interested in a trade-off where $0 < m < n$, i.e., we consume less *TT* communication slots than the case when $m = n$ and achieve better control performances compared to the case when $m = 0$.
- **What will happen in case of $m << n$?** In this case we have comparatively higher number of control applications and we want to spend less *TT* communication. Naturally, there will be too many *pending* mode change requests $M_{ss} \rightarrow M_{tr}$. This will essentially results in control performance similar as that with *ET* communication (but system will still be stable). Therefore, one can make a trade-off between how many *TT* communication slots should be used and improvement in control performance.
- **How do we choose dwell time T^i_{dw}?** The choice of T^i_{dw} depends on the application and also on the expected range of initial conditions [8]. The larger the dwell time slower the switching is and more guaranteed the switching stability is. Although a large dwell time provides stronger stability guarantee, larger dwell time might results in higher waiting time (inferior performance) for the lower priority applications in case simultaneous mode change request. Therefore, the dwell time should be chosen considering the above mentioned trade-off. In this work, we assume that the proper values of T^i_{dw} are given such that $T^i_{dw} > \xi_i$.
- **What is the maximum waiting time for a mode change request to be granted?** Multiple mode change request can occur in case *disturbances* (or transient) occur

[3]Note that the priorities assigned by mode change protocol are not related to the priorities associated with the ET communication.

in more than one control applications simultaneously. Effectiveness of the scheme lies on the assumption that disturbances do not occur too frequently, e.g., the inter-arrival time of two consecutive disturbances in any application is more than the sum of all T^i_{dw}. This is a realistic assumption because every controller is usually designed to meet performance requirement based on some specified bound on disturbances. With such occurrence of disturbances, the maximum waiting time of a controller is the sum of T^i_{dw} of all the higher priority control applications.

A. Comments on switching stability

We provide an intuitive outline of why the resulting switching system is stable. We utilize the idea that energy (or Lyapunov function) decreases monotonically with time if the overall switching system is asymptotically stable. In the proposed switching scheme, there are two subsystems (M_{tr} and M_{ss}). Both the subsystems are asymptotically stable. Let us consider the case without external disturbances. In this case, once the system switches to M_{ss}, it will always be in M_{ss} which is a asymptotically stable subsystem. Hence, switching stability is obvious without external disturbance.

In case, the disturbance occurs in M_{ss} and $x[k]^T x[k] > E_{tr}$, a mode change request is issued. The mode change protocols allows mode change to M_{tr} after finite number of samples (when the higher priority control applications reside at the M_{tr}). M_{tr} being asymptotically stable, energy decreases monotonically in M_{tr} and switches to M_{ss} after T_{dw} samples. If dwell time $T^i_{dw} > \xi^i$ for all possible initial states of the C_i, then $x[k]^T x[k]$ will be close to zero and $x[k]^T x[k] \leq E_{th}$ at the end of T^i_{dw}. Hence, if the disturbances do not occur too frequently (as mentioned in the previous Section), the systems then stays in M_{ss} and achieves asymptotic stability.

In case a disturbance occurs in M_{tr} and control application is unable to reach steady-state after T^i_{dw} samples, it issues a mode change request immediately after the switch $M_{tr} \rightarrow M_{ss}$. The subsequent switching sequence will be same as described above.

B. Illustrative Example

We illustrate the significance of the scheme by considering two distributed second order discrete-time systems (C_1 and C_2) given by (2) and (3). Let us now consider the three cases, each with different number of available *TT* communication slots.

$$A_1 = \begin{bmatrix} 0.4 & 1.0 \\ -1.56 & -0.9 \end{bmatrix}, B_1 = \begin{bmatrix} 0.3 \\ 0.1 \end{bmatrix}, \quad (2)$$

$$A_2 = \begin{bmatrix} 1.2 & 0.2 \\ -1.8 & -2.1 \end{bmatrix}, B_2 = \begin{bmatrix} 0.2 \\ 0.3 \end{bmatrix}. \quad (3)$$

Case I (m=n=2): We consider assigning *TT* communication slots to a message in each control application. Using Ackermann's formula, we compute $K^1_{tr} = \begin{bmatrix} 7.4394 & 2.6819 \end{bmatrix}$ and $K^2_{tr} = \begin{bmatrix} 0.0417 & 2.9722 \end{bmatrix}$ by placing the poles at -0.2 and $+0.2$ (for both applications). Applying the control law $u[k] = K^i_{tr} x[k]$, we obtain $x_1[k]$ as shown in Fig. 4 (we use identical initial conditions for both the control applications). The settling times (ξ_i) for both the controllers is around 0.14 second. The integral error cost and the input cost of C_1 are $\sum_k x[k]^T x[k] = 8.6 \times 10^3$ and $\sum_k u[k]^2 = 4.7276 \times 10^4$. For C_2, $\sum_k x[k]^T x[k] = 1.8136 \times 10^3$ and $\sum_k u[k]^2 = 1.2857 \times 10^4$. Here, for two control applications, we consume two *TT* communication *slots*.

Case II (m=0): We assign the same control messages in the event-triggered segment for the *ET* communication. Let $K^1_{ss} = \begin{bmatrix} 3.4674 & 2.7978 \end{bmatrix}$ and $K^2_{ss} = \begin{bmatrix} -6.1031 & -4.0312 \end{bmatrix}$. We apply the control law $u[k] = K^i_{ss} x[k-1]$ and obtain $x_1[k]$ as shown in Fig. 5. The ξ_i of C_1 and C_2 are 2.42 seconds and 0.28 second respectively. The integral error cost and the input cost of C_1 are $\sum_k x[k]^T x[k] = 6.9648 \times 10^6$ and $\sum_k u[k]^2 = 9.1703 \times 10^6$. For C_2, $\sum_k x[k]^T x[k] = 5.4479 \times 10^4$ and $\sum_k u[k]^2 = 3.0933 \times 10^5$. We do not consume any *TT* communication *slot*. The quality of the control performance is noticeably inferior compared to the case where we used *TT* communication *slots*, i.e., (m=2).

Case III (m=1): We choose $E_{tr} = 0.02$ for both the applications and $T^1_{dw} = 15$ and $T^2_{dw} = 10$ samples[4]. Therefore, C_2 has higher priority than C_1. We demonstrate the performance of our scheme at the worst case situation, i.e., when both the applications request for mode change at the same time and C_1 has to wait for T^2_{dw} samples. All the gains values are already mentioned in the previous paragraphs. We show the performance of the controller using our scheme in Fig. 6. The ξ_1 of C_1 is 0.36 second and $\sum_k x[k]^T x[k] = 1.3651 \times 10^6$ and $\sum_k u[k]^2 = 2.3607 \times 10^6$. The ξ_2 of C_2 is 0.14 second and $\sum_k x[k]^T x[k] = 1.8136 \times 10^3$ and $\sum_k u[k]^2 = 1.2857 \times 10^4$. We can notice that ξ_i of both the applications is much lesser and also the other integral costs are improved compared to those with *ET* communication schedule (Fig. 6). At the same time, we consume only one *TT* communication slot for running the applications in M_{tr}. Therefore, we could achieve improved control performance utilizing lesser *TT* communication. The effectiveness of the above scheme is more prominent when number of control applications are highly unstable (e.g., C_1). The difference between control performance with *TT* and *ET* communication is more visible in the case of higher instability margin of the original plant.

IV. FlexRay Control Co-Simulation Framework

Now we illustrate the above scheme in the context of a setup where multiple control applications are communicating over a FlexRay bus.

A. FlexRay Protocol

The FlexRay communication protocol [6] is organized as a periodic sequence of communication cycles as shown in Fig. 2. Each cycle is of fixed length and is indexed by a *cycle counter* that is incremented from 0 to 63 after which the counter is reset to 0. Every such cycle consists of a

[4]Note that because there was no switching involved in the previous cases, these parameters were not relevant.

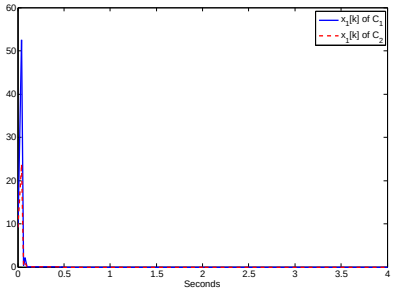

Fig. 4. The plots of $x_1[k]$ with *TT* communication (Case I).

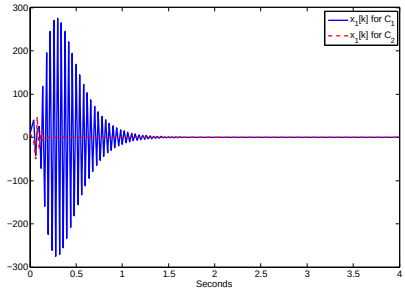

Fig. 5. The plots of $x_1[k]$ with *ET* communication (Case II).

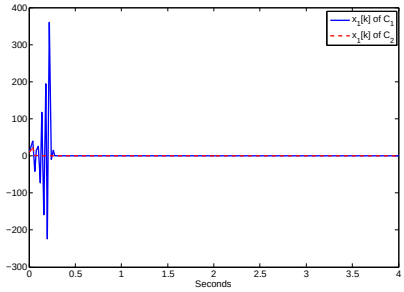

Fig. 6. The plots of $x_1[k]$ with our switching scheme (Case III).

static (ST) segment that is partitioned into time *slots*. We use ST segment schedules for *TT* communication. Further, there is a dynamic (DYN) segment that is partitioned into *minislots* indexed by a *minislot counter*. The DYN segment schedules are used for *ET* communication. A *slot counter* counts the communication slots in ST and DYN segment in order to indicate time windows for admissible message transmissions. Each FlexRay message m_i is assigned a schedule (S_i, B_i, R_i) for uniquely specified transmission points where S_i denotes the assigned slot number, R_i specifies the number of cycles that must elapse between two allowable transmissions and B_i denotes the cycle offset within 64 cycles.

B. Simulator

Our simulator is made up of a SystemC based *FlexRay event simulator* in order to simulate communication delays and the *discrete-time system model* in order to simulate the discrete-time control system. The FlexRay simulator consists of several submodules:

- *FlexRay clock* provides the global time base
- *Event generator* generates input event streams based on the system description
- *FlexRay communication model* implements the FlexRay protocol and computes the message delays

The message delays are used by the *discrete-time system model* to compute the sensor-to-actuator delays in order to simulate the stability of the system.

Discrete-Time Control 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. 1 (a). A task in mode M is represented by tuple $T_i = \{P_i^M, O_i^M, M, \Theta_i^M\}$ where P_i^M, O_i^M and Θ_i^M are the period, offset and the schedule of the task in mode M. The task offset O_i^M in *TT* (static segment) communication schedule is chosen such that the tasks are triggered at the beginning of its ST slot. For the *ET* (dynamic segment) communication schedules, O_i^M is chosen such that $\lceil \frac{\tau}{h} \rceil = 1$. For example, the sensor task T_s is triggered with offset O_s, the controller task T_c is processed on the same processor with offset $O_c > O_s + r_s$ where r_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 FlexRay bus with period $P = h$ and time delay τ. The actuator task T_a is triggered with an offset $O_a = O_s$ and performs the input to the plant.

TABLE I
COMPARISON OF VARIOUS CONTROL/SCHEDULER CO-DESIGN CHOICES

C_i	schedule	$\sum_k x[k]^T x[k]$	$\sum_k u[k]^2$	ξ_i (seconds)
C_1	*TT*	8.6×10^3	4.7276×10^4	0.14
	ET	6.9648×10^6	9.1703×10^6	2.42
	Case A	5.5494×10^6	7.5994×10^6	0.56
	Case B	1.3651×10^6	2.3607×10^6	0.36
C_2	*TT*	1.8136×10^3	1.2857×10^4	0.14
	ET	5.4479×10^4	3.0933×10^5	0.28
	Case A	5.4479×10^4	3.0933×10^5	0.28
	Case B	1.8136×10^3	1.2857×10^4	0.14
C_3	*TT*	9.0173×10^4	1.4646×10^5	0.10
	ET	4.2168×10^5	1.1068×10^5	0.22
	Case A	9.0173×10^4	1.4646×10^5	0.10
	Case B	9.0173×10^4	1.4646×10^5	0.10
C_4	*TT*	3.3182×10^5	6.6168×10^5	0.12
	ET	1.0428×10^{10}	2.5380×10^{10}	1.02
	Case A	7.4845×10^9	1.8053×10^{10}	0.31
	Case B	1.3454×10^9	$1.3.2326 \times 10^9$	0.19

V. RESULTS

System description: The FlexRay configuration parameters have been specified using a commercial off-the-shelf design tool called EB Designer Pro [11]. The cycle length is set to $5ms$ with ST segment of length $2ms$ and 10 static slots. The rest of the cycle has been distributed to the DYN segment. Several applications being mapped on the DYN segment are exchanging 14 messages on the bus. We consider four distributed control applications with four control-related messages being transmitted via FlexRay. Each control application has three tasks, i.e., T_s, T_a and T_c with an architecture shown in Fig. 1 (a) for all the four control applications. The two of the control applications are as per (2) and (3). The other control applications are as per (4) and (5). The sampling interval h of all the controllers is 20 ms. The controller gains for C_1 and C_2 are already mentioned in Section III-B. For C_3 and C_4, $K_{tr}^3 = \begin{bmatrix} 3.2492 & 1.3777 & 1.0214 \end{bmatrix}$, $K_{tr}^4 = \begin{bmatrix} 3.4318 & -1.6184 & 4.8646 \end{bmatrix}$, $K_{ss}^3 = \begin{bmatrix} 1.4505 & 0.9700 & 1.0840 \end{bmatrix}$ and $K_{ss}^4 = \begin{bmatrix} -3.3690 & -2.5287 & -3.6729 \end{bmatrix}$. Four control applications are assigned four DYN segment slots such that the resulting sensor-to-actuator delay of the control related messages is $0 < \tau \leq h$. The *ET* communication is realized by four DYN segment slots (recall Section IV-A) $S_{ss}^1 : (30, 0, 2)$, $S_{ss}^2 : (33, 0, 2)$, $S_{ss}^3 : (35, 0, 2)$ and $S_{ss}^4 : (37, 0, 2)$. We simulate the above distributed architecture in the

co-simulation framework described in Section IV.

$$A_3 = \begin{bmatrix} 0.4 & 0.6 & 0.7 \\ -1.56 & -0.9 & -0.6 \\ -3.6 & 1.2 & 1.1 \end{bmatrix}, B_3 = \begin{bmatrix} 0.1 \\ 0.7 \\ 0.5 \end{bmatrix}, \quad (4)$$

$$A_4 = \begin{bmatrix} 0.7 & 0.9 & 0.3 \\ 1.6 & -1.0 & 1.1 \\ -1.6 & 1.2 & -2.8 \end{bmatrix}, B_4 = \begin{bmatrix} 0.3 \\ 0.1 \\ 0.5 \end{bmatrix}. \quad (5)$$

Control Performance: We choose the dwell time for the four control applications as $T^1_{dw} = 10$, $T^2_{dw} = 7$, $T^3_{dw} = 6$ and $T^4_{dw} = 8$ (in samples). Therefore, C_3 is the highest priority application and then comes C_2, C_4 and C_1 respectively. We choose $E_{tr} = 0.02$ for all the applications. Comparison of various control performance parameters for different choices of communication schedules is shown in Table I.

Case A (m=1): In this case, we have one subgroup consisting of all the applications. We choose the second ST slot for the *TT* communication, i.e., S_{tr}. Table I shows the performance when all the four control applications have issued mode change request together (the worst case performance for our scheme).

Case B (m=2): We divide the control applications into two subgroups: $\{C_1, C_2\}$ and $\{C_3, C_4\}$ and assign second and third ST segment slots for *TT* communication. The control performances are shown in Table I.

Discussions: We can see from the Table I that using our scheme we could achieve a control performance which is better than performance with only ET communication. At the same time, we consume lesser TT communication *slots* compared to the case when only TT communication is used. Therefore, we could make a good trade-off between control performance and use of TT communication slots. Furthermore, improvement in control performance is more when we consume more TT communication *slots* (e.g., Case B gives better performance than Case A). Therefore, the consumption of TT communication *slots* can be adjusted depending on the designer's requirement on the control performance. Finally, the performance improvements are more prominent in the control applications with more instability (e.g., C_1 and C_4) compared to others (i.e., C_2 and C_3).

Effect of disturbances: Let us consider the application C_4 which is initially blocked by higher priority applications for 0.12 second. Suppose, large disturbances occur at sensor signals as $x_3[k] = 1000$ at $t = 0.5$ seconds and $x_1[k] = 1000$ at $t = 1.5$ seconds. We can see the variation of $x_1[k]$ in Fig. 7 (a). In both cases, the application goes back to steady-state within 0.36 seconds. Now, consider the same application C_4 with disturbance of $x_3[k] = 100$ at $t = 2.0$ seconds. At the occurrence of disturbance C_4 issues a mode change request, but mode change protocol does not allow a mode change until 2.14 seconds as M_{tr} is occupied by a higher priority application. The Fig. 7 (b) shows the plot of $x_1[k]$ which comes back to steady-state after 0.36 seconds.

VI. Concluding Remarks

In the case of distributed control applications communicating over a hybrid (time- and event-triggered) protocol like FlexRay, there are two possibilities: (i) all control messages are scheduled on the time-triggered segment, or (ii) all control messages are scheduled on the event-triggered segment. Traditionally, depending on (i) or (ii), the controller is designed accordingly. (i) has the advantage of better temporal behavior but poor bus utilization and flexibility, while (ii) suffers from the lack of temporal guarantees and hence possibly poor control performance. In this paper we have shown how control applications may be re-engineered to operate in two modes, which is in line with the bi-model characteristics of the communication protocol. We have shown that this provides a good trade-off between control performance and the number of utilized time-triggered slots. As a part of future work, we plan to quantify the disturbance model and analyze this problem from a schedulability-theoretic perspective (i.e., for how long might an application have to wait after issuing a mode-change request).

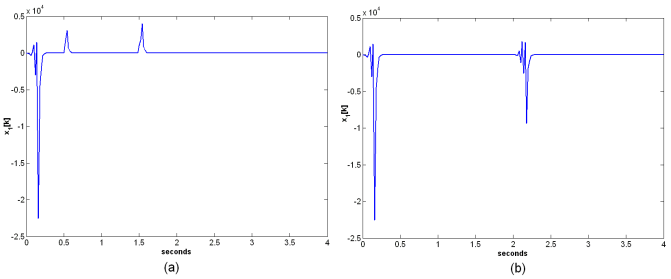

Fig. 7. Effect of disturbance in application C_4: (a) $x_3[k] = 1000$ at $t = 0.5$ seconds and $x_1[k] = 1000$ at $t = 1.5$ seconds (b) $x_3[k] = 100$ at $t = 2.0$ seconds with 0.14 seconds waiting time for the mode change.

VII. Acknowledgement

The first author of the paper is 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*, vol. 42, no. 3, pp. 88 – 89, 2009.

[2] H. Voit, R. Schneider, D. Goswami, A. Annaswamy, and S. Chakraborty, "Optimizing hierarchical schedules for improved control performance," in *International Symposium on Industrial Embedded Systems (SIES)*, 2010.

[3] 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*, vol. 14, no. 4, pp. 776 – 787, 2006.

[4] S. Samii, A. Cervin, P. Eles, and Z. Peng, "Integrated scheduling and synthesis of control applications on distributed embedded systems," in *Design Automation and Test in Europe (DATE)*, 2009.

[5] "ISO/CD11898-4, Road Vehicles Controller Area Network (CAN) Part 4: Time-Triggered Communication, International Standards Organization, Geneva," 2000.

[6] "The FlexRay Communications System Specifications, Ver. 2.1," www.flexray.com.

[7] W. Jiang, E. Fridman, A. Kruszewski, and J. Richard, "Switching controller for stabilization of linear systems with switched time-varying delays," in *IEEE Conf. on Decision and Control (CDC)*, 2009.

[8] G. Chesi, P. Colaneri, J. C. Geromel, R. Middleton, and R. Shorten, "Computing upper-bounds of the minimum dwell time of linear switched systems via homogeneous polynomial lyapunov functions," in *American Control Conference (ACC)*, 2009.

[9] S. Hirche, C.-C. Chen, and M. Buss, "Performance oriented control over networks switching controllers and switching delay," *Asian Journal of Control*, vol. 10, no. 1, pp. 24 – 33, 2008.

[10] "http://www.esr.ruhr-uni-bochum.de/rt1/syscontrol/node105.html."

[11] "Elektrobit tresos, www.elektrobit.com."