Challenges in Automotive Cyber-physical Systems Design

Dip Goswami[1] Reinhard Schneider[1] Alejandro Masrur[1]
Martin Lukasiewycz[2] Samarjit Chakraborty[1] Harald Voit[3] Anuradha Annaswamy[4]
[1]Institute for Real-Time Computer Systems, TU Munich, Germany
[2]TUM-CREATE Center for Electromobility, Singapore
[3]Institute for Automatic Control, TU Munich, Germany
[4]Massachusetts Institute of Technology, USA
E-mail: dip.goswami@tum.de, reinhard.schneider@rcs.ei.tum.de, masrur@rcs.ei.tum.de
martin.lukasiewycz@tum-create.edu.sg, samarjit@tum.de
voit@lsr.ei.tum.de, aanna@mit.edu

Abstract

Systems with tightly interacting computational (cyber) units and physical systems are generally referred to as cyber-physical systems. They involve an interplay between embedded systems, control theory, real-time systems and software engineering. A very good example of cyber-physical systems design arises in the context of automotive architectures and software. Modern high-end cars have 50-100 processors or electronic control units (ECUs) that communicate over a network of buses such as CAN and FlexRay. In such complex settings, traditional control-theoretic approaches – where control engineers are only concerned with high-level plant and controller models – start breaking down. This is because implementation-level realities such as message delay, jitter, and task execution times are not adequately considered when designing the controller. Hence, it is becoming necessary to adopt a more holistic, cyber-physical systems design approach where the semantic gap between high-level control models and their actual implementations on multiprocessor automotive platforms is quantified and consciously closed. In this paper we give several examples on how this may be done and the current research challenges in this area that are being faced by the academia and the industry.

1 Introduction

Today automotive in-vehicle networks consist of up to 100 ECUs, sensors, and actuators that run a large number of control loops which are closed over a network of shared resources (Fig. 1(a)). Typical examples include the engine control unit, the body control subsystem, the chassis control, and safety functions like adaptive cruise control. In such setups, various functionalities are realized by a number of distributed tasks which communicate by exchanging messages over the shared network. The performance of these applications depends on the operating system (OS) running on the ECU, where common automotive OSs can either be preemptive (e.g., OSEK, OSEKTime) or non-preemptive (e.g., eCos) and the response time of messages transmitted over networks which depends on the arbitration policy implemented. In general, the scheduling policies also can either be time-triggered (e.g., the static segment of FlexRay) or event-triggered (e.g., the dynamic segment of FlexRay, or CAN). Inherent heterogeneity and diverse nature of the modern automotive functionalities make such systems very complex. The necessity of tightly interacting cyber and physical components requires introduction of new design paradigm [14].

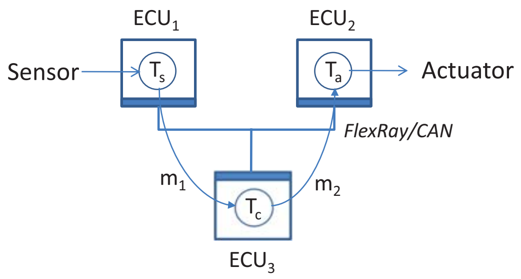

Figure 1: Distributed automotive control application.

Typically, a feedback control function aims to achieve the desired behavior of a dynamic system by applying appropriate inputs to the system. As illustrated in Fig. 2, the traditional automotive design flow of such applications has two distinct and decoupled phases: (i) control algorithm design and (ii) its implementation on a specific embedded systems platform. The design phase decides on the control algorithm based on the dynamic model (in form of a set of differential or difference equations) obtained by the

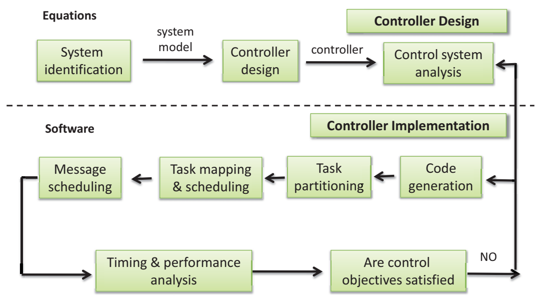

Figure 2: Control Systems Implementation

system identification techniques. Hence, the outcome of this phase is the control algorithm which is designed and analyzed in simulation setups, e.g., Simulink/Stateflow, to meet all application-level specifications/requirements (e.g., stability, settling time, or tracking error). The implementation phase starts with platform-specific code generation for the proposed control algorithm. Due to the distributed nature of the automotive architecture, the control code needs to be partitioned into a number of software tasks for sampling the sensor readings (T_s), computing the control inputs (T_c), and performing actuations (T_a) as shown in Fig. 1. These software tasks need to be mapped and scheduled on different ECUs. Depending on the task mappings, control signals need to be exchanged via a shared, arbitrated network. Hence, data processed by the tasks, e.g., feedback signals, are packetized as messages which need to be scheduled accordingly (Fig. 1(b)). Finally, it is verified whether the high-level control objectives are met and if not, a design revision is deemed necessary which starts again with another control design phase iteration.

1.1 CPS-oriented Design

In the context of the above-mentioned complex distributed settings, the traditional design flow – where controller design and their implementations are performed in a decoupled fashion – results in (i) a *semantic* gap between the control algorithms and their implementations and (ii) in many occasions, an overly conservative design. In the following, we illustrate the main shortcomings of traditional design approaches and the motivations for more holistic cyber-physical systems design approach.

- Traditionally, the control theory community has largely ignored the implementation aspects (both software and hardware architecture) of controllers and has focused on mathematical models, their analysis, and high-level simulation. In this process, a number of simplifying assumptions have been made when designing controllers. They include neglecting the computation times when evaluating control laws and neglecting the control message communication times. As the implementation platforms became more complex and the *semantic* gap between high-level control models and their implementations widens, the control theory community started investigating what is referred to as *networked control systems* (NCS) [2, 13], where the focus has been on distributed controllers communicating over a wireless network. Research with NCS takes into account issues such as delays suffered by control messages, message loss and jitter, and factors these issues into controller design. In general, a majority of the research on NCS assumes a given network and focuses on the controller design related aspects. On the contrary, in the automotive setups under consideration, the designer has sufficient handle over the network parameters. Hence, the network design can significantly be directed by the control related requirements and vice-versa – which is exactly the focus of flurry of recent works on cyber-physical design [1, 3–5, 7–9, 12, 15].

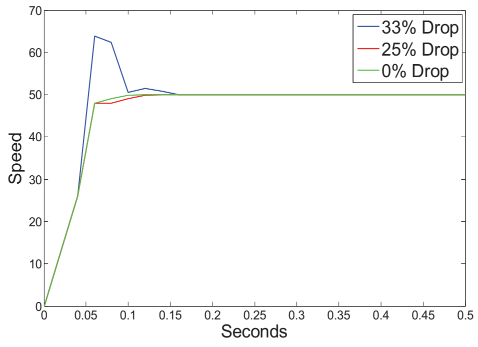

Figure 3: Inherent robustness of a control loop.

- In general, the control applications are different from usual real-time applications in many aspects. E.g., a real-time application is often characterized by its deadline whereas the control applications are mostly judged by their performance. Fig. 3 shows the effect of feedback (packet) drop or deadline miss in a DC motor speed control application in electric cars where the goal is to achieve a speed reference of 50 units. The result shows the system behavior in the presence of certain deadline miss. The system behavior does not deteriorate much with up to 25% of the feedback signals being dropped/miss their deadlines. Although the deadline requirements in control are soft in nature, they cannot be categorized as *soft* real-time applications (which are considered to be less critical) because of their stingent performance requirements. Clearly, the control applications are special in many ways. Ignoring such special properties in the design phase, the embedded implementation of control algorithms often leads to conservativeness both in terms of resource utilization and control performance. The cyber-phyiscal design paradigm [3,4,8,12] aims to improve the design by exploiting such specialities of control applications.

In this paper, we illustrate various aspects of the above-mentioned cyber-physical design approaches and associated challenges by using several design examples from the automotive domain. In Section 2, we describe a typical automotive problem setting and design challenges. The cyber-physical design approach is illustrated and demonstrated with a design example in Section 3. In the following two sections (Section 4 and 5), we demonstrate two design examples on how the special properties of control applications can be exploited by cyber-physical design approaches to improve the overall design.

2 Problem Setup and Design Challenges

Design of cyber-physical automotive applications involves challenges both from control theoretic and embedded systems sides. In this section, we first describe the overview of various design objectives from control side. Next, we describe the challenges involved in the design process of embedded platforms. In the subsequent sections, we show by a number of design examples how these challenges from both domains can be jointly addressed by holistic CPS design approaches.

2.1 Control Design Challenges

In general, dynamic systems (or plants) are modeled by a set of differential equations,

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + B_c u(t) \\ y(t) &= C_c x(t) \end{aligned} \tag{1}$$

where $x(t)$ is *state vector* and $u(t)$ is the *control input* to the system. A_c is *system matrix* and B_c is input vector. In a state-feedback control algorithm, control input $u(t)$ is computed utilizing the system states $x(t)$ (feedback signal). The computed $u(t)$ is then applied to the plant which is supposed to achieve the desired behavior. The overall loop performs three operations:

- *Measure*: The output of the dynamical system $x(t)$ is measured by one or more measuring devices or *sensors* and thus, the measured signals act as feedback signals.
- *Compute*: The feedback signals are compared with desired output and necessary correction in the input signal $u(t)$ is computed by the control algorithm.
- *Actuate*: The correction is incorporated by changing the input signal or actuation.

Typically, a high level goal of such feedback control aims to achieve desired behavior (i) fast, and (ii) accurately, while stability of the loop must be maintained. There are variety of stability notions, e.g., $x(t)$ and $u(t)$ are *bounded* (BIBO stability) and $y(t) \to r$ as $t \to \infty$, r is the *reference* (asymptotic stability). The speed and accuracy of the feedback control system quantify the performance of the controller. There is a wide variety of notions to quantify the performance of a control function. The performance notions are broadly classified into two categories: *steady state* and *transient phase* performance. An example of a transient performance index is the *settling time*, i.e., how fast the system responds to some external *disturbances*. On the other hand, a commonly used performance index at steady state is the quadratic cost function which captures a combination of tracking accuracy and input energy,

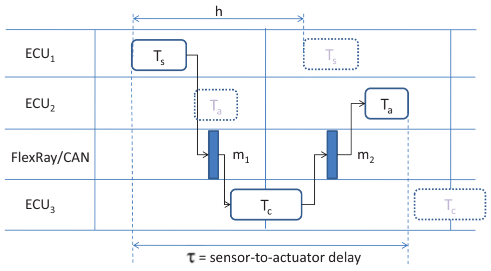

Figure 4: Timing diagram of distributed CPS.

$$J = \int_0^\infty [u(t)^2 + y(t)^T y(t)]dt, \tag{2}$$

where $u(t)$ and $y(t)$ are the system's input and output respectively.

Performing these operations continuously in any implementation platform requires infinite computation power. Hence, in a digital implementation platform of such feedback loops, these operations are performed only at discrete-time intervals (sampling instants). When the time interval between two consecutive sampling instants is constant, the continuous-time system (1) can be transformed into a discrete-time system of the form,

$$\begin{aligned} x[k+1] &= Ax[k] + Bu[k] \\ y[k] &= Cx[k] \end{aligned} \tag{3}$$

where sampling instants are $t = h_k$ ($k = \{1, 2, 3 \cdots\}$) and $h_{k+1} - h_k = h$ (sampling period). $x[k]$ and $u[k]$ are the values of $x(t)$ and $u(t)$ at $t = h_k$. The performance depends on (i) the sampling period h, and (ii) the sensor-to-actuator delay τ as illustrated in Fig. 4.

From control side, the design challenges rely on the facts that (i) shorter period improves the control performance, and (ii) a shorter sampling period requires higher computational and communication resource usage [1,8]. The overall goal is to choose sampling periods and associated control related parameters of the control tasks such that (i) the control functionalities are robust and provide desired performance, and (ii) all the control and other tasks and messages are schedulable, given the resource constraints.

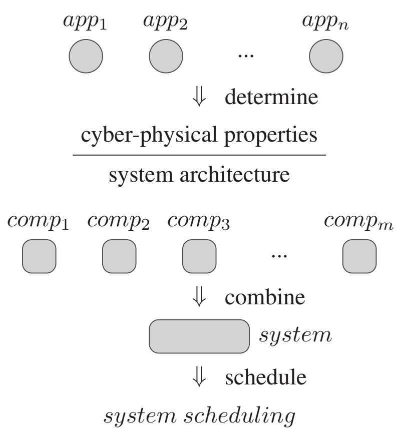

Figure 5: Illustration of a modular scheduling approach that is aware of the cyber-physical system properties.

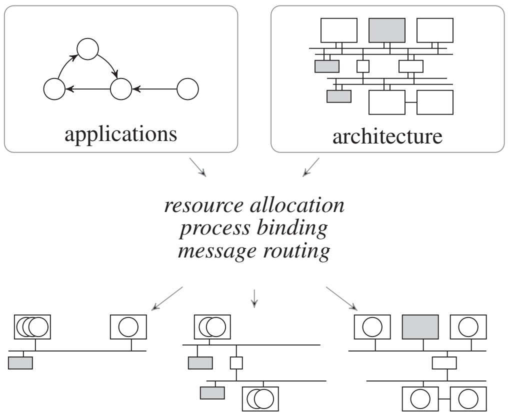

Figure 6: Illustration of a y-chart approach for the architectural design space exploration where a set of applications is mapped to an architecture template. The design space exploration might result in a multitude of different applications.

2.2 Platform Design Challenges

The introduction of the FlexRay bus in the automotive domain allows a fully synchronous scheduling of ECUs and bus systems. In contrast to the predominant CAN bus, a synchronous time-triggered scheduling reduces end-to-end latencies significantly and at the same time avoids jitter if it is well configured. A configuration of a time-triggered system requires the concurrent determination of schedules for all ECUs and buses which is a non-trivial task. For cyber-physical systems in the automotive domain, the scheduling has to be additionally aware of the underlying applications and its constraints. Here, an appropriate abstraction of the plant and its characteristics becomes necessary to support a control-aware scheduling as illustrated in Fig. 5. The properties from a cyber-physcial system have to be extracted and respected within a modular scheduling framework that determines a system model from the component models and finally determines a feasible schedule for the entire system.

The scheduling of cyber-physical systems requires an appropriate application model. In the automotive domain, usually all applications are periodic or defined by a minimal inter-arrival time. The goal of the scheduling is to define the start times for each task and assignments of messages to slots. The constraints of the ECUs and buses have to be fulfilled while the end-to-end latencies of all applications have to be respected. Depending on the underlying cyber-physical system, the latency of an application has also an influence on the control quality which should be considered.

The scheduling method has to take all the architectural constraints into account such as the bus systems and the operating systems. For the ECUs, standard operating systems are either based on non-preemptive scheduling or the preemptive OSEKTime scheduling that is using a last-in-first-out (LIFO) scheduler with a dispatch table for each task.

Development of scheduling methods for the entire systems is a challenging task and are often based on Integer Linear Program (ILP) solvers. In particular, achieving a high efficiency and modularity is an important design goal of such scheduling frameworks. Although current frameworks are able to handle more than 100 tasks, their scalability remains a major concern. Different combinations of heuristics and mathematical solvers are used to tackle this problem.

Design Space Exploration: With a growing flexibility, the optimization of automotive networks is becoming a major challenge. Here, the following components have to be determined: The selection of hardware resources and their configuration (CPU, RAM, etc), the selection of buses and the wiring, the distribution of tasks and messages that allow a feasible scheduling. Common design space exploration frameworks are based on the approach as illustrated in Figure 6. A set of applications is mapped to an architecture template. The design space exploration tasks are usually separated in (i) allocation of resources, (ii) mapping of tasks, and (iii) routing of messages. In a final step a feasible schedule has to be determined for each implementation.

State-of-the-art exploration frameworks are based on ILPs or Evolutionary Algorithms or the combination of both. To enable the usage of design space exploration methods for cyber-physical systems in the automotive domain, the specific constraints from the control theory and the physical systems have to be considered. The combination of existing architectural design space exploration ap-

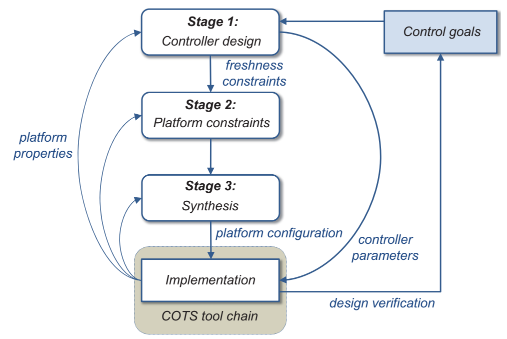

Figure 7: Constraint-driven design flow.

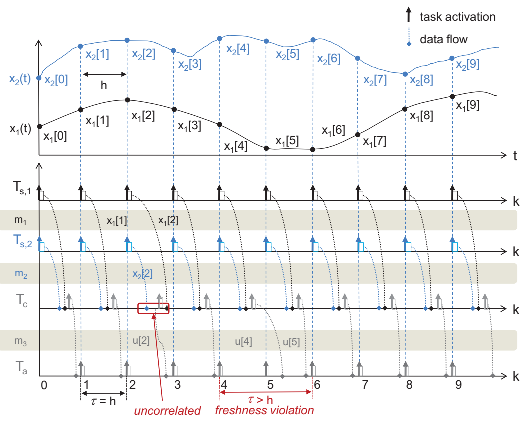

Figure 8: Correlation constraint.

proaches and scheduling approaches for synchronous systems is a challenging task that requires a holistic modeling of the entire system. Moreover, to improve the usability of design space exploration frameworks, the interaction between the designer and the framework become necessary. This includes the adjustment of priorities of the design objectives and the semi-manual determination of implementations.

3 Constraint-driven Algorithm/Architecture Co-design

In this section, we illustrate the cyber-physical design flow in the context of a distributed automotive setting described in Section 2. As discussed in the motivational section, the traditional design flow often ignores the implementation-level effects such as jitter and delays experienced by the feedback signals while being transmitted and processed by a network of shared resources. In this design example we demonstrate how a cyber-physical design approach can address this *semantic* gap and essentially come up with a better design by joint design of the controller and the embedded platform.

The idea is to come up with a set of constraints that captures platform-specific properties (e.g., the constraints coming from the OS running at the ECUs and scheduling policies running on the bus) and at the same time, translates controller-specific requirements into platform-specific constraints. The (optimal) design solution can either be obtained by casting it as a constraint solve problem (CSP) or an ILP problem. The overall design flow can broadly be broken into three stages (Fig. 7):

- **Stage 1 (Controller design):** The *controller design* essentially boils down to the problem of finding suitable controller gains that meet the control goals (i.e., stability and performance). For this, we (i) explore the available sampling periods which satisfy the control goals while checking their feasibility with respect to the underlying implementation platform, and (ii) restrict the design space of the corresponding scheduling parameters that may realize those sampling periods. The control gains are synthesized based on this restricted design space.

- **Stage 2 (Platform Constraints):** Further, the actual control performance depends on the the *age* or *freshness* of the feedback signals which are used to compute the control input. As a result, the freshness constraint imposed by the controller design is translated into a set of platform constraints.
  (1) synchronicity constraint; specifies the phase requirement between local task schedules and global bus schedules. Unsynchronized schedules might lead to large delays.
  (2) schedulability constraint; imposes requirements on task schedules and bus schedules with respect to compliance with the operating system specifications and bus protocols.
  (3) correlation constraint; specifies the time skew among the control input signal and each element of the measured states that is used to compute the input.

- **Stage 3 (Synthesis):** Finally, a feasible platform configuration which satisfies the imposed platform constraints is synthesized using a constraint solver or an ILP-based optimization framework.

To illustrate the above idea, we consider a feedback control application as in (3) with two states and an implementation platform with non-preemptive eCos running on the ECUs and FlexRay as a bus system. The control application is partitioned into four tasks $T_{s,1}, T_{s,2}, T_c, T_a$ and communicates via three messages m_1, m_2, m_3. The sensor tasks $T_{s,1}$, and $T_{s,2}$ read the states of the continuous signals $x_1(t)$, and $x_2(t)$ at every sampling interval h and compute the corresponding outputs $x_1[k]$ and $x_2[k]$ which are packetized as messages m_1 and m_2 and sent over the bus. The

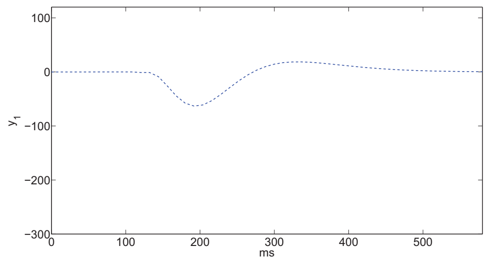

Figure 9: Desired control performance.

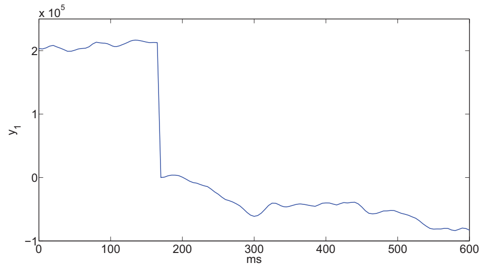

Figure 10: Unstable output due to constraint violation.

controller task T_c utilizes both input signals to compute the control input $u[k]$ which is transmitted over the bus and gets processed by the actuator task T_a which applies the control input to the system. Fig. 8 shows an exemplary timing diagram of a distributed controller implementation. Here, the freshness constraint $\tau = h$ implies that the control input $u[k]$ is utilized for actuation at sampling instants $(k + 1)$. The control objective is to bring back the system to reference (which is zero) after any disturbance. Fig. 9 shows the desired output of the control system which is obtained when the freshness constraint is satisfied. Clearly, the application is asymptotically stable as with $t \to \infty$, $y_1 \to 0$ as mentioned in Section 2.

Fig. 10 shows the case when the freshness constraint on τ is violated at sampling instance $k = 4$ (see Fig. 8) as the control input $u[4]$ gets delayed by more than one sampling interval, and hence arrives at the actuator task T_a not before $k = 5$. Hence, $u[4]$ will finally be applied at $k = 6$ which results in $\tau = 2h$ or $u[4]$ will be overwritten by $u[5]$. Further, at sampling instant $k = 2$ the value of $x_1[2]$ arrives at the controller after the controller task T_c has been triggered at $k = 2$, and hence the control law is computed based on $x_1[1]$ and $x_2[2]$ which leads to an unpredictable control input $u[2]$ and results in an unstable output as depicted in Fig. 10.

As we could see, the performance of applications significantly depends on the design process and the proposed cyber-physical design method could address a number of design- and implementation-level semantic gap.

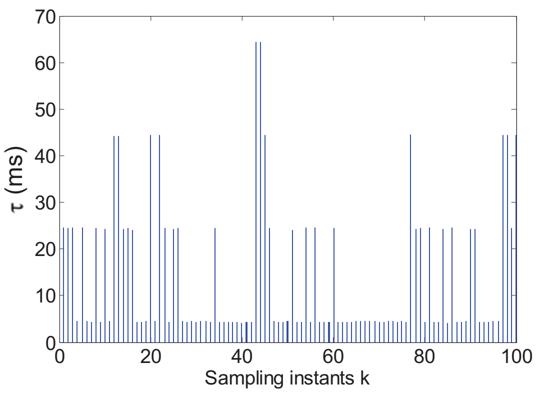

Figure 11: Delay variation over FlexRay dynamic segment.

4 Co-design with Flexible Delay Constraints

In this section, we present another design example where the properties specific to control applications are exploited in the platform design and vice-versa. More specifically, the schedule synthesis can significantly be directed by the inherent robustness of control loops. We consider feedback control loops closed over the dynamic segment of FlexRay. The event-triggered nature of communication in the dynamic segment of FlexRay might cause high variation in feedback delays and occasional large delays. Fig. 11 shows the typical variation of delays experienced by the control related messages in the FlexRay dynamic segment. Given such high variation in delay in the feedback loop, there are two possible design options:
(i) The most obvious design approach to handle such jitter in the feedback loop is to design the control algorithm based on the worst-case delay. Naturally, this leads to overly conservative design as it can be seen from Fig. 11 that the large delays occur very rarely.
(ii) Another possible way to design the feedback controller considering such timing-varying feedback delays is to utilize methods stemming from traditional NCS community [2, 13]. Mostly, these approaches are based on the assumption that the sensor-to-actuator delay *uniformly* varies between the best-case and the worst-case delays which is certainly not the case in most of the real-life architecture. Because of such assumptions, such design often becomes overly conservative and fails to identify feasible designs (or is only applicable to stable plants or plants with marginal instability, as in the example provided in [10]).

Proposed Approach: In contrast to the above approaches, our proposed approach utilizes knowledges from the overall system by (i) quantifying the *robustness* of control loops in terms of permissible *deadline* miss and (ii) extracting

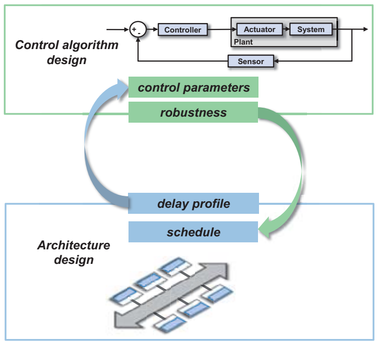

Figure 12: Co-design with Flexible Delay Constraints.

the *delay profile* experienced by the control messages for a specific choice of communication schedule. In this context, a large number of feedback messages suffer a delay no more than some smaller threshold delay say $\tau = \delta$ and control algorithm that is designed based on this threshold delay provides a very good performance. When the control messages experience delay $\tau > \delta$, the control performance deteriorates and can potentially lead to instability. The robustness of the control loops is defined by how frequently such violation of deadline (i.e., $\tau > \delta$) is permissible to the control applications. Here, the robustness is quantified as the ratio between number of samples meeting their deadlines and missing their deadlines as shown in (4) . The other aspect is to abstract the delay profile that the control messages are going to experience for a particular choice of schedule. Utilizing these two aspects from control and its implementation platform it is possible (see Fig. 12) (i) to design the control parameters such that deadline violation constraint μ is satisfied for a given schedule, or (ii) to design the schedule such that the deadline violation constraint μ is satisfied for given control parameters.

$$\mu = \frac{\textit{No. of samples meeting their deadlines}}{\textit{No. of samples missing their deadlines}}. \tag{4}$$

Illustrative Case-study: We illustrate the applicability of the proposed co-design guidelines considering an automotive application, namely cruise control system. A cruise control system receives reference or commanded vehicle's speed from the driver and regulates the speed following the driver's command. Based on the reference speed and the feedback signals, the cruise control system regulates the vehicle's speed by adjusting the engine throttle angle to increase or decrease the engine drive force. The system output is the speed of the vehicle $v(t)$ and $u(t)$ is the engine throttle angle. The objective is to choose $u(t)$ such that $v(t) = r$, a constant desired speed. We have chosen $r = 100$. We have simulated external disturbances by periodically making $v(t) = 80$ (from $v(t) = 100$).

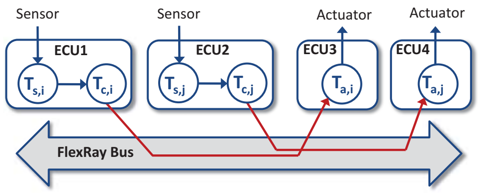

Figure 16: Control application mapping onto a distributed automotive architecture featuring a FlexRay bus

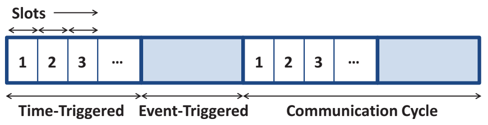

Figure 17: Schematic representation of the FlexRay protocol: time- and event-triggered communication segments

Fig. 13 is the system response with higher priority schedule with $\mu = 40$ which is greater than desired $\mu = 10$ (i.e., meeting design requirement). $v[k]$ reaches $r = 100$ in $1.76sec$ (i.e., settling time=$1.76sec$). Fig. 14 shows the system response with relatively lesser priority schedule resulting in $\mu = 14$ which is still greater than desired $\mu = 10$ (i.e., meeting design requirement). $v[k]$ reaches $r = 100$ in $2sec$. Fig. 15 shows the system response with low priority schedule resulting in $\mu = 9$ which fails to meet desired $\mu = 10$ (i.e., design requirement is not met) and the resulting system is unstable. From the above three cases, it is clear that control application does not need as high priority as $\mu = 40$ and the priorities which result in $\mu \geq 10$ are sufficient to achieve acceptable control performance.

5 Multi-mode CPS Design

As already discussed, available technologies often cannot fulfill and exploit control-theoretic design requirements which can potentially improve the design. Let us consider a set of control applications as given in Eq. (3). Every control application denoted by C_i is partitioned onto a distributed automotive architecture such that $T_{s,i}$ (i.e., the sensing task of C_i) and $T_{c,i}$ are mapped to the same ECU, whereas $T_{a,i}$ is mapped to a separate ECU. As a result, control-related messages need to be transmitted over a shared bus as shown in Fig. 16.

FlexRay is a so-called hybrid bus, i.e., it allows a mix of time- and event-triggered communication as shown schematically in Fig. 17. A communication cycle in FlexRay consists of a static or time-triggered segment and a dynamic or event-triggered segment. When all control messages are mapped onto the static segment of the bus, the time-triggered slots and the ECUs may be perfectly syn-

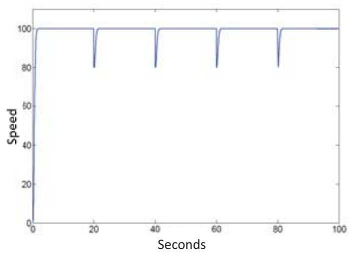

Figure 13: $\mu = 40$ which is greater than desired $\mu = 10$ (i.e., meeting design requirement).

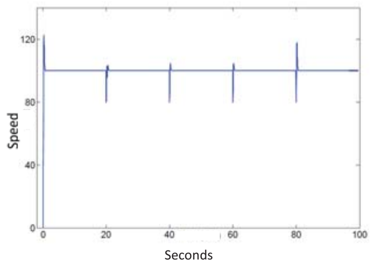

Figure 14: $\mu = 14$ which is greater than desired $\mu = 10$ (i.e., meeting design requirement).

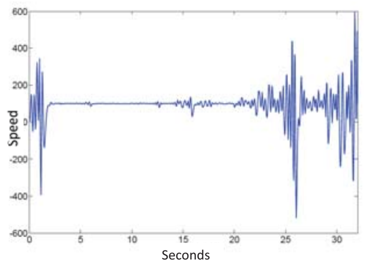

Figure 15: $\mu = 9$ which is lesser than desired $\mu = 10$ (i.e., design requirement is not met).

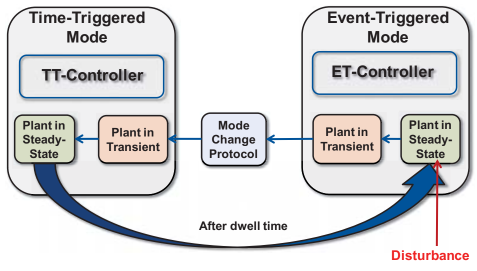

Figure 18: Switching scheme between time- and event-triggered communication for control-related messages

chronized resulting in *zero* (or negligible compared to the sampling period h) communication delays. Clearly, this leads to a good control performance (Fig. 19).

However, it is widely believed that as application complexity and hence communication requirements continue to grow, the bandwidth of the time-triggered segment will not suffice and a *purely* time-triggered implementation might be overly expensive. On the other hand, priority-driven event-triggered implementations suffer from the usual temporal non-determinism which results in poor control performance (Fig. 21) (a lot of oscillations in the response).

We propose an intermediate design possibility where the aim is to achieve control performance close to a purely time-triggered implementation, but use *fewer* time-triggered slots than what would be necessary for fully time-triggered implementation. This is achieved by exploiting the following two observations: (i) the *settling time* of a controller is more susceptible to deterioration during its *transient phase* (i.e., when the system experiences external disturbances), (ii) it is unlikely for independent control applications to experience external disturbances simultaneously. As a result, an event-triggered implementation might suffice when the applications are in their *steady states*. When they move to a transient state because of an external disturbance, we propose to switch the relevant control messages to a time-triggered slot in order to minimize the reaction or settling time. The controllers used in these two modes (time-triggered and event-triggered) are such that they can cope with the different properties of the event-triggered and time-triggered mode such as different delays, packet drop rates, response times, steady-state error, etc. Next to linear controllers like PID controllers, advanced non-linear controllers such as adaptive controllers can be used [11]. This is illustrated in Fig. 18 and the corresponding system response is shown in Fig. 20. With the proposed scheme, control applications do not require a time-triggered slot in an exclusive manner but only at the event of a disturbance. As a result, multiple applications can be assigned to the same slot. Clearly, the proposed switching scheme implies a trade-off between control performance and the cost of communication (i.e., number of time-triggered slots).

5.1 Schedulability Analysis

In the above scheme, a priority-based scheduling becomes necessary to grant access to the shared slot in the event of simultaneous occurrence of disturbances to multiple control applications – the priority given to applications should reflect their urgency. Before switching to time-triggered in the event of a disturbance, an application might have to wait (depending on whether the shared time-triggered slot is currently being used by another application). Such waiting time essentially deteriorates the control performance and the control performance becomes closer to the one with purely event-triggered implementation with higher waiting time.

To meet a given control performance requirement (which cannot be achieved by purely event-triggered implementation), it becomes necessary for the control applications to access the assigned time-triggered slot within a maximum waiting time. This maximum waiting time can be computed from the controller and its performance requirement. If the waiting time for a control application to access time-triggered slot exceeds this maximum time, the control application fails to meet the performance requirements. Hence, this acts as *deadline* for the control application. Clearly, the shared time-triggered slot acts as a processor, the control applications are similar to the task running on it and the occurrence of disturbances which necessitate time-triggered communication behaves as task triggering. Overall, it is a complex schedulabilty problem which is needs to decide on (i) slot assignment to the applications, (ii) arbitration policy among the applications to guarantee performance [6, 7].

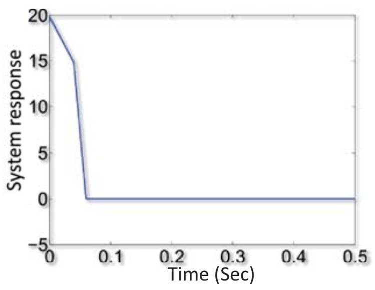

Figure 19: Purely time-triggered communication.

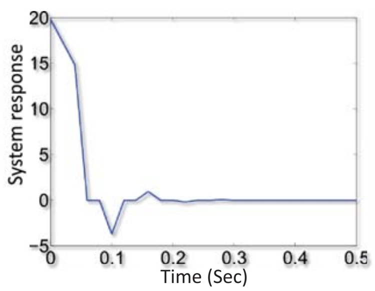

Figure 20: Mixed time-/event-triggered communication.

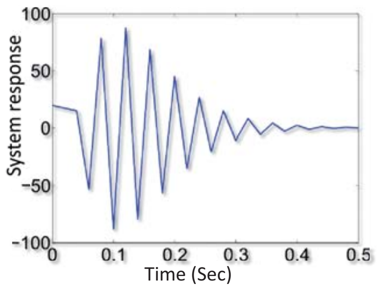

Figure 21: Purely event-triggered communication.

6 Concluding Remarks

Automotive in-vehicle networks are inherently complex in nature. In modern cars, several functionalities need feedback control loops. The traditional design flow performs controller and implementation platform design in a decoupled fashion which often leads to a non-optimal and overly conservative design. In this paper, it is shown using several design examples that a holistic cyber-physical design approach is more suitable for such complex design problem. There are two main aspects in such design methodology (i) joint control/architecture design or co-design, (ii) exploiting special properties of control application in the design process. Design examples demonstrate significant improvement in overall design.

References

[1] A. Cervin and P. Alriksson. Optimal on-line scheduling of multiple control tasks: A case study. In *ECRTS*, 2006.

[2] 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*, 14(4):776–787, 2006.

[3] D. Goswami, R. Schneider, and S. Chakraborty. Co-design of Cyber-Physical Systems via controllers with flexible delay constraints. In *ASP-DAC*, 2011.

[4] D. Goswami, R. Schneider, and S. Chakraborty. Re-engineering Cyber-Physical control applications for hybrid communication protocols. In *DATE*, 2011.

[5] S. Hong, X. S. Hu, and M. D. Lemmon. Reducing delay jitter of real-time control tasks through adaptive deadline adjustments. In *ECRTS*, 2010.

[6] A. Masrur, D. Goswami, S. Chakraborty, J. Chen, A. Annaswamy, and A. Banerjee. Timing analysis of cyber-physical applications for hybrid communication protocols. In *DATE*, 2012.

[7] A. Masrur, D. Goswami, R. Schneider, H. Voit, A. Annaswamy, and S. Chakraborty. Schedulability analysis of distributed Cyber-Physical applications on mixed time-/event-triggered bus architectures with retransmissions. In *IEEE SIES*, 2011.

[8] R. Schneider, D. Goswami, S. Zafar, M. Lukasiewycz, and S. Chakraborty. Constraint-driven synthesis and tool-support for FlexRay-Based automotive control systems. In *CODES+ISSS*, 2011.

[9] S. Soheil, P. Eles, Z. Peng, P. Tabuada, and A. Cervin. Dynamic scheduling and control-quality optimization of self-triggered control applications. In *IEEE RTSS*, 2010.

[10] X. Tang and J. Yu. Stability analysis for discrete time-delay systems. In *Conference on Networked Computing and Advanced Information Management*, 2008.

[11] H. Voit, A. Annaswamy, R. Schneider, D. Goswami, and S. Chakraborty. Adaptive switching controllers for systems with hybrid communication protocols. In *American Control Conference*, 2012.

[12] H. Voit, R. Schneider, D. Goswami, A. Annaswamy, and C. S. Optimizing hierarchical schedules for improved control performance. In *IEEE SIES*, 2010.

[13] G. C. Walsh, H. Ye, and L. G. Bushnell. Stability analysis of networked control systemss. *IEEE Trans. on Control System Technology*, 10(3):438–446, 2002.

[14] W. Wolf. Cyber-physical systems. *IEEE Computer*, 42(3):88 – 89, 2009.

[15] F. Zhang, K. Szwaykowska, W. Wolf, and V. J. Mooney. Task scheduling for control oriented requirements for Cyber-Physical Systems. In *IEEE RTSS*, 2008.