Multi-Objective Co-Optimization of FlexRay-based Distributed Control Systems

Debayan Roy*, Licong Zhang*, Wanli Chang†, Dip Goswami‡, Samarjit Chakraborty*
*TU Munich, Germany, †TUM CREATE, Singapore, ‡Eindhoven University of Technology, Netherlands
Email:*{debayan.roy, licong.zhang, samarjit}@tum.de, †wanli.chang@tum-create.edu.sg, ‡d.goswami@tue.nl

***Abstract*—Recently, research on control and architecture co-design has been drawing increasingly more attention. This is because these techniques integrate the design of the controllers and the architecture and explore the characteristics on both sides to achieve more efficient design of embedded control systems. However, there still exist several challenges like the large design space and inadequate trade-off opportunities for different objectives like control performance and resource utilization. In this paper, we propose a co-optimization approach for FlexRay-based distributed control systems, that synthesizes both the controllers and the task and communication schedules. This approach exploits some FlexRay protocol specific characteristics to reduce the complexity of the whole optimization problem. This is done by employing a customized control design and a nested two-layered optimization technique. Therefore, compared to existing methods, the proposed approach is more scalable. It also allows multi-objective optimization taking into account both the overall control performance and the bus resource utilization. This approach generates a Pareto front representing the trade-offs between these two, which allows the engineers to make suitable design choices.**

I. Introduction

Distributed control systems are commonly found in various domains including automotive, industrial automation and avionics. These systems are often safety-critical and have stringent timing requirements. In the automotive domain, the FlexRay-based network of electronic control units (ECUs), where a number of ECUs are connected through a FlexRay bus, is becoming increasingly more common for implementation of such systems. A distributed control application is partitioned into several software tasks mapped on different ECUs and tasks communicate through messages sent over the bus. As the scale and complexity of the distributed control systems have increased drastically in the past decade, the communication resource has become scarce, making efficient design a quite important issue. One reason for this is that new applications can be mapped on additional ECUs, whereas adding a bus cluster is much more difficult. Moreover, faster and more efficient hardware can increase the computational power of ECUs, but the bus bandwidth could not be easily increased. In conventional design paradigm, the embedded platform and the controllers are designed separately and then integrated [1], [2]. To ensure the safety of the control applications, there is often considerable conservativeness in such a design scheme and thus not leading to the most efficient one.

To address this problem, in recent years, the subject of control and platform co-design has emerged and has been drawing increasingly more attention [3], [4], [5]. This design paradigm combines the design of controllers and the underlying embedded platform together, often using a holistic approach, to explore the characteristics of both sides in order to reduce design conservativeness. The control design aims to find the sampling period and control gains for a control application. The platform parameters include the task schedules and (or) priorities, and communication schedules. Often a mathematical model with platform-specific constraints and design objectives is established and solved, which automates the design process and offers the designer the opportunity to co-design control and system according to specific objectives.

However, there still exist some challenges for the co-design problem in FlexRay-based distributed control systems. One challenge is the large design space. This is due to the fact that for various platform-specific behaviors such as sampling period, delay, jitter, etc., the control design problem does not fit in standard closed-form optimal control framework. Therefore, it often becomes necessary to perform exhaustive search to find the right system parameters such as poles. This results in a large design space for the controllers. In addition, the FlexRay-based platform requires the configuration of a huge number of parameters. A combined space with controller and FlexRay-based network and ECU parameter is huge and challenging to handle. Such synthesis problem requires considerable computational effort. This is aggravated by the increase in the system size and the whole problem easily becomes intractable. Secondly, existing approaches usually consider only the optimization of the control performance as the design objective and there is little design freedom for trade-offs between the control performance and the resource utilization.

In this paper, we address the aforementioned challenges by introducing a co-optimization approach that synthesizes simultaneously controllers, task and communication schedules for a FlexRay-based ECU network. The approach consists mainly of two stages, namely the control design stage and the co-optimization stage. This separation is necessary because we are dealing with a large design space combining the dimensions of both control and platform design. Therefore, we need to partition the whole space into smaller subspaces while considering all feasible regions in the design space by exploiting some domain-specific characteristics. In this case, it is possible because the sampling periods of the control applications can only take selected discrete values. In addition, only the sampling period influences the platform parameters, not the control gains. In the control design stage, we synthesize an optimal controller at each possible sampling period for each control application. This is done by using the pole placement control design method and exploring the design space of all possible pole values. By designing first the optimal controllers for the possible values of the sampling periods, we avoid unnecessary schedule synthesis for suboptimal or unstable controllers. In the co-optimization stage, we formulate a bi-objective optimization problem

and propose a customized method to generate a number of feasible design parameter sets, where each set represents a Pareto point reflecting the trade-off between the objectives of control performance and bus resource utilization. Here, making use of the discrete nature of the bus resource utilization, we turn a bi-objective optimization problem into a set of single-objective optimization problems. Each of these single-objective optimization problems may generate a Pareto point. To solve each problem, a nested two-layered technique is employed. This technique utilizes the fact that only a subset of all the parameters contributes to the objective, while the others do not, and thus partitions the whole optimization problem into two. The outer layer calculates the optimal value of the objective, whereas the inner layer finds a feasible parameter set corresponding to that value. The designer can then select a parameter set corresponding to a Pareto point in the Pareto front according to the design requirements. This approach can be trivially extended to similar architectures for safety-critical systems based on Time Division Multiple Access (TDMA) bus and time-triggered real-time operating system.

Related Work: The synthesis of FlexRay schedules is a well researched subject. [7] has provided a formulation to synthesize the schedules while minimizing the number of slots used. The task and network schedule co-synthesis problem based on a TDMA bus is studied in [8]. But these works focus mainly on the schedule synthesis problem of the processing units and bus system, while the aspect of control design is not considered. In the context of control and platform co-design, [4] has proposed a co-design method that integrates both the control design and the task and message scheduling and optimizes the overall control performance. Moreover, [5] has proposed a constraint-driven synthesis for the co-design of controllers with task and FlexRay schedules. A stable controller is designed first for each control application assuming one sampling period sensor-to-actuator delay, and based on the results the schedules are synthesized according to the pre-selected sampling periods. [9] has introduced a co-design method for such a system where the design of the controllers and the schedules are integrated into a holistic framework and the sampling period of the control applications and the schedules can both be synthesized while optimizing the control performance. This work considers both the variable sampling period and delay in the control design. However, [4], [5], [9] have not considered the trade-off between multiple optimization objectives. Furthermore, some approaches appear difficult to scale. For example, in the case study of [9], it already takes more than one hour to synthesize a system of 5 applications, amongst which 3 are control applications. Multi-objective optimization has been applied in scheduling of wireless communication networks [10], [11]. Both the time and the energy consumption are optimized in [10], when scheduling a wireless sensor network. In [11], radio resource scheduling is considered, aiming to optimize outage, capacity and throughput, for the given available resources. The significance of applying multi-objective optimization techniques in the problems naturally embedded with more than one design objective is that the trade-off among various objectives can be analyzed and that the decision maker is given the flexibility to choose a design point based on customized situations. But these related works do not address the specific problem in the control and platform co-design domain.

Contribution: The contribution of this paper is twofold: (i) Firstly, we propose a co-optimization technique for jointly optimizing the platform and the control parameters. Compared to existing approaches, our approach explores the Pareto front, involving both the objectives of overall control performance and bus resource utilization. Hence, it offers more freedom of design trade-offs between the two objectives, which is of practical relevance to resource-efficient design of distributed control systems. (ii) Furthermore, to tackle the huge design space including dimensions in both the control and platform design and the difficulty of model formulation in this case, the proposed approach exploits some control theoretic and platform-specific knowledge to partition the space in an intelligent fashion. Therefore it reduces the complexity of the model formulation and the computational effort. We further propose a nested two-layered optimization technique to reduce the synthesis time. The result shows that this approach can scale up to a system size of at least 24 control applications.

The rest of the paper is organized as follows. In Sec. II, we explain the setup we are considering including the FlexRay-based ECU architecture and the distributed feedback control applications. We then describe the proposed co-optimization approach in Sec. III, which is followed by the nested two-layered optimization technique in Sec. IV. In Sec. V, the experimental results based on a case study are presented, together with an analysis on the scalability of the proposed approach. Finally we conclude in Sec. VI.

II. Problem Formulation

We consider a distributed architecture where a set of ECUs represented by $E_i \in \mathcal{E}$ are connected through a FlexRay bus. A number of control applications denoted by $C_i \in \mathcal{C}$ are mapped on such an embedded platform. Each control application is partitioned into several dependent software tasks performing functions like sensor reading, computation and actuation. These tasks are typically mapped on physically distributed ECUs and the data between the tasks are sent via the FlexRay bus.

A. FlexRay-based ECU Network

ECU task model: In this paper, we consider the case where the scheduling scheme of the real-time operating system on the ECUs follows a time-triggered non-preemptive scheme. The tasks of the control applications are considered to be periodic tasks. Such a task τ_i can be defined by the tuple $\tau_i = \{p_i, o_i, e_i\}$, where p_i, o_i and e_i denote respectively the period, the offset and the WCET of the task. Thus if we define $t(\tau_i, k)$ and $\tilde{t}(\tau_i, k)$ as the starting and the latest finishing time of the kth ($k \in \mathbb{Z}^*$) instance of task τ_i , then

$$t(\tau_i, k) = o_i + kp_i \tag{1}$$

$$\tilde{t}(\tau_i, k) = o_i + kp_i + e_i. \tag{2}$$

A set of *communication tasks* are required besides the application tasks. The communication task on the sending

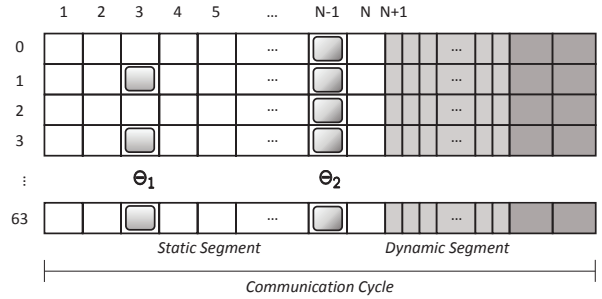

Fig. 1: An example of FlexRay schedules.

ECU writes the data produced by the application tasks into the corresponding *transmit buffers* of the communication controller, and on the receiving ECU, it reads the data from the corresponding *receiver buffers* and forwards them to the application tasks. The nature of these communication tasks depends on the specific implementation. Here we consider that the execution time of all communication tasks is bounded by ϵ and we schedule the communication task directly after its corresponding application task at the sending side and directly before the application task at the receiving side.

FlexRay communication: FlexRay [18] is an automotive communication protocol usually applied for safety-critical applications. Being a hybrid protocol, it offers both Time-Triggered (TT) and Event-Triggered (ET) communication services. FlexRay is organized as a series of *communication cycles*, the length of which we denote as T_{bus}. Each communication cycle contains mainly: the *static segment* (ST) and optionally *dynamic segment* (DYN), where the TT and ET communication services are implemented respectively. The static segment applies the TDMA scheme and is split into a number of *static slots* of equal length Δ. Here, we represent the slots on the static segment as $\mathcal{S} = \{1, 2, ..., N\}$. Once a static slot is assigned, if no data is sent in a specific communication cycle, the static slot will still be occupied. The dynamic segment follows a Flexible TDMA (FTDMA) approach, where the segment is divided into M number of *mini-slots* of equal length δ. A *dynamic slot* is a logical entity, which can consist of one or more mini-slots, depending on whether data is sent on the slot and how much data is sent. Once a dynamic slot is assigned, if no data is sent in a communication cycle, only one mini-slot is consumed. If data is to be sent, a number of mini-slots are occupied to accommodate the data. We define the dynamic slots as $S_{DYN} = \{N + 1, ..., N + M\}$.

The communication cycles are organized as sequences of 64 cycles. In a sequence, each communication cycle is indexed by a *cycle counter* which counts from 0 to 63 and is then set to 0. A FlexRay schedule can be defined as $\Theta_i = (S_i, B_i, R_i)$, where S_i represents the slot number, B_i represents the *base cycle* and R_i, the *repetition rate*. The repetition rate is the number of communication cycles that elapse between two consecutive transmissions of the same frame and takes the value $R_i \in \{2^n | n \in \{0, ..., 6\}\}$. The base cycle is the offset of the cycle counter. The sequence of 64 communication cycles and some examples of FlexRay schedules are shown in Fig. 1. In the context of this paper, we consider the FlexRay Version 3.0.1 [18], where *slot multiplexing* amongst different ECUs is allowed. It means that a particular slot S_i can be assigned to different ECUs in different communication cycles. We further consider all messages are sent over the static segment of the FlexRay bus, i.e., on the static slots. The starting and ending time of the kth instance ($k \in \mathbb{Z}^*$) of the FlexRay schedule Θ_i, which are denoted respectively as $t(\Theta_i, k)$ and $\tilde{t}(\Theta_i, k)$, can be defined as

$$t(\Theta_i, k) = B_i T_{bus} + k R_i T_{bus} + (S_i - 1)\Delta, \tag{3}$$

$$\tilde{t}(\Theta_i, k) = B_i T_{bus} + k R_i T_{bus} + S_i \Delta. \tag{4}$$

In this paper, we consider the bus resource utilization as the total amount of bandwidth of the static segment that is allocated to the ECUs. In terms of static segment, this can be translated into the total number of static slots assigned in every 64 communication cycle sequence. Let Γ denote the set of all FlexRay schedules on the static segment, where $\Theta_i \in \Gamma$, then the resource utilization U can be defined as

$$U = \sum_{\Theta_i \in \Gamma} \frac{64}{R_i}, \tag{5}$$

where the smaller the value of U, the better is the resource utilization.

B. Distributed Feedback Control Systems

Feedback control systems: A continuous-time linear control system can be represented by the following differential equation

$$\begin{aligned} \dot{x}(t) &= A_c x(t) + B_c u(t) \\ y(t) &= C_c x(t), \end{aligned} \tag{6}$$

where $x(t)$, $y(t)$ and $u(t)$ denote respectively the system state vector, the output and the control input of the system and A_c, B_c and C_c denote respectively the system, input and output matrix. In an embedded implementation, the system is discretized according to a sampling period h and the corresponding discrete system can be represented as

$$\begin{aligned} x[k+1] &= A_d x[k] + B_d u[k] \\ y[k] &= C_d x[k], \end{aligned} \tag{7}$$

where $x[k]$, $y[k]$ and $u[k]$ denote the state vector, the output and the control input respectively at the kth time instant ($k \in \mathbb{Z}^*$) and

$$A_d = e^{A_c h}, B_d = \int_0^h (e^{A_c t} dt) \cdot B_c, C_d = C_c. \tag{8}$$

Distributed implementation: A control application implemented on a distributed embedded system can be partitioned into three different types of software tasks: (i) *sensor task* measures the system states (using sensors) of the physical system if measurable. (ii) *controller task* computes the controller input based on the measured system states. (iii) *actuator task* applies the control input (using actuators) to the physical system. These software tasks are usually mapped onto different ECUs, due to physically distributed topology of sensors and actuators. Here we denote the sensor, controller and actuator task of a control application C_i respectively as $\tau_{s,i}$, $\tau_{c,i}$ and $\tau_{a,i}$. Without

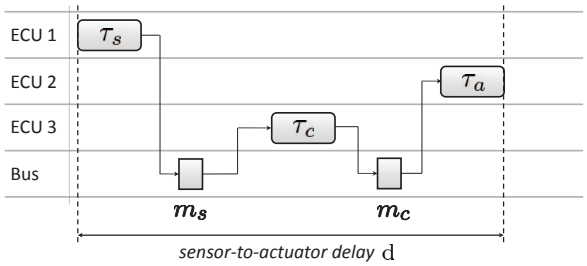

Fig. 2: Distributed embedded control application.

loss of generality, we assume that three tasks are mapped on different ECUs. Then the sensor values measured by $\tau_{s,i}$ are sent on the bus through message $m_{s,i}$ and the control input is sent as message $m_{c,i}$. The time between the start of sensor task and the end of actuator task is defined as the *sensor-to-actuator delay*, denoted as d. As shown in Fig. 2, this delay depends on the interplay between the task and communication schedules.

Controller design: The control input $u[k]$ of a distributed control application based on state-space feedback control can be designed as

$$u[k] = Kx[k - \left\lceil \frac{d}{h} \right\rceil] + Fr, \tag{9}$$

where K is the feedback gain to stabilize the system and F is the static feedforward gain. r represents the reference value for $y[k]$ to reach. d and h are respectively the sensor-to-actuator delay and the sampling period. In the case where $d \ll h$, it can be approximated as $u[k] = Kx[k] + Fr$ and if $d \approx h$, the control input can be approximated as $u[k] = Kx[k-1] + Fr$.

Control performance: There are different metrics to measure the performance of a control system. Here we consider two common metrics to measure the control performance. (i) The steady-state performance of a control application can be commonly measured by a *cost function* [5], which in the discrete case can be represented as

$$J = \sum_{k=0}^{n} [\lambda u[k]^2 + (1-\lambda)\sigma[k]^2]h, \tag{10}$$

where λ is a weight taking the value between 0 and 1, $u[k]$ is the control input and $\sigma[k] = |r - y[k]|$ is the tracking error. We further consider the (ii) *settling time* ξ as another control performance, where ξ denotes the time necessary for the system to reach and remain within 1% of the reference value

$$J = \xi. \tag{11}$$

Depending on the specific requirements of the control application, one of the aforementioned performance metric can be used. In both cases, smaller value of J implies better control performance. For a specific control application C_i, J_i depends both on the sampling period h_i and the control gains K_i and F_i. In a system consisting of multiple control applications with different plant models and performance metrics, we need to normalize the control performance in order to compare and

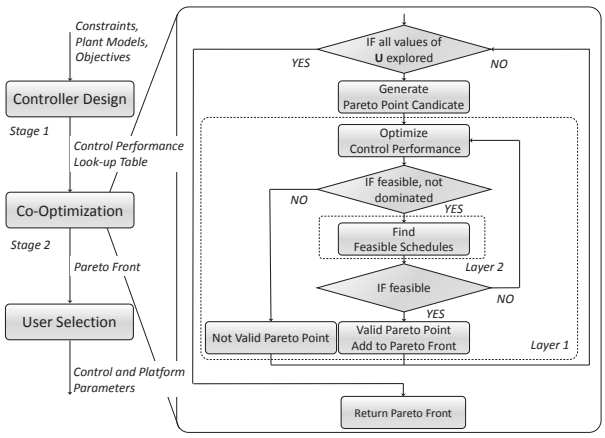

Fig. 3: Design flow of the co-optimization approach.

combine them. Each control system C_i with control performance J_i must satisfy some control performance requirement J_i^r defined by the user. Thus we can normalize it likewise as follows

$$J_i^n = \frac{100 \cdot J_i}{J_i^r} \tag{12}$$

and thus the overall control performance of a set of control applications $\mathcal{C}$ can be represented as a weighted sum

$$J_o = \sum_{C_i \in \mathcal{C}} w_i J_i^n, \tag{13}$$

where w_i stands for the weight.

C. Co-optimization Problem

Consider a control application C_i consisting of sensor task $\tau_{s,i}$, controller task $\tau_{c,i}$ and actuator task $\tau_{a,i}$ and messages carrying the sensor data $m_{s,i}$ and control input $m_{c,i}$. Messages $m_{s,i}$ and $m_{c,i}$ are each mapped onto a specific FlexRay schedule. We denote these schedules as $\Theta_{s,i}$ and $\Theta_{c,i}$ respectively. The control parameters for C_i include the sampling period h_i and the control gains K_i and F_i. Then the co-optimization problem boils down to finding a set of parameters for each $C_i \in \mathcal{C}$, which can be denoted as $par_i = \{\tau_{s,i}, \tau_{c,i}, \tau_{a,i}, \Theta_{s,i}, \Theta_{c,i}, h_i, K_i, F_i\}$, while optimizing the total FlexRay bus resource utilization as in Eq. (5) and the overall control performance as in Eq. (13). Here, we further define the control parameters of C_i as $par_i^c = \{h_i, K_i, F_i\}$ and the embedded platform parameters as $par_i^s = \{\tau_{s,i}, \tau_{c,i}, \tau_{a,i}, \Theta_{s,i}, \Theta_{c,i}\}$, where $par_i = par_i^s \cup par_i^c$. The parameter set of the whole system is represented as $\mathcal{P}$, where $par_i \in \mathcal{P}$.

III. Co-Optimization Approach

A. Design Flow

In this paper, we propose an approach that can efficiently solve the co-optimization problem formulated in Sec. II. Fig. 3 shows the design flow of the proposed approach. The whole design process is divided into two stages. In the first stage, for each control application, possible controllers that optimize the control performance at different sampling periods are synthesized and the results are recorded in a look-up table. In this stage, for each control application we employ the pole placement design method and search the pole space for

optimal controllers at each possible sampling period. In the second stage, the co-optimization stage, we synthesize both the control and the platform parameters based on the constraints, objectives and the look-up tables obtained in the first stage. Here, we formulate a bi-objective optimization problem and propose a customized approach to generate a Pareto front of the two objectives considered. Pareto point candidates are iteratively identified, exploring the characteristics of the bus resource utilization objective, i.e., it can take only selected discrete values. For each candidate, a nested two-layered optimization process finds a feasible set of parameters that optimizes the control performance or proves that such a point is not feasible. The output of the second stage is a Pareto front representing optimal parameter sets taking both objectives into account. Based on this, the designer can select one set of parameters that is the most suitable for the overall design requirements. The control design stage and the co-optimization stage of this approach will be explained in details in the following subsections.

B. Controller Design

Besides the control plant model, the performance J_i of the control application C_i depends mainly on three factors: (i) the sampling period h_i, (ii) the sensor-to-actuator delay d_i and (iii) the control gains K_i and F_i. Depending on each combination of the sampling period and delay, we have to design a set of optimal control gains. In this paper, we consider schedules for the control tasks and the messages leading to the case where the delay equals to the sampling period, i.e., $d_i = h_i$. This would reduce the dimensions of the design space from all three factors (i) - (iii) to only (i) and (iii), thus reducing the complexity and enhancing the scalability. It should be noted that our approach can be easily adapted to other cases with a fixed delay value (e.g., $d_i = D_i$, where D_i is a constant) or a delay value proportional to the sampling period (e.g. $d_i = \psi h_i$) by replacing the controller design method [16]. However, this might have some influence on the scalability of the approach. With $d_i = h_i$, the closed-loop system experiences one sampling period delay and we use the pole placement method reported in [6] for such delayed system. This method enables to find the control gains according to a set of specified poles. For such a delayed system, pole placement is only possible for a *restricted* set of specified poles as indicated in [6]. The design space for the control design then consists of two dimensions, namely the sampling period and the closed-loop system poles. To the best of our knowledge, there is no standard closed-form optimal control design framework that can be directly applied in such a delayed system. Therefore, the naive way is to exhaustively search the whole design space, which can be quite computationally costly. However, we can make use of the fact the sampling period can only take discrete values to prune the design space. Since each control application C_i is implemented by the tasks $\tau_{s,i}$, $\tau_{c,i}$, $\tau_{a,i}$ and messages $m_{s,i}$, $m_{c,i}$, there is a dependency between the sampling period h_i and the repetition rate of the messages $R_{s,i}$, $R_{c,i}$, which can be represented as

$$h_i = R_{s,i} T_{bus} = R_{c,i} T_{bus}. \tag{14}$$

Due to the fact that $R_{s,i}$, $R_{c,i}$ can only take discrete values in $\{2^k | k \in \{0, ..., 6\}\}$, the choice of h_i is also constrained to the corresponding discrete values.

We can denote the control performance as $J_i = f(h_i, K_i, F_i)$. Then, the control performance at each discrete value $h_i^k = 2^k T_{bus}$ of the sampling period can be represented as $J_i(h_i^k) = g(K_i^k, F_i^k)$. The purpose of the controller design step is to determine the control gains for each possible value of the sampling period that optimizes the control performance. Here we search the space of the poles that stabilizes the system. The search is carried out with a specified granularity in the restricted pole space. Since this search discretizes the pole space, it does not guarantee to obtain the optimal controller. However, the smaller the search granularity, the better are the chances of getting a controller closer to the optimum. Let us assume that there exists a set of control gains K_i^{k*}, F_i^{k*}, that optimizes the control performance to G_i^{k*} at sampling period h_i^k, then we could represent the optimal control performance at h_i^k as $J_i^*(h_i^k) = G_i^{k*}$. The control design problem can be translated into the problem of finding for each discrete value h_i^k, a set of gains K_i^{k*}, F_i^{k*} that optimizes the control performance $J_i(h_i^k)$ to the value of G_i^{k*}.

Algorithm 1 Control design at each sampling period for a control application

Input: $[A_c, B_c, C_c]_i$, h_i^k
Output: G_i^{k*}, K_i^{k*}, F_i^{k*}
Initialization: $G_i^{k*} = \infty$, $K_i^{k*} = 0$, $F_i^{k*} = 0$
1: $[\sigma_\rho, isStabilizable] =$ calculatePoleSum$([A_c, B_c, C_c]_i, h_i^k)$
2: **if** $true == isStabilizable$ **then**
3: $\zeta =$ getSystemOrder$([A_c, B_c, C_c]_i)$
4: $P = \{\rho_i\}$, where $1 \leq i \leq \zeta$,
5: $P =$ getFirstPoleSet()
6: **while** $P \neq null$ **do**
7: **if** $|\sum \rho_i - \sigma_\rho| \leq gran_\rho/2$ **then**
8: $\rho_\zeta = \sigma_\rho - \sum_{1,\zeta-1} \rho_i$
9: $[K_i^k, F_i^k] =$ calculateGains$([A_c, B_c, C_c]_i, h_i^k, P)$
10: $J_i^k =$ calculatePerformance$([A_c, B_c, C_c]_i, h_i^k, K_i^k, F_i^k)$
11: **if** $J_i^k < G_i^{k*}$ **then**
12: $G_i^{k*} = J_i^k$
13: $K_i^{k*} = K_i^k$
14: $F_i^{k*} = F_i^k$
15: **end if**
16: **end if**
17: $P =$ getNextPoleSet$(P, grad_\rho)$
18: **end while**
19: **end if**
20: **return** G_i^{k*}, K_i^{k*}, F_i^{k*}

The algorithm for designing an optimal controller at each discrete value of sampling periods $\{h_i^k\}$ for a control application C_i is summarized in Algorithm 1. It takes as input, the continuous-time plant model denoted by $[A_c, B_c, C_c]_i$ and the sampling period value h_i^k and returns the value of the performance G_i^{k*} and gains K_i^{k*}, F_i^{k*} of the best controller. $gran_\rho$ defines the granularity of the pole search and is a parameter of the algorithm. Here, ρ_i denotes a closed-loop pole and P denotes a set of all closed loop system poles. G_i^{k*} is initialized as infinity and the control gains are initialized as 0. Line 1 calculates the constraints on the sum of all closed-loop poles that can stabilize the plant according to [6], which is denoted as σ_ρ. It also returns the boolean variable $isStabilizable$ denoting whether the system can be stably

controlled at all. If it is not possible, the algorithm does not return a valid controller (Line 2). Line 3 calculates the order ζ of the closed-loop system. In Line 5, all the poles in the pole set P are initialized as -1. This is because for a discrete control system, poles of the closed-loop system needs to be inside the unit circle. Line 6-18 traverses all the pole sets possible by incrementing a specific pole by the granularity $gran_\rho$ in each iteration (Line 17) until all poles reach the value of 1. If the sum of poles lies in the vicinity of σ_ρ (Line 7), we adjust the value of ρ_ζ so that the constraint on the sum of poles is strictly satisfied (Line 8). Then the pole set can stabilize the system and we compute the control gains (Line 9) according to [6] and the corresponding performance of the closed-loop system (Line 10) by simulation until a user-defined time value. If the performance is better than the current best performance, the current best control gains and performance are then updated to the new values (Line 11-15).

After this stage, we can basically formulate a look-up table for each control application C_i where for each of the possible sampling period h_i^k we can assign an optimal control performance G_i^{k*} corresponding to the control gains K_i^{k*}, F_i^{k*}. In the co-optimization stage, the sampling periods of all the control applications will be used as variables in the problem formulation. The objective of the overall control performance can therefore be formulated as a function of the sampling periods. Depending on the values of the sampling periods, the control laws can be selected by referring to these look-up tables. Therefore, the sampling periods here serve as the main interface for the interplay between the control design stage and the co-optimization stage.

C. Optimization Problem Formulation

With the results of the control design step, the whole problem can then be formulated into a constrained programming model. The parameters to be synthesized here include (i) the sampling period of the control applications, (ii) the task schedules and (iii) the message schedules. The constraints consist of the platform constraints and the control performance constraints. As the optimization objectives, we consider both the bus resource utilization and the overall control performance.

1) Constraints: The system constraints for the FlexRay-based ECU system are well-studied and discussed in [5], [7], [9]. In our framework, we will use the majority of the constraints formulated there.

(C1) Sampling period constraint: We know that all the tasks and messages of a control application must have the same period of repetition which is also the sampling period of the system. This constraint can be formulated as

$$\forall C_i \in \mathcal{C}, x \in \{s,c,a\}, y \in \{s,c\}, \quad p_{x,i} = R_{y,i} T_{bus} = h_i. \tag{15}$$

(C2) Dataflow constraint: In a control application all task executions and message transmission have to be carried out in the correct temporal order, as illustrated in Fig. 2.

1) Relation between $\tau_{s,i}$ and $\Theta_{s,i}$

$$\forall k \in \mathbb{Z}^*, C_i \in \mathcal{C}, \quad \tilde{t}(\tau_{s,i}, k) + \epsilon < t(\Theta_{s,i}, k). \tag{16}$$

2) Relation between $\Theta_{s,i}$ and $\tau_{c,i}$

$$\forall k \in \mathbb{Z}^*, C_i \in \mathcal{C}, \quad \tilde{t}(\Theta_{s,i}, k) < t(\tau_{c,i}, k) - \epsilon. \tag{17}$$

3) Relation between $\tau_{c,i}$ and $\Theta_{c,i}$

$$\forall k \in \mathbb{Z}^*, C_i \in \mathcal{C}, \quad \tilde{t}(\tau_{c,i}, k) + \epsilon < t(\Theta_{c,i}, k). \tag{18}$$

4) Relation between $\Theta_{c,i}$ and $\tau_{a,i}$

$$\forall k \in \mathbb{Z}^*, C_i \in \mathcal{C}, \quad \tilde{t}(\Theta_{c,i}, k) < t(\tau_{a,i}, k) - \epsilon. \tag{19}$$

(C3) Sensor-to-actuator delay constraint: The constraint stating that the sensor-to-actuator delay for the control applications is equal to exactly one sampling period can be formulated as

$$\forall k \in \mathbb{Z}^*, C_i \in \mathcal{C}, \quad \tilde{t}(\tau_{a,i}, k+1) - t(\tau_{s,i}, k) = h_i. \tag{20}$$

(C4) Non-overlapping task constraint: In a time-triggered non-preemptive scheduling scheme as considered in this paper, when more than one task is mapped on an ECU, they must be scheduled in such a way that they do not overlap. This can be formulated as a constraint as

$$\begin{gathered}
\forall \quad C_i, C_j \in \mathcal{C}, \quad x,y \in \{s,c,a\}, \quad E_k \in \mathcal{E} \\
\forall \quad \{m \in \mathbb{Z}^* | 0 \le m < lcm(p_{x,i}, p_{y,j})/p_{x,i}\}, \\
\{n \in \mathbb{Z}^* | 0 \le n < lcm(p_{x,i}, p_{y,j})/p_{y,j}\} \\
\textbf{if} \quad \tau_{x,i}, \tau_{y,j} \in \mathcal{T}_{E_k} \quad \textbf{then} \\
\tilde{t}(\tau_{x,i}, m) + \epsilon \cdot \quad (x \in \{s,c\}) < t(\tau_{y,j}, n) - \epsilon \cdot \quad (y \in \{c,a\}) \\
\textbf{or} \\
\tilde{t}(\tau_{y,j}, n) + \epsilon \cdot \quad (y \in \{s,c\}) < t(\tau_{x,i}, m) - \epsilon \cdot \quad (x \in \{c,a\}),
\end{gathered} \tag{21}$$

where $\mathcal{T}_{E_k}$ denotes the set of all tasks mapped on ECU E_k. $(.)$ is the indicator function and takes the value of 1 if the input is true and 0 if otherwise.

(C5) Non-overlapping message constraint: FlexRay messages must be scheduled in such a way that no two messages share the same slot in the same cycle. This constraint can be established as

$$\begin{gathered}
\forall \quad C_i, C_j \in \mathcal{C}, \quad x,y \in \{s,c\} \\
\forall \{n \in \mathbb{Z}^* | 0 \le n < max(R_{x,i}, R_{y,j})/R_{x,i}\}, \\
\{m \in \mathbb{Z}^* | 0 \le m < max(R_{x,i}, R_{y,j})/R_{y,j}\} \\
\textbf{if} \quad S_{x,i} == S_{y,j} \quad \textbf{then} \quad B_{x,i} + nR_{x,i} \neq B_{y,j} + mR_{y,j}.
\end{gathered} \tag{22}$$

(C6) FlexRay scheduling constraint: Taking into consideration the scheduling constraints imposed by the FlexRay protocol, we need to constrain $S_{x,i}$ and $B_{x,i}$ as

$$\forall \ C_i \in \mathcal{C}, \ \ x \in \{s,c\}, \ \ 1 \le S_{x,i} \le N \tag{23}$$

$$\forall \ C_i \in \mathcal{C}, \ \ x \in \{s,c\}, \ \ 0 \le B_{x,i} < R_{x,i}, \tag{24}$$

where N is the number of static slots. In addition, the bus utilization on the static segment must not exceed the total number of static slots available in 64 communication cycles. This can be formulated as

$$U \le 64N. \tag{25}$$

(C7) ECU scheduling constraint: On the ECUs, for task schedules, we must consider

$$\forall \ C_i \in \mathcal{C}, \ \ x \in \{s,c,a\}, \ \ 0 \le o_{x,i} + e_{x,i} < p_{x,i}. \tag{26}$$

The utilization constraints on an ECU can be represented as

$$\forall E_k \in \mathcal{E}, x \in \{s,c,a\}, \sum_{\tau_{x,i} \in \mathcal{T}_{E_k}} \frac{e_{x,i} + \epsilon + \epsilon \cdot \quad (x \in \{c\})}{p_{x,i}} \le 1, \tag{27}$$

denoting that ECU cannot be more than 100 percent loaded.

(C8) Performance constraint: For each control system C_i with sampling period h_i, user specifies a control performance

requirement J_i^r. As mentioned in Sec. III-B, we have developed a look-up table for each control system which contains the performance of seven possible controllers corresponding to seven possible sampling periods. Therefore, the domain of h_i, denoted as $dom[h_i]$ is constrained according to control performance requirement as

$$\forall k \in \{0, 1, ...6\}, \quad J_i^*(h_i^k) \leq J_i^r \iff h_i^k \in dom[h_i]. \tag{28}$$

Now, let J_i represent the control performance of C_i. Mathematically, we can formulate

$$h_i == 2^k T_{bus} \iff J_i == J_i^*(h_i^k). \tag{29}$$

2) Optimization Objectives: As the objectives for the optimization problem, we consider the overall system control performance and the bus resource utilization.

(O1) Overall system control performance:

$$J_o = \sum_{C_i \in \mathcal{C}} w_i J_i^n = \sum_{C_i \in \mathcal{C}} w_i \sum_k \mu_{i,k} J_i^{n*}(h_i^k), \tag{30}$$

where $\mu_{i,k}$ are binary variables satisfying

$$\sum_k \mu_{i,k} = 1, \tag{31}$$

and $J_i^{n*}(h_i^k)$ represents the normalized optimal control performance of C_i at h_i^k, which can be formulated as

$$J_i^{n*}(h_i^k) = \frac{100 J_i^*(h_i^k)}{J_i^r}. \tag{32}$$

(O2) Resource Utilization : The resource utilization in this case can be defined as

$$U = \sum_{C_i \in \mathcal{C}} (\frac{64}{R_{s,i}} + \frac{64}{R_{c,i}}) = \sum_{C_i \in \mathcal{C}} \frac{128 T_{bus}}{h_i}. \tag{33}$$

The value of the resources utilization can only take certain discrete values and is bounded by the upper and lower limit U^+ and U^-, which can be expressed as

$$\begin{aligned} U^+ &= \sum_{C_i \in \mathcal{C}} \frac{128 T_{bus}}{\max\limits_{h_i \in dom[h_i]}(h_i)}, \\ U^- &= \sum_{C_i \in \mathcal{C}} \frac{128 T_{bus}}{\min\limits_{h_i \in dom[h_i]}(h_i)}. \end{aligned} \tag{34}$$

D. Multi-objective Optimization

As discussed above, the control and system co-design of the setting considered can be formulated as a constrained optimization problem with two objectives, namely the bus resource utilization and overall control performance. There exist several methods dealing with multi-objective optimization. A simple way is to convert the multiple objectives into one single objective with scalarization. However, the problems using this method here are (i) both of the objectives are completely different in nature and difficult to be combined as a single metric, and (ii) it would not be possible for the designer to fathom the design trade-off, which is necessary since the two design objectives are noticed to be often conflicting. In this case, a much more informative and designer-friendly approach is to first generate a Pareto front and let the designer explore the trade-off between the two objectives according to his customized preference.

In computing the Pareto points forming up the Pareto front, there are two requirements. Firstly, the obtained design points cannot be dominated by any other point in the objective space. In other words, there can be no point that is better than the solution points in both objectives. Secondly, the Pareto front should have a good space distribution. Design points too close to each other are of little help. It is noted that a Pareto point can be obtained by assigning a weight to each objective depending on its importance and optimizing the sum of all objectives. By changing the set of weights, different solutions can be obtained after solving the single-objective optimization problem. With this method, the first requirement on dominance is guaranteed. However, well distributed weights do not necessarily generate well distributed Pareto points, which means that the second requirement on distribution might not be fulfilled.

In this work, we propose a customized optimization approach to obtain the desired Pareto front. Since the objective on resource utilization U is discrete and only takes a limited number of integers, we first compute the maximum and minimum bus utilization U^+ and U^-, which bound the set of U. For each possible value of U from U^- to U^+, i.e., given the equality constraint on U, we solve the optimization problem with J_o as the single objective and obtain a solution. The additional constraint is that J_o of this solution has to be better than J_o of the last solution, in order to ensure that all solutions are non-dominated, which corresponds to the first requirement discussed above. With this method, the obtained Pareto points have a good distribution on the resource utilization, since for each possible value of U, there is a solution generated, as long as there exists a non-dominated solution. This fulfills the second requirement discussed above. This customized multi-objective optimization technique is adapted from [17], where one objective is translated into a constraint and the other is optimized.

Algorithm 2 Multi-objective optimization

Output: Ω
Initialization: Ω = {}, $J^+ = \infty$
1: U^- = calculateMinU()
2: U^+ = calculateMaxU()
3: **for** $U_i = U^-$ **to** U^+ **do**
4: $[\mathcal{P}_i, isPareto, J_{o,i}]$ = findCandidate(U_i, J^+)
5: **if** $true == isPareto$ **then**
6: $J^+ = J_{o,i}$
7: $\omega_i = [U_i, J_{o,i}, \mathcal{P}_i]$
8: Ω.addElement(ω_i)
9: **end if**
10: **end for**
11: **return** Ω

Algorithm 2 shows the whole multi-objective optimization. Here we denote a Pareto point as $\omega_i = \{U_i, J_{o,i}, \mathcal{P}_i\}$, where U_i stands for a discrete value of U and $J_{o,i}$, $\mathcal{P}_i$ represent respectively the corresponding optimal overall control performance and parameter set. The entire Pareto front is denoted as $\Omega = \{\omega_i\}$. In Algorithm 2, the Pareto front Ω is initialized as an empty set and J^+ as infinity. Line 1-2 calculate the minimal and maximal value of U according to Eq. (34). Then in the *for loop* between Line 3 and Line 10, the algorithm traverses all possible values of U. For each U value, the nested two-layered optimization (Line 4) is called to find a feasible parameter set that optimizes the control performance and can be considered

as a Pareto point (i.e. it is not dominated by other points). The nested two-layered optimization is described in the next section. If such a set can be found, it will be added to the Pareto front Ω (Line 8).

IV. Nested Two-Layered Optimization

As already discussed in the previous section, the co-optimization problem with two objectives is turned into a series of single-objective optimization problems, where each may generate a Pareto point on the Pareto front. Popular approaches for solving such optimization problems include Mixed Integer Linear Programming (MILP) or meta-heuristic methods. However, MILP is not applicable in this case. This is due to two facts. First, some constraints are not linear and cannot be linearized in a straightforward way. Second, formulating the whole problem into one single model would cause the number of variables and constraints to explode, thus making it extremely computationally expensive and not solvable for larger system sizes. On the other hand, although meta-heuristic methods like evolutionary algorithms are often applied to deal with high complexity, their capability to handle constraints is often limited. As discussed in Sec. III-C, the design space of this particular optimization problem is highly constrained. Therefore, it is difficult for meta-heuristics to find a solution that respects all constraints and prevents them from being effective.

In this work, considering that some decision variables only appear in constraints, but are not related to the objective, we propose a nested two-layered technique to solve this optimization problem. On Layer 1, the outer layer, we consider only constraint (C8) and an equality constraint translated from (O2) and optimize the (O1). Decision variables related to the objectives, i.e., the sampling periods, are determined. On Layer 2, the inner layer, we synthesize the remaining decision variables satisfying the constraints (C1) - (C7) based on the results of Layer 1. This process is iterative in the way that if the synthesis fails in Layer 2, we go back to the Layer 1 for the next best solution. This optimization technique ensures optimality and also efficiency. The success of this nested two-layered technique is due to the fact that the feasible region of the design space has good objective values in this problem.

The nested two-layered optimization is represented in Algorithm 3. This function takes as input, the value of U and J^+. J^+ represents the best control performance amongst the already identified Pareto points. In each iteration of the while loop (Line 1-19), the algorithm starts by trying to find the most suitable candidates based on the upper and lower bound of the control performance J^+ and J^- (Line 2). All the determined candidates have the same overall control performance. One of the candidates can possibly represent a Pareto point. In function "optimizeCP" in Line 2, a MILP model is formulated and solved according to the objective of overall control performance. The candidates are each represented by a set of control parameters $\mathcal{P}_j^c$, where all the candidates found are represented by the set $\{\mathcal{P}_j^c\}$. If a set of candidates can be found, the lower bound of the control performance J^- will be updated to the current performance value (Line 6), in order to exclude already evaluated candidates in the next iteration. Then the algorithm will evaluate them one by one. For each candidate, Layer 2 (Line 9) is called to find feasible schedules. This is represented by the function "findSchedules" in Line 9.

Algorithm 3 findCandidate – Nested two-layered optimization

Input: U, J^+
Output: $\mathcal{P}$, $isPareto$, J
Initialization: $solFound = false$, $J^- = 0$
1: **while** $false == solFound$ **do**
2: $[\{\mathcal{P}_j^c\}, J, isFeasibleU] =$ optimizeCP$(U, J-, J+)$
3: **if** $false == isFeasibleU$ **then**
4: **break**
5: **else**
6: $J^- = J$
7: $\bar{\mathcal{P}}^c = \{\mathcal{P}_j^c\}$.getFirstSet()
8: **while** $\bar{\mathcal{P}}^c \neq \{\}$ **and** $false == solFound$ **do**
9: $[\bar{\mathcal{P}}^s, isFeasibleH] =$ findSchedules$(\bar{\mathcal{P}}^c)$
10: **if** $true == isFeasibleH$ **then**
11: $\mathcal{P} = \bar{\mathcal{P}}^c \cup \bar{\mathcal{P}}^s$
12: $solFound = true$
13: **break**
14: **else**
15: $\bar{\mathcal{P}}^c = \{\mathcal{P}_j^c\}$.getNextSet()
16: **end if**
17: **end while**
18: **end if**
19: **end while**
20: **if** $true == solFound$ **then**
21: **return** $\mathcal{P}$, J, $isPareto = true$
22: **else**
23: **return** $\mathcal{P} = \emptyset$, $J = 0$, $isPareto = false$
24: **end if**

In this function, another MILP model is formulated and solved without objective for a feasible schedule set. If such a set can be found for a candidate, the algorithm considers this candidate a valid Pareto point and stop evaluating further ones (Line 10-13). The algorithm also stops once it has rendered all the already generated candidates infeasible and no further candidates can be found (Line 3-4). In the end, if a feasible solution can be found (Line 20), then the function returns the corresponding parameter set $\mathcal{P}$, performance J and a true flag. Otherwise, it returns a false flag, an empty parameter set and zero control performance.

V. Results

In this section, we use a case study to (i) illustrate the design flow of the proposed approach and to (ii) show the importance of the Pareto front for the evaluation of trade-offs for distributed control systems design. Furthermore, an analysis is provided to show the scalability of the approach.

A. Case Study

In this case study, we use a system motivated by the applications in the automotive domain. Due to the confidentiality issue, it is difficult to find a case study system actually applied in the industry and obtain all the details including the mathematical model of the plants and the task and message models. Therefore, we use a synthetic case study, consisting of a FlexRay based ECU network implementing 5 control applications typical to the automotive domain.

Plant Models: We consider a system consisting of 5 control applications $\mathcal{C} = \{C_1, C_2, C_3, C_4, C_5\}$. For each of the control applications, we use a plant model derived from the automotive domain. C_1 to C_5 represent respectively the DC motor speed control (DCM), the car suspension system (CSS), the electronic wedge brake (EWB), and two variants of the cruise

control (CC1) and (CC2). The plants used are described as follows:
(i) The DC motor speed control (DCM) is adapted from [21], where the state variables $x = [x_1\ x_2]^T$ represent the rotational speed of the motor shaft and the armature current. The control input u is the motor terminal voltage. This control model can for example be applied to the wheel speed control in an electric vehicle. The system matrices for this plant are represented as

$$A = \begin{bmatrix} -10 & 1 \\ -0.02 & -2 \end{bmatrix},\ B = \begin{bmatrix} 0 \\ 2 \end{bmatrix},\ C = [1 \quad 0]. \tag{35}$$

(ii) The car suspension system (CSS), adapted from [5] has the state variables $x = [x_1\ x_2\ x_3\ x_4]$, where x_1 and x_2 represent the position and velocity of the car and x_3 and x_4 are the position and velocity of the mass of the suspension system. The control input u is the force applied to the body by the suspension system. The system matrices can be represented as

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -8 & -4 & 8 & 4 \\ 0 & 0 & 0 & 1 \\ 80 & 40 & -160 & -60 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 80 \\ 20 \\ -1120 \end{bmatrix}, C = [1 \quad 0 \quad 0 \quad 0]. \tag{36}$$

(iii) The electronic wedge brake (EWB) is a simplified version adapted from the self-reinforced brake-by-wire solution developed by Siemens [22]. Modeling of the wedge results in a second-order system. Two state variables $x = [x_1\ x_2]$ are the position and the velocity of the braking wedge, respectively. While the DC motor model is simplified, the control input u is the force provided by the motor. The plant model are represented as

$$A = \begin{bmatrix} 0 & 1 \\ 8.3951 \times 10^3 & 0 \end{bmatrix},\ B = \begin{bmatrix} 0 \\ 4.0451 \end{bmatrix},\ C = \begin{bmatrix} 7.9920 \times 10^3 & 0 \end{bmatrix}. \tag{37}$$

(iv) The variant 1 of the cruise control (CC1) is a simplified version of the cruise control system from [21], neglecting the dynamics of the powertrain, tires, etc. Here, the state variable x represents the speed of a vehicle and the control input u is the force exerted on the vehicle. The plant can be represented as

$$A = -0.05,\ B = 0.001,\ C = 1. \tag{38}$$

(v) The second variant of the cruise control (CC2) is a more detailed version of the cruise control system, which is used in [6] and derived from [24]. The cruise control system regulates the vehicle speed to follow the driver's command. The state space representation of this system can be described as [24]

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6.0476 & -5.2856 & -0.238 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 2.4767 \end{bmatrix}, C = [1 \quad 0 \quad 0]. \tag{39}$$

Architecture: The hardware platform used in this case study, as shown in Fig.4, consists of three ECUs connected by FlexRay in a bus topology. Table I shows the task mapping on the ECUs and Table II shows two sets of bus parameters considered. Set 1 is obtained from a related work [5]. Set 2 is adapted from an industrial application [23]. In the original parameters of [23], the length of a static slot is not able to accommodate the largest frame in our case study, the payload of which consists of 4 float sensor data of 4 bytes each. Therefore the static slot length is adjusted to $39.875\mu s$ to accommodate this frame. The bus topology is a common one in automotive domain, which can be found in related works [5], [6], [9]. The number of ECUs, the number of control applications and task mapping are chosen for ease of demonstration. The applicability of the approach, however, is not limited to this setup. In fact, as will be explained in the scalability analysis in V-B, the approach can be applied to a number of different setups with varying system sizes and task mappings.

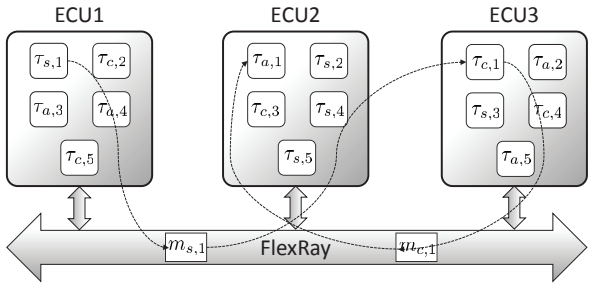

Fig. 4: The platform architecture for the case study.

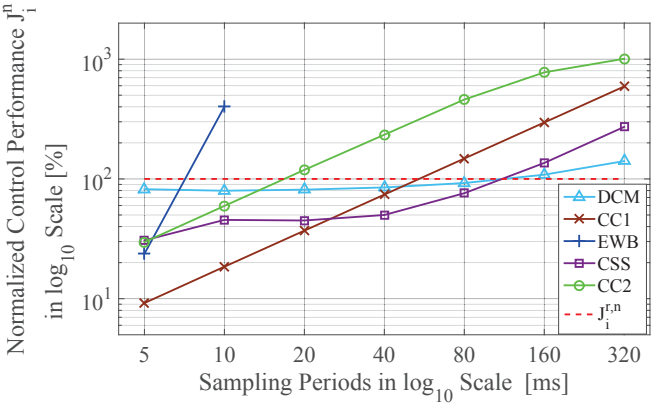

Fig. 5: Control performance.

ECUs	Tasks				
E_1	$\tau_{s,1}$,	$\tau_{c,2}$,	$\tau_{a,3}$,	$\tau_{a,4}$,	$\tau_{c,5}$
E_2	$\tau_{a,1}$,	$\tau_{s,2}$,	$\tau_{c,3}$,	$\tau_{s,4}$,	$\tau_{s,5}$
E_3	$\tau_{c,1}$,	$\tau_{a,2}$,	$\tau_{s,3}$,	$\tau_{c,4}$,	$\tau_{a,5}$

TABLE I: Task mapping.

Bus Parameters	Values	
	Set 1	Set 2
Bus Speed	10 $Mbps$	10 $Mbps$
T_{bus}	5 ms	5 ms
MacroTick	1 μs	1.375 μs
N	25	75
M	237	267
Δ	100 μs	39.875 μs
δ	10 μs	6.875 μs

TABLE II: FlexRay Bus Configuration.

Controller Design: In the controller design phase, a search of poles is carried out at each possible sampling period for each control application. Fig. 5 shows the results of the optimal control performance for each control application as the sampling period increases. Except for EWB, where controllers stabilizing the plant can only be found for sampling period 5ms and 10ms, controllers can be designed for all control applications at sampling periods 5ms, 10ms, 20ms, 40ms, 80ms, 160ms and 320ms. This is because the bus communication cycle is 5ms, and the sampling periods

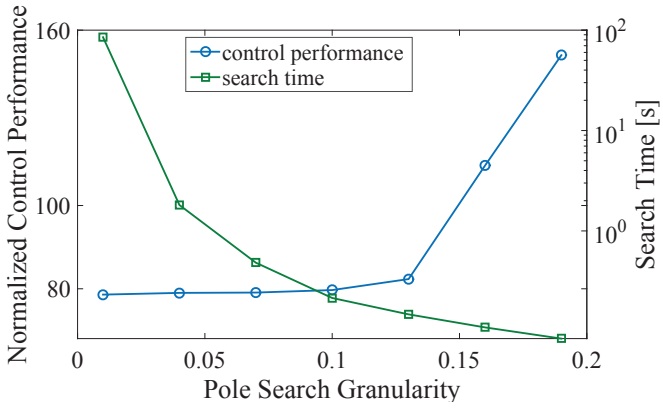

Fig. 6: Granularity of pole search for DCM.

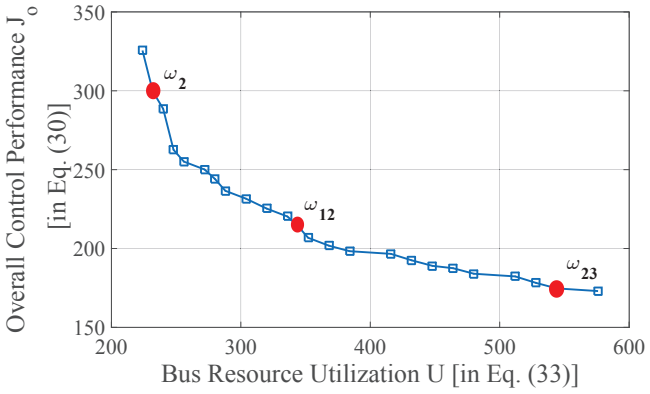

Fig. 7: Pareto front.

correspond to the discrete values of the repetition rate. The thick red dashed line in the plot shows the normalized required performance for all the control applications (i.e., 100%). Only the points below the red line meets the design requirement for performance and only these points will be considered in the following co-optimization stage. Fig. 6 illustrates the influence of the granularity of the pole search. As can be observed from the figure, the finer the granularity, the more time it requires to search the pole space, and the controllers with better control performance can be found. It should be noted that the order of the control system model also influences the search time. With the same granularity, the higher the order of the system, the longer it would take to search the pole space. The plant model (DCM) of the control application shown in Fig. 6 is a second-order system.

Co-Optimization: For the co-optimization stage, we use two different sets of bus parameters as shown in Table II. We first consider the bus parameter Set 1. The Pareto front of the whole system in the case study obtained in the co-optimization stage is shown in Fig. 7. It can be observed from the figure, that the proposed approach obtains 24 reasonably well-distributed Pareto points. Each point represents a design option that offers a specific choice of trade-off between bus resource utilization and overall control performance. The user can then choose one design option according to his own preference. The value of the bus resource utilization – number of static slots used in 64 communication cycle – ranges from 224 to 576, which correspond to 14% and 36% of the bus bandwidth in the static segment. The value of the overall control performance varies from 173 to 325.6, corresponding an average control performance of 34.6% and 65.12% of the required value for each control application. It should be noted that for the control performance defined in this paper, the smaller the value, the better the performance. It is obvious that there is a large freedom among these viable designs. If the resource efficiency is the top design priority, the engineer can only use 14% of the bus bandwidth in the static segment to achieve stable control. If a performance optimal design is desired, a much better performance (average gain of 30% in each application) can be achieved at the cost of an extra 22% of the bandwidth. Therefore the Pareto points delivered can be explored to obtain a design choice most suitable for the requirement. Note that for a relatively small system size, there is already such a considerable design freedom available. For larger systems, an engineer could possibly profit even more from the trade-off choices between the two objectives. In the case of bus parameter Set 2, the Pareto front is exactly the same as for Set 1, although each Pareto point corresponds to a different set of schedules for the tasks and the FlexRay messages. The difference here is that the number of static slots in one communication cycle is 75 instead of 25. Therefore, the number of static slots used in all design options in the Pareto front ranges from 224 to 576, which correspond to 4.7% to 12% of the bus bandwidth in the static segment in this case. In terms of control performance, the values are the same as for Set 1. It can also be observed that not for all discrete values of U a Pareto point exists. The reason for this could be that either (i) for such a resource utilization value, a feasible parameter set can not be synthesized, or (ii) the optimal solution for this value is dominated by other points and therefore no longer considered as a Pareto point. For the case study, the co-optimization stage for Set 1 took 7.83s on a Laptop with an Intel Core i7-5600U CPU of 2.60GHz and 8GB RAM. The approach is implemented primarily in Matlab. We use the Gurobi [19] solver for the inner optimization layer and CPLEX [20] solver for the outer layer. This is because in the outer layer we need all optimal solutions and CPLEX provides a feature *solution pool* for that.

Simulation Validation: In order to validate the results, we have used SIMTOOLS [27], a commercial industrial design toolbox integrated with MATLAB/SIMULINK, which supports model-based virtual prototyping of FlexRay-based distributed embedded systems. We have developed the FlexRay-based software model for the system under consideration. In addition, we have also developed simulation models for each of the control plants. For each control application, the sensor task model receives the sensor values from the corresponding plant model and the actuator task model applies the control input to the corresponding plant model. The complete model, comprising the software model and the plant models, is configured successively with parameters corresponding to 3 well spaced Pareto points ω_2, ω_{12} and ω_{23} as marked in Fig. 7, and consequently tested for plausibility. The models corresponding to the 3 Pareto points are successively simulated

according to 4^{th} level of simulation available in SIMTOOLS, where the ECUs and the communication system are simulated based on the timings of application tasks, communication tasks and bus schedules. The control responses are recorded for each of the simulation runs and are shown in Fig. 8. Here, the control systems C_1, C_3, C_4 and C_5 are applied step references while the system C_2 is applied an impulse reference. It may be observed in the figure, that the control response of the car suspension system change appreciably for Pareto point ω_{23} from the responses for Pareto points ω_2 and ω_{12}. This verifies our assumption in the co-optimization stage that the control performance depends mainly on the sampling period of the control application. For CSS, the sampling period is $40ms$ in case of Pareto points ω_2 and ω_{12} and $5ms$ in case of Pareto point ω_{23}. Therefore, the control responses of CSS corresponding to Pareto points ω_2 and ω_{12} are similar despite different task and message schedules. Furthermore, we can observe that the settling times of the control responses of the first variant of cruise control system are respectively 54 ms, 106 ms and 415 ms corresponding to sampling periods of 5 ms, 10 ms and 40 ms and are approximately in the ratio 1:2:8 which can be verified from the control performance curve shown in Fig. 5. Similarly, control performances of all applications can be verified against the calculated values from the simulation results.

B. Scalability Analysis

In order to evaluate the scalability of the proposed approach, we have conducted an analysis on a set of synthetic test cases of different system sizes. The system sizes we consider range from 6 to 24 control applications. When the number of the control applications increases, we also increase the number of ECUs. Furthermore, we define η as the ratio between the number of control applications and the number of ECUs in the whole system and η reflects the average ECU load. We consider three different cases each with $\eta = 1$, $\eta = 1.5$ and $\eta = 2$ respectively. For each system size and average ECU load value, 10 test cases are randomly generated (including task mapping on ECUs and WCET of tasks). Then 4 control plant models, namely DCM, CSS, CC1 and CC2, are replicated and mapped onto the control applications. The control performance curves of these plants are shown in Fig 5. For this analysis, we use a similar FlexRay bus configuration as the Set 1 shown in Table II. The only difference is that the dynamic segment is turned into extra 23 static slots.

The results are shown in Table III. From the results, we can conclude that the approach can scale to a system size of 24 applications. For such a system size, there are 72 tasks and 48 messages in the system. Comparing to two real-world automotive benchmark subsystems described in [25], which contains 9 ECUs, 44 tasks, 19 messages, and [26], which contains 15 ECUs and 53 messages (only part of the task descriptions are provided), we can assume that a system size of 24 applications in our case resembles a reasonable system size of a bus cluster in the automotive domain. In addition, in [25], the task to ECU ratio is 4.9 and the message to ECU is 2.1 and in [26], the message to ECU ratio is 3.5. Therefore, we consider the value of η up to 2, which results in the task to ECU ratio of 6 and the message to ECU ratio of 4. For $\eta = 1$ and $\eta = 1.5$, the average synthesis times for the co-optimization in the case of 24 control applications are 181.19s and 189.32s respectively. For $\eta = 2$, the average synthesis time is 1219.03s (approx. 20min), which is quite reasonable for an offline approach, considering also that parameters are synthesized for 149 Pareto points in total. Our analysis shows that in some test cases, one or more of the Pareto points correspond to already 85% bus utilization and 55% average ECU utilization. Note that for two out of ten test cases (with 24 applications and $\eta = 2$) an optimal Pareto front cannot be synthesized within 1.5 hours. The analysis of these two test cases shows that for some of the Pareto point candidates, one or more of the ECUs are more than 90% loaded. Therefore,

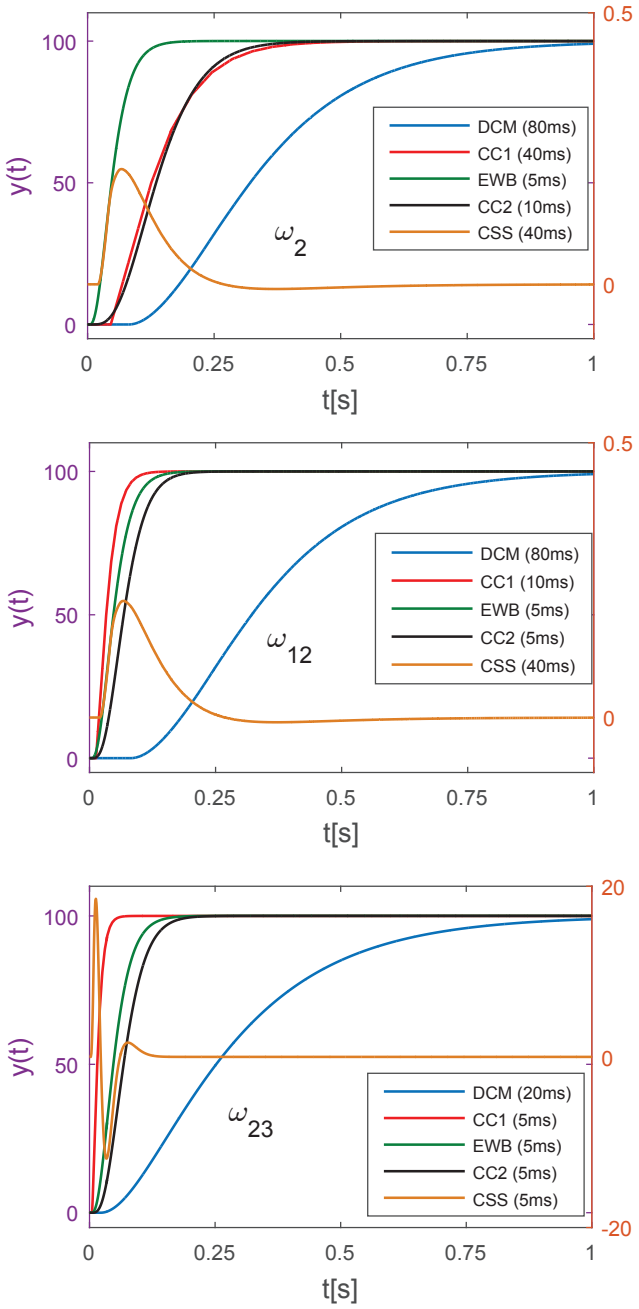

Fig. 8: Control responses corresponding to the pareto points ω_2, ω_{12} and ω_{23} (Sampling period values are given in brackets).

it is more difficult to synthesize a feasible schedule or render it infeasible in the inner optimization layer (findSchedules in Algorithm 3).

	No. Control Applications			
	6	12	18	24
$\eta = 1$	26.36	55.23	88.86	181.19
$\eta = 1.5$	27.12	54.48	94.85	189.32
$\eta = 2$	28.14	55.55	100.09	1219.03[1]

TABLE III: Scalability analysis: average synthesis time in seconds for the co-optimization stage for different system sizes and different values of η.

VI. Concluding Remarks

In this paper, we have proposed a co-optimization approach for the design of FlexRay-based distributed control systems. A customized control design and a nested two-layered optimization technique are introduced to reduce the complexity of the problem. The approach generates a Pareto front to offer design trade-offs between the objective of overall control performance and bus resource utilization. In the future work, we would first like to consider a more sophisticated control design methods considering the variable sensor-to-actuator delays, while not making too much compromise on the scalability of the approach. Secondly, we have considered in this paper only the co-optimization problem with a single FlexRay bus cluster. Future work may extend this to multiple FlexRay bus clusters, even heterogeneous ones. The challenges here include firstly the scalability to even larger systems sizes and possible trade-offs between more optimal designs and less computational effort. In addition, to deal with less deterministic bus systems, extensions like timing analysis could be needed. Thirdly, we want to ease the selection procedure of the designer by designing a filtering algorithm for the Pareto points according to specific requirement. In general, there can be a very large number of Pareto points (exponential in problem size) [28], [29]. There have been previous works on approximating the Pareto front with polynomial number of points [28], [29]. In the context of this work, this could be useful to provide the designer a reasonable number of Pareto points in good distribution to help him to conveniently make design choices.

Acknowledgment

This work was partially funded by (i) the project ARTEMIS 621429 EMC2 and (ii) the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) program.

References

[1] P. Marti, R. Villa, J.M. Fuertes, and G. Fohler. On real-time control tasks schedulability. European Control Conference (ECC), 2001, 2227-2232.

[2] L. Ma, F. Xia, Z. Peng. Integrated Design and Implementation of Embedded Control Systems with Scilab. Sensors, 2008, 5501-5515.

[3] M. 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, Z. Peng. Integrated scheduling and synthesis of control applications on distributed embedded systems. Design Automation and Test in Europe (DATE), 2009, 57-62.

[1]For 2 out of the 10 test cases an optimal Pareto front cannot be obtained within 1.5 hours.

[5] R. Schneider, D. Goswami, S. Zafar, M. Lukasiewycz, and S. Chakraborty. Constraint-driven synthesis and tool-support for FlexRay-based automotive control systems. International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011, 139-148.

[6] D. Goswami, R. Schneider, and S. Chakraborty. Relaxing signal delay constraints in distributed embedded controllers. IEEE Transaction on Control Systems Technology, 2014, 22(6), 2337-2345.

[7] M. Lukasiewycz, M. Glaß, P. Milbredt, and J. Teich. FlexRay schedule optimization of the static segment. International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2009, 363-372.

[8] W. Zheng, J. Chong, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Extensible and scalable time triggered scheduling. International Confierence on Application of Concurrency to System Design (ACSD), 2005, 132-141.

[9] D. Goswami, M. Lukasiewycz, R. Schneider, and S. Chakraborty. Time-triggered implementations of mixed-criticality automtoive software. Design, Automation, and Test in Europe (DATE), 2012, 1227-1232.

[10] J. Mao, Z. Wu, and X. Wu. A TDMA scheduling scheme for many-to-one communications in wireless sensor networks. Computer Communications, Vol. 30, Iss. 4, pp. 863-872, 2007.

[11] M. Elmusrati, H. Sallabi, and H. Koivo. Applications of multi-objective opitmization techniques in radio resource scheduling of cellular communication systems. IEEE Wireless Communications, Vol. 7, Iss. 1, pp. 343-353, 2008.

[12] M. Glaß, J. Teich, and L. Zhang. A co-simulation approach for system-level analysis of embedded control systems. International Conference on Embedded Conputer Systems: Architectures, Modeling and Simulation (SAMOS), 2012, 355-362.

[13] Q. Zhu, and P. Deng. Design synthsis and optimazation for automotive embedded systems. International Symposium on Physical Design (ISPD), 2014, 141-148.

[14] A. Aminifar, S. Samii, P. Eles, Z. Peng, and A. Cervin. Designing high-quality embedded control systems with guaranteed stability. IEEE Real-Time Systems Symposium (RTSS), 2012, 283-292.

[15] A. Aminifar, P. Eles, Z. Peng, and A. Cervin. Control-quality driven design of cyber-physical systems with robustness guarantees. Design, Automation, and Test in Europe (DATE), 2013, 1093-1098.

[16] K. J. Astrom, B. Wittenmark. Computer-controlled systems. Theory and design. 3rd Edition. Prentice Hall. 1997.

[17] W. Chang, M. Lukasiewycz, S. Steinhorst, and S. Chakraborty, Dimensioning and configuration of EES systems for electric vehicles with boundary-conditioned adaptive scalarization, International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013.

[18] The FlexRay communications system protocol specification, Version 3.0.1. www.flexray.com.

[19] Gurobi Optimization, Inc.. Gurobi Optimizer Reference Manual, 2015. http://www.gurobi.com.

[20] IBM ILOG CPLEX Optimization Studio. http://www-03.ibm.com/software/products/de/ibmilogcpleoptistud.

[21] W. C. Messner, and D. M. Tilbury. Control tutorials for MATLAB and Simulink: a web-based approach. Menlo Park, CA: Addison-Wesley, 1998. http://ctms.engin.umich.edu/CTMS.

[22] J. Fox, R. Roberts, C. Baier, L. Ho, L. Lacraru and B. Gombert, Modeling and control of a single motor electronic wedge brake, SAE Technical Paper, (2007-01-0866), 2007.

[23] A. Schedl. Goals and architectures of FlexRay at BMW. In slides presented at the Vector FlexRay Symposium, 2007.

[24] K. Osman, M. Rahmat, and M. Ahmad. Modelling and controller design for a cruise control system. International Colloquium on Signal Processing and Its Applications (CSPA), 2009.

[25] P. Castelpietra, Y. Song, F. Simonot-Lion, and M. Attia. Analysis and simulation methods for performance evaluation of a multiple networked embedded architecture. IEEE Transactions on Industrial Electronics, Vol. 49, No. 6, 2002.

[26] S. Kollman, V. Pollex, K. Kempf, F. Slomka, M. Traub, T. Bone, and J. Becker. Comparative application of real-time verification methods to an automotive architecture. 18th International Conference on real-time and network systems, 2010, 89-98.

[27] SIMTOOLS. Model-Based Design Tools for FlexRay-based Applications. http://www.simtools.at/.

[28] C. Papadimitriou, and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources, 41st Annual Symposium on Foundations of Computer Science (FOCS), 2000, 86-92

[29] C. Papadimitriou, and M. Yannakakis. Multiobjective query optimization. Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), 2001, 52-59.