Control-Quality Driven Design of Cyber-Physical Systems with Robustness Guarantees

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

***Abstract*—Many cyber-physical systems comprise several control applications sharing communication and computation resources. The design of such systems requires special attention due to the complex timing behavior that can lead to poor control quality or even instability. The two main requirements of control applications are: (1) robustness and, in particular, stability and (2) high control quality. Although it is essential to guarantee stability and provide a certain degree of robustness even in the worst-case scenario, a design procedure which merely takes the worst-case scenario into consideration can lead to a poor expected (average-case) control quality, since the design is solely tuned to a scenario that occurs very rarely. On the other hand, considering only the expected quality of control does not necessarily provide robustness and stability in the worst-case. Therefore, both the robustness and the expected control quality should be taken into account in the design process. This paper presents an efficient and integrated approach for designing high-quality cyber-physical systems with robustness guarantees.**

I. Introduction and Related Work

Cyber-physical systems are often implemented on distributed platforms consisting of several computation and communication resources. Many such systems comprise several control applications sharing the available resources. Such resource sharing, if not properly taken into account during the design process, can lead to poor control quality. It is well-known that traditional approaches based on the principle of *separation of concerns* can lead to either resource under-utilization or poor control performance and, in the worst-case, may even lead to instability of control applications [1], [2]. Therefore, in order to achieve high control performance while guaranteeing stability even in the worst case, it is essential to consider the timing behavior extracted from the system schedule during control synthesis, and to consider both control performance and stability during system scheduling. The issue of control–scheduling co-design [2] has become a notable research direction in recent years, and there have been several solution attempts aimed at this problem [3–13]. In spite of the fact that considering both control performance and stability is of great importance, previous work only focuses on one of these two aspects.[1] A design approach which only takes the average control quality into consideration, does not necessarily guarantee the stability of control applications in the worst case. On the other hand, considering merely the worst-case scenario often results in a system with poor expected control performance. This is due to the fact that the design is tuned to a specific scenario that occurs very rarely.

In our previous work [14], we have proposed an efficient solution to the above problem, which, however, is based on a set of properties that only hold in the uniprocessor case. A completely different approach is needed in the case of distributed platforms, which are the typical infrastructure for cyber-physical systems. In this paper, we propose an integrated control–scheduling approach to design high-quality cyber-physical systems with guarantees on the worst-case performance. To this end, two kinds of metrics are considered: (1) robustness (stability-related) metrics and (2) stochastic control performance metrics. The former are considered to be measures of the worst-case control performance, whereas the latter capture the expected (as in *mathematical expectation*) performance of an application. Even though the overall control performance of a system is determined by the expected control performance, taking the stability requirements into consideration during design space exploration is of huge importance [15]. Therefore, the optimization is performed considering the expected control performance as the objective function, subject to the robustness requirements.

II. System Model

A. Plant Model

Let us consider a given set of plants $\mathbf{P}$. Each plant P_i is modeled by a continuous-time system of equations [16]

$$\begin{aligned} \dot{\boldsymbol{x}}_i &= A_i \boldsymbol{x}_i + B_i \boldsymbol{u}_i + \boldsymbol{v}_i, \\ \boldsymbol{y}_i &= C_i \boldsymbol{x}_i + \boldsymbol{e}_i, \end{aligned} \tag{1}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are the plant state and control signal, respectively. The control signal is updated at some point in each sampling period and is held constant between updates. The additive plant disturbance $\boldsymbol{v}_i$ is a continuous-time white-noise process with zero mean and given covariance matrix. The plant output, denoted by $\boldsymbol{y}_i$, is sampled at some point in each sampling period—the measurement noise $\boldsymbol{e}_i$ is discrete-time Gaussian white-noise with zero mean and given covariance.

B. Platform and Application Model

We consider a distributed execution platform that consists of several computation nodes, $N_i \in \mathbf{N}$, connected by communication controllers to a bus.

For each plant $P_i \in \mathbf{P}$ there exists a corresponding control application denoted by $\Lambda_i \in \boldsymbol{\Lambda}$, where $\boldsymbol{\Lambda}$ indicates the set of control applications. Each control application Λ_i is modeled as a task chain. A task chain consists of a set of tasks and a set of edges, identifying the dependencies among tasks. Thus, an application is modeled as a graph $\Lambda_i = (\mathbf{T}_i, \boldsymbol{\Gamma}_i)$, where $\mathbf{T}_i$ denotes the set of tasks and $\boldsymbol{\Gamma}_i = \left\{(\tau_{ij}, \tau_{i(j+1)}) \middle| \tau_{ij}, \tau_{i(j+1)} \in \mathbf{T}_i\right\}$ denotes the dependencies between tasks. We denote the j^{th} task of the task chain of application Λ_i by τ_{ij}. The message between tasks τ_{ij} and $\tau_{i(j+1)}$ is indicated by the ordered pair $\gamma_{ij(j+1)} = (\tau_{ij}, \tau_{i(j+1)}) \in \boldsymbol{\Gamma}_i$. The execution-time, c_{ij}, of task τ_{ij} is modeled as a stochastic variable with probability function

[1] Although [4], [12] can guarantee stability besides control performance optimization, their approaches are restricted to static-cyclic and time-triggered scheduling.

978-3-9815370-0-0/DATE13/©2013 EDAA

ξ_{ij},[2] bounded by the best-case execution-time c_{ij}^{b} and the worst-case execution-time c_{ij}^{w}. The message transmission time between tasks τ_{ij} and $\tau_{i(j+1)}$ is constant and is denoted by $c_{ij(j+1)}$. Further, we consider the mapping of tasks given by a mapping function $\mathsf{map} : \bigcup_{\Lambda_i \in \mathbf{\Lambda}} \mathbf{T}_i \rightarrow \mathbf{N}$. The communication $\gamma_{ij(j+1)}$ is done on the bus if tasks τ_{ij} and $\tau_{i(j+1)}$ are mapped on different nodes (i.e., $\mathsf{map}(\tau_{ij}) \neq \mathsf{map}(\tau_{i(j+1)})$); otherwise, the communication is done locally and the overhead is considered in the computation times of tasks τ_{ij} and $\tau_{i(j+1)}$.

Control applications typically provide a satisfactory performance within a range of sampling periods [16]. Hence, each application Λ_i can execute with a period $h_i \in \mathbf{H}_i$, where $\mathbf{H}_i$ is the set of suggested periods that application Λ_i can be executed with. However, the actual periods are determined during the co-design procedure, considering the relation between scheduling parameters and the controller synthesis.

III. Control Performance and Synthesis

A. Expected Control Performance

In order to capture the expected performance of an application Λ_i, we use a standard quadratic cost function [16]

$$J_{\Lambda_i}^{\mathrm{e}} = \lim_{T\to\infty} \frac{1}{T} \mathbb{E} \left\{ \int_0^T \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix}^{\mathsf{T}} Q_i \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix} dt \right\}. \qquad (2)$$

Here, $\mathbb{E}\{\cdot\}$ denotes expected value, and the positive semi-definite weight matrix Q_i is given by the designer. To compute the expected cost $J_{\Lambda_i}^{\mathrm{e}}$ for a given delay distribution, the Jitterbug toolbox is employed [17].

While appropriate as a metric for the average quality of control, the above cost function cannot provide a *hard guarantee* of stability in the worst case. Using Jitterbug, the stability of a plant can be analyzed in the mean-square sense *if* all time-varying delays are assumed to be independent stochastic variables. However, by their nature, task and message delays do not behave as independent stochastic variables and therefore, stability results based on the above quadratic cost are not valid as worst-case guarantees.

B. Worst-Case Control Performance

We quantify the worst-case control performance of a system by computing an upper bound on the worst-case gain G from the plant disturbance d to the plant output y. The plant output is then guaranteed to be bounded by

$$\|y\| \leq G\|d\|.$$

If $G = \infty$, then stability of the system cannot be guaranteed. A smaller value of G implies a higher degree of robustness.

To numerically compute the worst-case gain G, we use the Jitter Margin toolbox [18]. As inputs, the toolbox takes the plant model P_i, the control application Λ_i with associated sampling period, the nominal sensor–actuator (input–output) delay L_i, the worst-case sensor (input) jitter Δ_{is}^{w}, and the worst-case actuator (output) jitter Δ_{ia}^{w}. The worst-case performance is hence captured by a cost function

$$J_{\Lambda_i}^{\mathrm{w}} = G(P_i, \Lambda_i, L_i, \Delta_{is}^{\mathrm{w}}, \Delta_{ia}^{\mathrm{w}}). \qquad (3)$$

The worst-case sensor and actuator jitters are computed using response-time analysis (see Section IV, Figure 1).

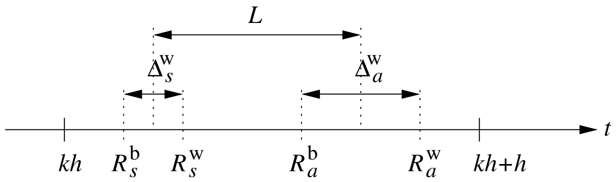

Fig. 1. Graphical interpretation of the nominal sensor–actuator delay L, worst-case sensor jitter Δ_s^{w}, and worst-case actuator jitter Δ_a^{w}

C. Control Synthesis

For a given sampling period h_i and a given, constant sensor–actuator delay (i.e., the time between sampling the output $\boldsymbol{y}_i$ and updating the controlled input $\boldsymbol{u}_i$), it is possible to find the control-law $\boldsymbol{u}_i$ that minimizes the cost $J_{\Lambda_i}^{\mathrm{e}}$ [16]. Since the overall performance of the system is determined by the expected control performance, the controllers are designed for the expected (average) behavior of the system. The sensor–actuator delay distribution is strongly dependent on scheduling parameters and the quality of the constructed controller is degraded if the actual sensor–actuator delay distribution is different from the one assumed during the control-law synthesis. This clearly motivates the need for a control–scheduling co-design approach. Given the scheduling parameters, system simulation can be performed to obtain the delay distribution and the expected sensor–actuator delay. The Linear-Quadratic-Gaussian (LQG) controller, which is *optimal with respect to the expected control performance* (Equation 2), is then synthesized *to compensate for the expected sensor–actuator delay* using MATLAB and the Jitterbug toolbox [17]. In reality, however, the sensor–actuator delay is not *constant and equal* to this expected value, due to the interference by other applications competing for the shared resources which may degrade the quality of the synthesized controller. The overall expected control quality of the controller for a given delay distribution is obtained according to Section III-A.

IV. Jitter and Delay Analyses

In order to apply the worst-case control performance analysis, we shall compute the three parameters mentioned in Section III-B (Equation 3), namely, the nominal sensor–actuator delay L_i, worst-case sensor jitter Δ_{is}^{w}, and worst-case actuator jitter Δ_{ia}^{w} for each control application Λ_i. Figure 1 illustrates the graphical interpretation of these three parameters. These parameters can be obtained by applying response-time analysis as follows,

$$\begin{aligned} \Delta_{is}^{\mathrm{w}} &= R_{is}^{\mathrm{w}} - R_{is}^{\mathrm{b}}, \\ \Delta_{ia}^{\mathrm{w}} &= R_{ia}^{\mathrm{w}} - R_{ia}^{\mathrm{b}}, \\ L_i &= \left(R_{ia}^{\mathrm{b}} + \frac{\Delta_{ia}^{\mathrm{w}}}{2} \right) - \left(R_{is}^{\mathrm{b}} + \frac{\Delta_{is}^{\mathrm{w}}}{2} \right), \end{aligned} \qquad (4)$$

where R_{is}^{w} and R_{is}^{b} denote the worst-case and best-case response times for the sensor task τ_{is} of the application Λ_i, respectively. Analogously, R_{ia}^{w} and R_{ia}^{b} are the worst-case and best-case response times for the actuator task τ_{ia}.

We consider that tasks are executed based on a preemptive fixed-priority policy. Furthermore, we consider the computation nodes are connected via a CAN (Controller Area Network) bus (non-preemptive fixed-priority arbitration policy). Computation of the end-to-end worst-case and best-case response times is done using the holistic response-time analysis [19], [20] based on the BWCRT algorithm in [21]. The BWCRT algorithm calculates the best-case and worst-case response times iteratively and updates the static and dynamic offsets until convergence. For more general system models, e.g., heterogeneous architectures,

[2] Note that the worst-case stability guarantees only depend on the worst-case and best-case execution times. The probability function ξ_{ij} is only needed for system simulation which is used in computing the expected sensor–actuator delay and the expected control performance.

and complex task dependencies, the SymTA/S [22] tool can be used.

The worst-case response time of task τ_{ij} can be computed using the offset-based analysis [23] as follows,

$$R^{\mathrm{w}}_{ij} = \max_{\forall \tau_{ik} \in hp(\tau_{ij}) \cup \{\tau_{ij}\}} \left\{ \max_{\forall p} \left\{ \mathrm{w}_{ijk}(p) - \varphi_{ijk} - (p-1)h_i + \Phi_{ij} \right\} \right\},$$

where $hp(\tau_{ij})$ is the set of higher priority tasks which are mapped on the same computation node and $\mathrm{w}_{ijk}(p)$ is the worst-case busy period of the p^{th} job of τ_{ij} in the busy period, numbered from the critical instant initiated by τ_{ik}. The value of $\mathrm{w}_{ijk}(p)$ is determined as follows,

$$\mathrm{w}_{ijk}(p) = B_{ij} + (p - p_{0,ijk} + 1)c^{\mathrm{w}}_{ij} + W_{ik}(\tau_{ij}, \mathrm{w}_{ijk}(p)) + \sum_{\forall a \neq i} W^*_a(\tau_{ij}, \mathrm{w}_{ijk}(p)),$$

where $W_{ik}(\tau_{ij}, t)$ and $W^*_a(\tau_{ij}, t)$ are the worst-case interference by the higher priority tasks in the same task chain and the maximum of all possible interferences that could be caused by task chain Λ_a, on τ_{ij}, for a busy period of duration t, respectively. B_{ij} is the maximum interval during which τ_{ij} can be blocked by the lower priority tasks.

The CAN network is modeled as a unit arbitrated according to a non-preemptive priority driven policy and the delays induced by message passing are considered inside the above analysis [24], [23], [20].

Under fixed-priority scheduling, the best-case response time of task τ_{ij} is given by the following equation [21]

$$\mathrm{w}_{ij} = c^{\mathrm{b}}_{ij} + \sum_{\forall \tau_{ab} \in hp(\tau_{ij})} \left\lceil \frac{\mathrm{w}_{ij} - (h_a + R^{\mathrm{w}}_{pre(ab)} - R^{\mathrm{b}}_{ab})}{h_a} \right\rceil_0 c^{\mathrm{b}}_{ab},$$

$$R^{\mathrm{b}}_{ij} = \mathrm{w}_{ij} + R^{\mathrm{b}}_{pre(ij)},$$

where $R^{\mathrm{w}}_{pre(ab)}$ captures the worst-case response time of the direct predecessor of task τ_{ab} in the task chain.

For the best-case response time for message $\gamma_{ij(j+1)}$ on a CAN bus, we consider the following,

$$R^{\mathrm{b}}_{ij(j+1)} = c_{ij(j+1)} + R^{\mathrm{b}}_{ij}.$$

V. Motivational Example

We consider three plants $\mathbf{P} = \{P_1, P_2, P_3\}$. For each plant P_i, a discrete-time LQG controller Λ_i is synthesized for a given period and constant expected sensor–actuator delay using the Jitterbug toolbox [17] and MATLAB. The expected sensor–actuator delay is obtained using our system simulation environment for distributed real-time systems. Each controller Λ_i is modeled as a task chain consisting of two tasks, sensor task τ_{is}, and computation and actuator task τ_{ica}. The task chains and the mapping of the tasks on the distributed platform (two processing nodes $\mathbf{N} = \{N_1, N_2\}$ connected via a bus) are depicted in Figure 2. The numbers in parentheses are execution times of tasks or communication times of messages. All time quantities are given in milliseconds throughout this section. The total expected control cost, for a set of plants $\mathbf{P}$, is $J^{\mathrm{e}}_{\mathrm{total}} = \sum_{P_i \in \mathbf{P}} J^{\mathrm{e}}_{\Lambda_i}$, whereas the total worst-case control cost is defined to be $J^{\mathrm{w}}_{\mathrm{total}} = \sum_{P_i \in \mathbf{P}} J^{\mathrm{w}}_{\Lambda_i}$. Further, we consider the fixed-priority scheduling policy and assume that application Λ_i has higher priority than application Λ_j iff $i > j$. Having assigned the priorities,[3] a design solution is captured by a tuple $\mathrm{DS}_i = (h_1, h_2, h_3)$ which identifies the assigned periods of the applications. [4] It should be noted that for this particular example, the values of the nominal sensor–actuator delay, worst-case sensor jitter, and worst-case actuator jitter are the same for all explored design solutions.

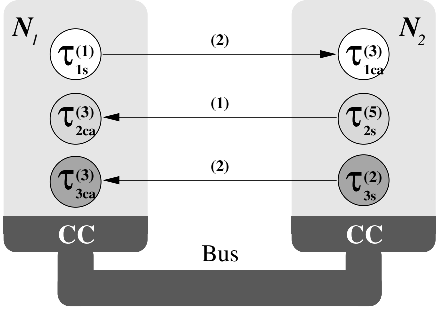

Fig. 2. Motivational example

Let us consider $\mathrm{DS}_1 = (50, 40, 50)$ to be the initial period assignment. The total expected control cost, calculated by the Jitterbug toolbox, for this design solution is equal to $J^{\mathrm{e}}_{\mathrm{total}} = 11.5$. However, using the Jitter Margin toolbox, we realize that the stability of control application Λ_1 cannot be guaranteed ($J^{\mathrm{w}}_{\Lambda_1} = \infty$).

In order to decrease the interference by higher priority applications for application Λ_1, the designer might increase the period of application Λ_3 to 60. In addition, since smaller period often leads to a better control performance, the designer might decide to decrease the period of application Λ_1 to 40, leading to the design solution $\mathrm{DS}_2 = (40, 40, 60)$. The total worst-case control cost is $J^{\mathrm{w}}_{\mathrm{total}} = 31.7$ which, since finite, represents a guarantee of stability for all applications. The total expected control cost, however, is increased to $J^{\mathrm{e}}_{\mathrm{total}} = 26.0$.

Another solution would be $\mathrm{DS}_3 = (40, 40, 50)$. This leads to the expected and worst-case control costs $J^{\mathrm{e}}_{\mathrm{total}} = 11.9$ and $J^{\mathrm{w}}_{\mathrm{total}} = 75.1$. Both DS_2 and DS_3 guarantee the stability of all applications since the worst-case control costs are finite. However, although the former solution (DS_2) provides better worst-case control performance, the latter (DS_3) is desirable since the total expected performance is better. It is also easy to observe that with DS_3 it has been possible to guarantee stability with a very small deterioration of the expected control cost, compared to DS_1.

It should be noted that although control application Λ_1 is assigned a smaller period in DS_3, the total expected control performance of DS_1 is slightly better. While this change leads to a better expected control performance for control application Λ_1 in DS_3, the expected control performance of the high priority control application Λ_3 is worse in DS_3. This is due to the non-preemptability of the communication infrastructure, which, in turn, has led to variation in the delay (i.e., delay distribution) experienced by control application Λ_3.

We conclude that, optimizing the expected control quality without taking the worst-case control performance into account can lead to design solutions which are unsafe in the worst-case (e.g. DS_1). Nonetheless, focusing only on stability, potentially, leads to poor overall control quality, since the system is optimized towards cases that might appear with only a low probability (e.g. DS_2). Therefore, it is essential and possible to achieve both safety (worst-case stability) and high level of expected control quality (e.g. DS_3).

[3] For simplicity of this example, we consider the priorities given. As shown later, priority assignment is also part of our co-design approach.

[4] Note that for each design solution DS_i, the controllers are synthesized for the given periods and the expected sensor–actuator delays which are obtained using system simulation.

VI. Problem Formulation

The inputs for our co-design problem are

- a set of plants $\mathbf{P}$ to be controlled,
- a set of control applications $\mathbf{\Lambda}$,
- a set of suggested sampling periods $\mathbf{H}_i$ for each Λ_i,
- execution-time probability functions ξ_{ij} of the tasks with their best-case and worst-case execution times c_{ij}^{b} and c_{ij}^{w}; transmission times $\gamma_{ij(j+1)}$ of messages,
- a distributed platform,
- a mapping function map for tasks to computation nodes.

The outputs are the period h_i for each control application Λ_i, unique priority ρ_i for each application Λ_i, and the control law $\boldsymbol{u}_i$ for each plant $P_i \in \mathbf{P}$.

The final control quality is captured by the weighted sum of the individual control costs $J_{\Lambda_i}^{\mathrm{e}}$ (Equation 2) of all applications $\Lambda_i \in \mathbf{\Lambda}$. To guarantee stability, the worst-case control cost $J_{\Lambda_i}^{\mathrm{w}}$ (Equation 3) must have a finite value. However, in addition to worst-case stability, the designer may require an application to satisfy a certain degree of robustness. Hence, the optimization problem is formulated as:

$$\begin{aligned} &\min_{\boldsymbol{h},\boldsymbol{u},\boldsymbol{\rho}} \sum_{P_i \in \mathbf{P}} w_{\Lambda_i} J_{\Lambda_i}^{\mathrm{e}} \\ &\text{s.t.} \quad J_{\Lambda_i}^{\mathrm{w}} < \bar{J}_{\Lambda_i}^{\mathrm{w}}, \quad \forall P_i \in \mathbf{P}, \end{aligned} \tag{5}$$

where the weights w_{Λ_i} are determined by the designer. The application Λ_i is the synthesized controller corresponding to the plant P_i. Further, $\bar{J}_{\Lambda_i}^{\mathrm{w}}$ captures the limit on tolerable worst-case cost for Λ_i and is decided by the designer. Therefore, the constraints in the formulation ensure satisfaction of the robustness requirements. If the requirement for an application Λ_i is only to be stable in the worst-case, the constraint on the worst-case control cost $J_{\Lambda_i}^{\mathrm{w}}$ is to be finite.

VII. Co-design Approach

The overall flow of our approach is illustrated in Figure 3. In each iteration, each control application is assigned a period using our period optimization algorithm (Section VII-A). For a certain period assignment, we proceed with priority optimization and control synthesis (Section VII-B). The assigned priorities should provide high quality of control and meet the worst-case performance requirements. Having assigned the periods and priorities and synthesized the controllers, we perform system simulation to extract delay distributions and compute the expected control cost. The algorithm terminates once the search method cannot find a better solution.

A. Period Optimization

The period optimization is performed using the coordinate search method [25] combined with the direct search method [26]. Both belong to the class of derivative-free optimization [25]. Derivative-free optimization is often used when the objective function is not available explicitly (in our case the objective function is calculated as result of the sequence inside the loop in Figure 3) and it is time consuming to obtain the derivatives using finite differences. The optimization is performed in two steps. In the first step, the coordinate search method, guided by the expected control performance, while considering the worst-case performance requirements, identifies a promising region in the search space. In other words, since shorter period often leads to better control performance, the coordinate search method, iteratively, assigns shorter periods to controllers which violate their worst-case robustness requirements or provide poor expected control performance. In the second step, the direct search method performs the search in the promising region identified in the first step. The direct search method iteratively performs a set of exploratory moves to acquire knowledge concerning the behavior of the objective function, identifies a promising search direction, and moves along the identified direction.

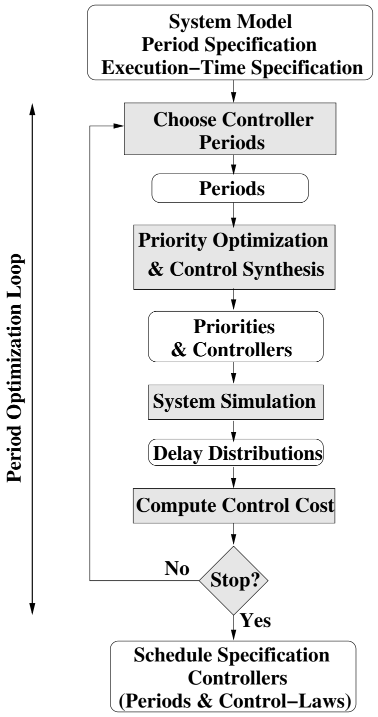

Fig. 3. Overall flow of our approach

Thus far, the period optimization approach corresponding to the loop in Figure 3 is discussed. Inside the loop, priority assignment and control synthesis is performed such that the design goals are achieved. The next subsection describes the optimization procedure performed inside the loop.

B. Priority Optimization and Control Synthesis

The priority optimization is done in two steps. The first step is performed exclusively based on the expected control performance. The second step assigns the priorities to increase the expected performance and meet the worst-case control performance requirements while preserving the already established priority assignment from the first step, as far as possible.

In the first step, initial priorities are assigned based on the bandwidths of the closed-loop control applications, as computed by MATLAB. The bandwidth of a closed-loop control system indicates the speed of system response—the larger the bandwidth, the faster the response. Further, a higher bandwidth implies that the system is more sensitive to a given amount of delay. Analogous to the rate-monotonic priority assignment principle, we hence assign higher priorities to control applications with larger bandwidth, leading to smaller induced delays due to interference from other applications. The priority order found in this step is passed, via the sequence $\mathbf{S}$, as an input to the optimization process in the next step. The sequence $\mathbf{S}$ contains the control applications in an ascending order of closed-loop bandwidth. It should be noted that in the first step we only consider the expected control performance.

In the second step, the optimization is performed using a backtracking algorithm outlined in Algorithm 1. The algorithm traverses the design solutions in such a way that

Algorithm 1 Priority Optimization

```
    % $\mathbf{S}$: sequence of remaining applications;
    % MAX_NUM: the max number of nodes the algorithm can visit;
    % *counter*: the number of nodes visited so far;
 1: function BACKTRACK($\mathbf{S}$, *priority*)
 2:     if $\mathbf{S} == \emptyset$ then
 3:         • Response-time analysis for sensors and actuators;
 4:         • Jitter and delay analyses $\Delta^{\mathrm{w}}_{s}$, $\Delta^{\mathrm{w}}_{a}$, $L$ for all $\Lambda_i \in \mathbf{\Lambda}$;
 5:         • Simulation to find the expected sensor–actuator delays
              and to find the sensors and actuators delay distributions;
 6:         • Control-law synthesis and delay compensation;
 7:         • Compute the worst-case control costs $J^{\mathrm{w}}_{\Lambda_i}$ for all $\Lambda_i \in \mathbf{\Lambda}$;
 8:         if $J^{\mathrm{w}}_{\Lambda_i} < \bar{J}^{\mathrm{w}}_{\Lambda_i}, \forall \Lambda_i \in \mathbf{\Lambda}$ then
 9:             • Compute the overall expected control cost $J^{\mathrm{e}}_{\mathrm{total}}$;
10:             • Update the best solution if the expected control cost
                  of the current solution is the best one found so far;
11:         end if
12:     end if
    % *The following loop iterates through the remaining applications*
    % $\mathbf{S}$, *while considering the order in sequence* $\mathbf{S}$;
13:     for all $\Lambda_i \in \mathbf{S}$ if *counter* < MAX_NUM do
14:         • *counter* = *counter* + 1;
15:         • Consider $\rho_i = priority$ and $hp(\Lambda_i) = \mathbf{S} \setminus \{\Lambda_i\}$;
16:         • Response-time analysis for sensor and actuator;
17:         • Jitter and delay analyses $\Delta^{\mathrm{w}}_{is}$, $\Delta^{\mathrm{w}}_{ia}$, $L_i$;
18:         • Simulation to find the expected sensor–actuator delay
              for $\Lambda_i$;
19:         • Control-law synthesis and delay compensation;
20:         • Compute the worst-case control cost $J^{\mathrm{w}}_{\Lambda_i}$ for $\Lambda_i$;
21:         if $J^{\mathrm{w}}_{\Lambda_i} < \bar{J}^{\mathrm{w}}_{\Lambda_i}$ then
22:             BACKTRACK($\mathbf{S} \setminus \Lambda_i$, *priority* + 1);
23:         end if
24:     end for
25: end function
```

it preserves, as far as possible, the priority order established in the first step. In other words, the backtracking algorithm applies the priority order established in the first step whenever there are multiple options available. The idea is to find the set of applications which can meet their worst-case performance requirements even if they are assigned the lowest priority. In order to investigate whether a control application meets its robustness requirement, we shall synthesize an LQG controller compensating for the expected sensor–actuator delay using the Jitterbug toolbox and MATLAB (Line 19). To obtain the expected sensor–actuator delay, we use our system simulation environment for distributed real-time systems (Line 18). In addition to controller synthesis, we need to obtain the nominal sensor–actuator delay, worst-case sensor jitter, and worst-case actuator jitter (Equation 4) (Line 17). To this end, we perform the best-case and worst-case response-time analyses as discussed in Section IV (Line 16). Having synthesized the controllers and found the values of the nominal sensor–actuator delay, worst-case sensor jitter, and worst-case actuator jitter, we use the Jitter Margin toolbox to check the robustness requirements (Lines 20–21). Then, we assign the lowest priority to the application (among the applications which can meet their robustness requirements) which has the lowest priority according to the priority order produced in the first step. We remove this application from the sequence of all applications and continue this process for the remaining applications (Line 22). Once all applications are assigned a unique priority, we shall perform a final check to make sure that the robustness requirements are satisfied (Lines 2–8). This is needed due to the fact that the response times, and consequently the nominal delay and jitters, are not only dependent on the set of higher priority applications, but also their actual priority order. If the robustness requirements are satisfied, we compute the overall expected control cost and update the final solution if it is better than the best found so far (Lines 9–10).

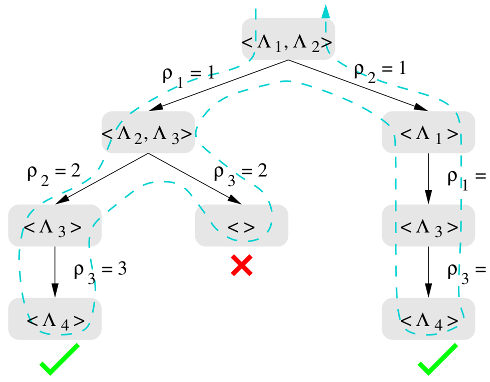

Fig. 4. An example of backtracking algorithm

The backtracking algorithm stops searching once a certain number of nodes, specified by the designer, in the search tree is visited (Line 13). Such a stopping condition provides the designer with the possibility of trading time for quality.

Figure 4 illustrates the backtracking algorithm using a small example. Let us consider four control applications $\mathbf{\Lambda} = \{\Lambda_1, \Lambda_2, \Lambda_3, \Lambda_4\}$. Furthermore, let us assume the first step priority assignment results in priority order $\langle \Lambda_1, \Lambda_2, \Lambda_3, \Lambda_4 \rangle$, i.e., application Λ_i has higher priority than application Λ_j iff $i > j$. The search tree is shown in Figure 4. The nodes are labeled with the sequence of applications, among the remaining applications, which meet their robustness requirements even if they are assigned the next lowest priority level. For instance, the root of the tree is labeled $\langle \Lambda_1, \Lambda_2 \rangle$, meaning that among all applications, only applications Λ_1 and Λ_2 can be assigned the lowest priority level (priority level 1). The edge labels depict priority assignment progress, e.g., $\rho_2 = 1$ indicates that application Λ_2 is assigned priority level 1. The dashed line depicts the order in which our backtracking algorithm traverses the search tree. Considering node $\langle \Lambda_2, \Lambda_3 \rangle$, either application Λ_2 or application Λ_3 can be assigned priority level 2. However, according to the first step priority optimization, it is beneficial, in terms of the expected control performance, to assign application Λ_2 priority level 2. If application Λ_3 is assigned priority level 2, in the next step, it turns out that this priority assignment cannot lead to a valid solution, considering the robustness requirements and, therefore, this branch is pruned. It is worth noting that the complete search tree for this example has 65 nodes.

VIII. Experimental Results

To investigate the efficiency of our proposed approach, several experiments have been conducted. Our proposed approach (EXP–WST) is compared against a baseline approach that only considers the expected performance (EXP) for a set of 100 benchmarks. The plants considered in each benchmark are chosen randomly from a database consisting of inverted pendulums, ball and beam processes, DC servos, and harmonic oscillators [16]. These plants are considered to be representatives of realistic control applications and are extensively used for experimental evaluation. In our benchmarks, the number of control applications varies from 2 to 11. The tasks of the task chain models of control applications are mapped randomly on platforms consisting of 2 to 6 computation nodes connected via a bus. Without loss of generality, our goal here is to find high-quality stable design solutions (the constraints on the worst-case control costs are to have finite values).

The evaluation of the proposed EXP–WST approach is performed against an optimization approach, called EXP, which only takes the expected control performance into consideration. While the period assignment in this approach is

TABLE I
Experimental Results

Number of control applications	Comparison with EXP approach	
	Difference $\left(\frac{J^{\mathrm{e}}_{\mathrm{EXP-WST}} - J^{\mathrm{e}}_{\mathrm{EXP}}}{J^{\mathrm{e}}_{\mathrm{EXP-WST}}}\right) \times 100$	Percentage of invalid solutions
2–3	0%	20%
4–5	1%	85%
6–7	6%	90%
8–9	10%	90%
10–11	11%	95%
Average	6%	76%

similar to our proposed approach, of course without considering the worst-case robustness requirements, the priority assignment is done using a genetic algorithm similar to [10]. In principle, the EXP approach should outperform our proposed approach in terms of expected control cost since the search is not constrained by worst-case stability requirements. The comparison has been made considering the relative expected control cost difference $\frac{J^{\mathrm{e}}_{\mathrm{EXP-WST}} - J^{\mathrm{e}}_{\mathrm{EXP}}}{J^{\mathrm{e}}_{\mathrm{EXP-WST}}}$, where $J^{\mathrm{e}}_{\mathrm{EXP}}$ and $J^{\mathrm{e}}_{\mathrm{EXP-WST}}$ are the expected control costs of the final solutions found by the EXP and the EXP–WST approaches, respectively. The results are shown in the second column of Table I. Our optimization approach, while guaranteeing stability, is on average only 6% away from the EXP approach, in terms of the expected control performance. However, since the EXP approach does not take the worst-case stability into consideration, it is possible that the stability of the final solution cannot be guaranteed. The percentage of the benchmarks for which stability is not guaranteed (invalid solutions) is shown in the third column of Table I. It can be seen that, on average, the EXP approach leads to potentially unstable design solutions for 76% of benchmarks.

We have measured the runtime of our proposed approach on a PC with a quad-core CPU running at frequency 2.83 GHz, 8 GB of RAM, and Linux operating system. The runtime of our algorithm is shown in Figure 5 as a function of the number of control applications. It can be seen that for systems with 11 control applications our proposed approach can find high-quality stable design solutions in less than one hour. The runtime of the optimization procedure with the EXP approach is also shown in Figure 5.

To sum up, we have shown the efficiency of our design approach which guarantees the worst-case stability of the system while providing high expected control performance.

IX. Conclusions

Sharing of the available computation and communication resources by control applications is commonplace in cyber-physical systems. Such resource sharing might lead to poor control performance or may even jeopardize the stability of applications if not properly taken into account during design. Therefore, not only the robustness and stability should be taken into account during the design process, but also the quality of control. In this paper, we have proposed an integrated approach for designing high performance cyber-physical systems with robustness guarantees and validated the efficiency of our proposed approach.

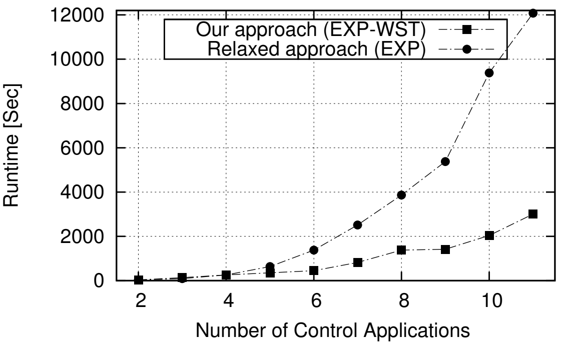

Fig. 5. Runtime of proposed approach

References

[1] Björn Wittenmark et al. “Timing Problems in Real-Time Control Systems”. In: *Proceedings of the American Control Conference.* 1995, pp. 2000–2004.

[2] K. E. Årzén et al. “An Introduction to Control and Scheduling Co-Design”. In: *Proceedings of the 39th IEEE Conference on Decision and Control.* 2000, pp. 4865–4870.

[3] D. Seto et al. “On Task Schedulability in Real-Time Control Systems”. In: *Proceedings of the 17th IEEE Real-Time Systems Symposium.* 1996, pp. 13–21.

[4] H. Rehbinder and M. Sanfridson. “Integration of Off-Line Scheduling and Optimal Control”. In: *Proceedings of the 12th Euromicro Conference on Real-Time Systems.* 2000, pp. 137–143.

[5] Anton Cervin et al. “The Jitter Margin and Its Application in the Design of Real-Time Control Systems”. In: *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications.* 2004.

[6] Truong Nghiem et al. “Time-triggered implementations of dynamic controllers”. In: *Proceedings of the 6th ACM & IEEE International conference on Embedded software.* 2006, pp. 2–11.

[7] E. Bini and A. Cervin. “Delay-Aware Period Assignment in Control Systems”. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium.* 2008, pp. 291–300.

[8] Fumin Zhang et al. “Task Scheduling for Control Oriented Requirements for Cyber-Physical Systems”. In: *Proceedings of the 29th IEEE Real-Time Systems Symposium.* 2008, pp. 47–56.

[9] Payam Naghshtabrizi and João Pedro Hespanha. “Analysis of Distributed Control Systems with Shared Communication and Computation Resources”. In: *Proceedings of the 2009 American Control Conferance (ACC).* 2009.

[10] S. Samii et al. “Integrated Scheduling and Synthesis of Control Applications on Distributed Embedded Systems”. In: *Proceedings of the Design, Automation and Test in Europe Conference.* 2009, pp. 57–62.

[11] Rupak Majumdar et al. “Performance-aware scheduler synthesis for control systems”. In: *Proceedings of the 9th ACM international conference on Embedded software.* 2011, pp. 299–308.

[12] Dip Goswami et al. “Time-Triggered Implementations of Mixed-Criticality Automotive Software”. In: *Proceedings of the 15th Conference for Design, Automation and Test in Europe.* 2012.

[13] Pratyush Kumar et al. “A Hybrid Approach to Cyber-Physical Systems Verification”. In: *Proceedings of the 49th Design Automation Conference.* 2012.

[14] Amir Aminifar et al. “Desiging High-Quality Embedded Control Systems with Guaranteed Stability”. In: *Proceedings of the 33th IEEE Real-Time Systems Symposium.* 2012, pp. –.

[15] Payam Naghshtabrizi and João Pedro Hespanha. “Distributed Control Systems with Shared Communication and Computation Resources”. Position paper for the National Workshop on High Confidence Automotive Cyber-Physical Systems. 2008.

[16] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems.* 3rd ed. Prentice Hall, 1997.

[17] B. Lincoln and A. Cervin. “Jitterbug: A Tool for Analysis of Real-Time Control Performance”. In: *Proceedings of the 41st IEEE Conference on Decision and Control.* 2002, pp. 1319–1324.

[18] A. Cervin. “Stability and Worst-Case Performance Analysis of Sampled-Data Control Systems with Input and Output Jitter”. In: *Proceedings of the 2012 American Control Conference (ACC).* 2012.

[19] Ken Tindell and John Clark. “Holistic schedulability analysis for distributed hard real-time systems”. In: *Microprocess. Microprogram.* 40.2-3 (1994), pp. 117–134.

[20] J.C. Palencia Gutierrez et al. “On the schedulability analysis for distributed hard real-time systems”. In: *Proceedings of the 9th Euromicro Workshop on Real-Time Systems.* 1997, pp. 136–143.

[21] J.C. Palencia Gutierrez et al. “Best-case analysis for improving the worst-case schedulability test for distributed hard real-time systems”. In: *Proceedings of the 10th Euromicro Workshop on Real-Time Systems.* 1998, pp. 35–44.

[22] R. Henia et al. “System level performance analysis - the SymTA/S approach”. In: *IEE Proceedings Computers and Digital Techniques* 152.2 (2005), pp. 148–166.

[23] J.C. Palencia and M. Gonzalez Harbour. “Schedulability analysis for tasks with static and dynamic offsets”. In: *Proceedings of the 19th IEEE Real-Time Systems Symposium.* 1998, pp. 26–37.

[24] Robert Davis et al. “Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised”. In: *Real-Time Systems* 35 (3 2007), pp. 239–272.

[25] J. Nocedal and S.J. Wright. *Numerical Optimization.* 2nd ed. Springer, 1999.

[26] Robert Hooke and T. A. Jeeves. ““Direct Search” Solution of Numerical and Statistical Problems”. In: *J. ACM* 8.2 (1961), pp. 212–229.