Journal of Systems Architecture 59 (2013) 1215–1230

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Multi-layered scheduling of mixed-criticality cyber-physical systems

Reinhard Schneider[a,*], Dip Goswami[b], Alejandro Masrur[c], Martin Becker[a], Samarjit Chakraborty[a]

[a] Institute for Real-Time Computer Systems, TU Munich, Germany
[b] Department of Electrical Engineering, TU Eindhoven, The Netherlands
[c] Department of Computer Science, TU Chemnitz, Germany

ARTICLE INFO

Article history:
Available online 25 September 2013

Keywords:
Scheduling
Cyber-physical systems
Mixed-criticality systems
Real-time systems

ABSTRACT

In this paper, we deal with the schedule synthesis problem of *mixed-criticality cyber-physical systems* (MCCPS), which are composed of hard real-time tasks and feedback control tasks. The real-time tasks are associated with deadlines that must always be satisfied whereas feedback control tasks are characterized by their Quality of Control (QoC) which needs to be optimized. A straight-forward approach to the above scheduling problem is to translate the QoC requirements into deadline constraints and then, to apply traditional real-time scheduling techniques such as Deadline Monotonic (DM). In this work, we show that such scheduling leads to overly conservative results and hence is not efficient in the above context. On the other hand, methods from the *mixed-criticality* systems (MC) literature mainly focus on tasks with different criticality *levels* and certification issues. However, in MCCPS, the tasks may not be fully characterized by only criticality *levels*, but they may further be classified according to their criticality *types*, e.g., *deadline-critical* real-time tasks and *QoC-critical* feedback control tasks. On the contrary to traditional deadline-driven scheduling, scheduling MCCPS requires to integrate both, deadline-driven and QoC-driven techniques which gives rise to a challenging scheduling problem. In this paper, we present a *multi-layered* schedule synthesis scheme for MCCPS that aims to jointly schedule *deadline-critical*, and *QoC-critical* tasks at different scheduling layers. Our scheduling framework (i) integrates a number of QoC-oriented metrics to capture the QoC requirements in the schedule synthesis (ii) uses *arrival curves* from real-time calculus which allow a general characterization of task triggering patterns compared to simple task models such as *periodic* or *sporadic*, and (iii) has pseudo-polynomial complexity. Finally, we show the applicability of our scheduling scheme by a number of experiments.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

The trend in many cost-driven domains such as automotive evolves from *federated architectures* which are comprised of many networked electronic control units (ECUs) each implementing a specific functionality, to more *integrated architectures* which require the integration of several functions on more powerful processing components [2]. As a consequence, such systems need to support the execution of software with different criticality, and hence, are generally referred to as *mixed-criticality systems* (MC). Such systems are prevalent in various application domains such as automotive systems, avionics, and automation industry, and are certified according to stringent industrial standards which precisely specify different *levels* of criticality. For instance, the DO-178B specification, an avionics software standard, describes five levels (from A to E) to indicate the criticality of functions [5]. Similarly, the ISO 26262, an automotive safety standard, introduces four safety integrity levels (ASIL) [6]. In this context, certification in MCs has drawn a lot of attention [7,15,8]. Towards this, tasks with high criticality are subject to stringent certification processes while less critical functions have to pass a less rigorous process. For example, depending on the criticality of a task the certification authority may specify different requirements to determine the value of the worst-case execution time (WCET). That is, for certification, high critical tasks may be characterized by more pessimistic WCET estimates for schedulability analysis than for conventional (internal) analysis. In particular, many of the high critical tasks in a car or an airplane implement safety–critical control applications where embedded computers monitor and regulate physical processes via feedback loops. Such systems are characterized by tight interactions of the computational components with the physical processes and are commonly referred to as Cyber-Physical Systems (CPS) [1]. Moreover, in highly integrated architectures the safety–critical feedback control tasks co-exist with conventional hard real-time tasks on the same platform sharing resources. More specifically, such systems are made up of a mix of (i) time-critical real-time tasks with hard deadlines, and (ii) safety–critical feedback control tasks imposing Quality of Control (QoC) requirements. In this work, we refer to such systems as *mixed-criticality cyber-physical systems* (MCCPS).

* Corresponding author. Tel.: +49 89 289 23565.
E-mail address: reinhard.schneider@rcs.ei.tum.de (R. Schneider).

Here, the difference to conventional MCs is that in MCCPS the criticality of tasks may not be sufficiently characterized by discrete criticality *levels*, but in addition, the tasks may be classified according to criticality *types*. For example, two tasks may exhibit the same criticality *level*, e.g., a time-critical real-time task, and a safety–critical control task, however, their criticality *type* may differ. Consequently, the criticality types must be exploited in the schedule synthesis. To this end, we consider two criticality *types*: (i) *deadline-critical* real-time tasks, and (ii) *QoC-critical* control tasks. On the one hand, *deadline-critical* tasks are associated with hard real-time constraints, i.e., *deadlines*, that must always be satisfied. That is, any deadline miss may result in a malfunction of the system potentially causing a catastrophe (e.g., a task deploying an airbag). On the other hand, *QoC-critical* control tasks impose stability and QoC constraints. Clearly, a stable system is *necessary* for the correct functioning of control applications. However, in most feedback control systems, stability is not a *sufficient* condition to ensure the desired and safe functioning of the system, and hence, specified QoC requirements need to be met. For instance, for cruise control it is not sufficient to ensure that the car reaches the desired speed v set by the driver (i.e., stability) but it is also important that v is reached within a certain specified time t with an accuracy of a (i.e., QoC).

The (necessary) stability condition may be expressed as a deadline constraint of the corresponding control task, and hence, can be verified using conventional schedulability analysis. In contrast, the QoC requirements cannot be easily translated into deadlines.

- Since a shorter delay potentially improves QoC [27], the QoC optimization problem can be cast as the problem of finding the minimum deadlines for which schedulability is not violated.
- In addition, finding the optimal deadlines for the control tasks depends on the interplay between the feedback delays and associated QoC characteristics. While considering multiple control tasks, the problem becomes even harder due to the different sensitivity to the feedback delays.

Consequently, platforms implementing MCCPS need to be capable of meeting the requirements of both, *deadline-critical* and *QoC-critical* control tasks. This gives rise to a challenging scheduling problem which requires conjoining control engineering and real-time disciplines in an integrated scheduling framework.

Related work: There exists a wide range of literature on scheduling of real-time systems. Along the lines of scheduling in MCCPS, three major directions of related work have become apparent which we classify as work on (i) jointly scheduling tasks with both soft and hard time constraints (ii) scheduling MCs, and (iii) control/scheduling co-design.

In (i), the problem of jointly scheduling soft real-time tasks while guaranteeing the deadlines of hard real-time tasks has already been addressed in early research works. Burns et al. [16] and Davis [17] introduced a *dual priority* scheme for jointly scheduling tasks with hard and soft deadlines. The range of available priorities is distributed among different priority bands where priorities associated with the upper band represent highest priorities. Further, hard real-time tasks are assigned two priorities, from the upper and lower bands, respectively, whereas tasks with soft deadlines are scheduled at run-time in the lower band.

Server-based approaches such as Priority Exchange Sever [9], Sporadic Server [10], and Deferrable Server [11], have been studied to jointly schedule real-time tasks and best-effort tasks. These approaches primarily use most processor bandwidth to service the periodic real-time tasks, e.g., scheduled according to Rate Monotonic (RM). The remaining bandwidth is then used to service the best-effort tasks. Similarly, server mechanisms have also been studied applying dynamic priority assignment schemes such as EDF [12]. However, the server-based approaches are restricted to periodic real-time tasks and consider best-effort tasks to be processed in the background at lower priorities, potentially resulting in long response times. To overcome this drawback, polling servers have been introduced for periodically executing the soft real-time tasks at highest priorities. However, if the computational demands exceed the server capacity, then the execution of the soft-real time tasks might not be completed before the next release of the polling server, which effectively leads to long response times. Another approach has been presented by Lehoczky et al. [13] and Davis et al. [14]. They propose *slack stealing* algorithms to minimize soft real-time task response times whilst guaranteeing the deadlines of the hard real-time tasks. For this, the maximum amount of slack which may be stolen from the hard real-time tasks is computed. However, the presented techniques suffer from restrictions which limit their practical use to only a small range of task sets. More precisely, both approaches are restricted to strictly periodic tasks only, and the latter approach suffers from a high execution time overhead due to the algorithm, which makes the approach infeasible in practice.

In summary, the above mentioned works present contributions for jointly scheduling soft real-time or best-effort tasks and hard real-time tasks. However, the characterization of feedback control tasks is fundamentally different in the following ways:

- *Soft deadlines* may be occasionally missed and their violation does not seriously harm the system. Similarly, *firm deadlines* can be missed but there is an upper limit on the number of misses within a given time interval. Due to the stringent stability requirements, control tasks are generally classified as firm real-time tasks rather than soft real-time tasks. However, since we aim to optimize the QoC, we do not allow any deadline misses to avoid potential QoC degradation.
- The QoC of a control task is highly sensitive to its feedback delay. Consequently, the optimal priority assignment among multiple control tasks is not a traditional delay minimization problem. Rather, it depends on the interplay between the delay and associated QoC characteristic.

As a result, the above techniques are not applicable in our setting.

In (ii), there has been a tremendous research effort towards analysis and synthesis of mixed-criticality systems. Generally, research on MCs is mainly centering on the question: *how to cope with the conflicting design requirements of safety assurance, resource efficiency, flexibility and extensibility*. Towards this, Islam et al. [18] propose a dependability-driven approach for the integration of both safety–critical and non-safety critical software functionalities on shared resources. In this context, the mapping process considers functional and extra-functional constraints of dependability and real-time. De Niz et al. [21] and Lakshmanan et al. [22] describe schedule synthesis techniques in MCs using *zero-slack scheduling* algorithms. Here, the goal is to minimize the utilization demand by reducing preemption of low-criticality tasks by tasks with high criticality using different modes. For this purpose, every task is either in *normal* mode where it is scheduled based on its fixed priority or in *critical* mode where the scheduler suspends tasks with low criticality such that critical tasks steal their slack. Further, Lakshmanan et al. [19] studied scheduling of mixed-criticality cyber-physical systems under overload conditions. This approach guarantees the timing requirements of the more critical tasks in overload situations. Recently, work related to verification and *certification* of MCs has gained a lot of attraction in academia and in industry. In this context, Vestal [20] assumes that tasks with different criticality levels are associated with different levels of assurance. That is, the higher the degree of assurance required (higher criticality), the larger and more conservative worst-case execution times (WCET) are considered for schedulability analysis. In other words, the WCET values must have the same level of assurance as required for the corresponding task. Towards this, it is shown that the classical real-time scheduling policies such as DM are not optimal for MCs. To overcome this drawback, they propose a multi-criticality scheduling technique as an extension of fixed priority scheduling (FPS). This line of research has been followed by a number of works which present contributions to various fields related to certification of MCs such as mixed-criticality workload models [7], mixed-criticality scheduling techniques [15], and response-time analysis for MCs [8].

Since the above lines of research made significant contributions in the field of certification and scheduling in MCs they are concerned with mixed-criticality task sets characterized by discrete criticality *levels*. In the setting we consider in this work, the quantification of mixed-criticality by criticality levels does not capture the QoC optimization of feedback control tasks. For this purpose, we extend the existing notion of mixed-criticality for MCCPS by *criticality types*, and we propose an integrated scheduling scheme which exploits the *criticality types* for optimizing QoC.

In the area of (iii) there has been considerable amount of work on schedule synthesis for control applications. In particular, the area of *control/scheduling co-design* [23] has recently gained a lot of attention with the goal to design the embedded platform, e.g., the schedules, and the controller, i.e., sampling periods and controller gains. In the area of scheduler design for optimized QoC, Voit et al. [25] studied the problem of optimally chosing the parameters of hierarchical schedules on the communication bus in order to improve multiple control performance metrics. Focusing on controller design, Bini et al. [27] studied the problem of optimal period assignment for multiple control tasks using approximate response-time analysis under FPS. Similarly, Wu et al. [29] propose an integrated approach to improve the control performance through proper selection of task periods and deadlines under EDF scheduling. The works presented by Goswami et al. [32], Jia et al. [28] and Majumdar et al. [31] allow transmission failures in the feedback loop. Exploiting this, the overall design of the controller and the scheduler is optimized. Aminifar et al. [26] present a design flow which uses both worst-case and average QoC and stability for determining periods, task schedules, and controller parameters. Similarly, Samii et al. [24] propose a design that integrates static and priority-based scheduling in the controller design with the goal to optimize QoC.

Although, these works made significant contributions in controller/scheduler co-design with the goal to optimize QoC, they are solely concerned with scheduling control tasks for optimized QoC. However, the work presented in this paper focuses on the joint schedule synthesis of *mixed* task sets comprising of feedback control tasks *and* hard real-time tasks. Such joint scheduling has been addressed in Goswami et al. [34] for a time-triggered architecture. In this work, we present a general framework for scheduling such mixed task sets under FPS.

This article extends an earlier version [33] of our work which investigated the basic MCCPS scheduling problem on an illustrative task set. In this paper, we extend the previous work in several directions: (i) we introduced multiple QoC-oriented metrics which can easily be integrated in our framework and enhance our analysis (ii) we provide run times of our scheduling algorithms, and we show that our results are close to the theoretical optimal solutions whereas the earlier version [33] contains no such evaluations, (iii) we perform a sensitivity analysis to study the sensitivity of our approach to variations in the real-time task parameters, and we compare our results with the classical DM approach and a QoC-aware priority assignment scheme, and finally (iv) we implemented the presented concept in a controller/plant co-simulation framework to apply and verify the theoretical outcomes of our analysis.

Problem statement and contributions: In this work, we address the schedule synthesis problem for a task set that is mapped onto a shared resource implementing an MCCPS as depicted in Fig. 1. The task set consists of a number of *deadline-critical* tasks $T \in \tau_{rt}$ (see T_1, T_2, T_3), and a number of *QoC-critical* control tasks $T \in \tau_c$ (see T_4, T_5). Due to its execution time and interference from other tasks, each task experiences a *response time* or *delay* D. Further, each *deadline-critical* task is associated with a relative deadline d which must always be satisfied. On the other hand, the QoC of a control task deteriorates with increasing D according to the function $J(D)$, which depends on the controller design, the system to be controlled, and the delay D experienced by the control task. Further, some control tasks are intrinsically more sensitive to delay than others. For instance, reducing the delay of a control task T_4 (see Fig. 1) by some amount Δ might result in a QoC increase which is less than that of reducing T_5's delay by the same amount Δ. As a consequence, finding a priority assignment that leads to the maximum possible overall QoC requires analyzing the behavior of all control tasks for all possible priority levels. That is, if n is the number of all tasks in the system, finding the optimal priority assignment requires $\mathcal{O}(n!)$ time (i.e., factorial of n), which results in an exponential complexity on the number of tasks in the system. An algorithm with exponential complexity is not suitable for large- or even mid-size problem settings, and hence, there is a need for designing efficient scheduling algorithms. To address this problem we need to jointly consider *deadline-driven* and *QoC-driven* scheduling techniques in an integrated schedule synthesis framework as depicted in Fig. 1. In this paper, the overall goal is to design a priority assignment technique with the following objectives:

1. All the real-time tasks meet their deadlines d in the worst-case
$$D \leqslant d, \quad \forall T \in \tau_{rt}. \tag{1}$$
2. The overall QoC of all the control tasks is maximized, i.e., maximize
$$J^* = \sum_{\forall T \in \tau_c} J(D). \tag{2}$$

Clearly, for *deadline-critical* tasks, it is meaningful to assign priorities according to the Deadline Monotonic (DM) policy, i.e., the shorter the deadline of a task the higher its priority, since this results in an *optimal* priority assignment [35]. That is, if a fixed-priority schedule is *feasible* under any priority assignment, then DM will also be feasible. However, the inverse does not hold. In other words, if DM is feasible, it does not necessarily imply that any other schedule (different to DM) would also be feasible. For MCCPSs, we define an *optimal* schedule such that (i) the schedule is *feasible*, i.e., all deadlines are satisfied (see (1)), and (ii) the overall QoC is maximum (see (2)). Consequently, the DM scheme does not necessarily provide optimal results as the criterion in (ii) is not explicitly taken into account, even though it provides an optimal priority assignment for purely deadline-driven tasks as described in (i). For this reason, in this work, we present a multi-layered scheduling scheme to synthesize schedules for *deadline-critical* and *QoC-critical* tasks according to 1 and 2. Our scheme can be summarized as follows:

- In Section 2, we give an overview of the timing analysis techniques and the control theoretic fundamentals that we use in our scheduling framework. In particular, the triggering patterns of tasks are modeled as *arrival curves* which allow for a more expressive representation of task triggering patterns compared

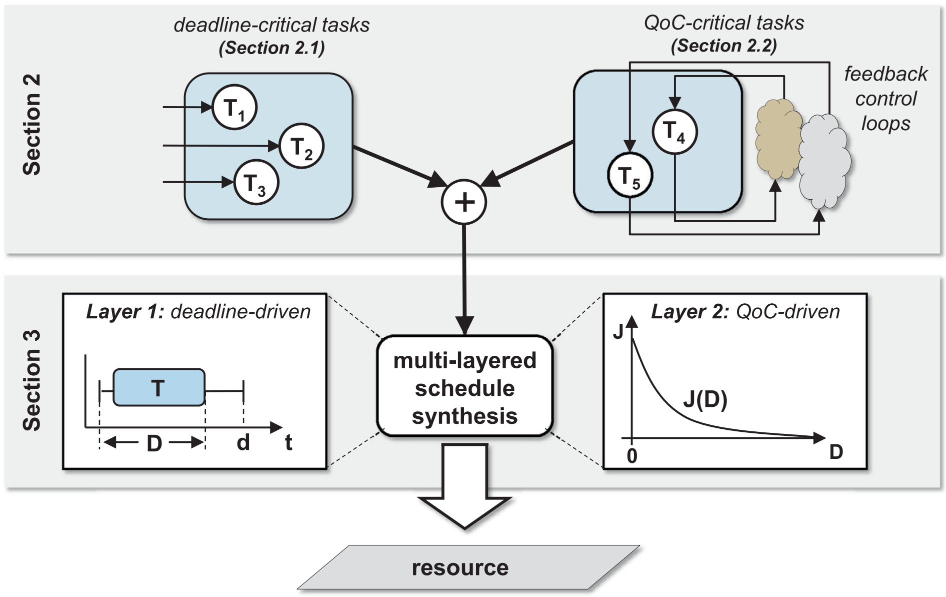

Fig. 1. Mixed-criticality cyber-physical system and overview of scheme.

to standard models. Hence, our scheduling framework is neither restricted to periodic task triggerings nor to any deadline constraints such as deadlines smaller than periods.

- In Section 3, we describe the *multi-layered* scheduling scheme which integrates appropriate scheduling strategies at each *layer* for different criticality *types*. That is, *deadline-critical* tasks are scheduled at *layer 1* with the goal to guarantee deadlines whereas *QoC-critical* control tasks are scheduled at *layer 2* with respect to stability and QoC optimization. For this purpose, we introduce several QoC-oriented metrics that allow capturing the QoC requirements during schedule synthesis. Finally, we show that our algorithm has *pseudo-polynomial* complexity.
- In Section 4, we demonstrate the applicability of our proposed scheme by a number of experiments. We evaluate and compare the results with the DM and a QoC-aware approach, and show a sensitivity analysis using several task sets for different resource utilizations. Finally, to demonstrate the usefulness of our scheme, we also implemented a MCCPS task set on a real hardware/software setup. Our proposed approach may be seamlessly applied with fixed priority preemptive schedulers which are widely used in the automotive domain (OSEK, AUTOSAR).

2. Theoretical background

Before we present our scheduling scheme, we first give an overview on the analytical techniques we use to model the MCCPS, and to compute the delays of tasks. For this purpose, we use *real-time calculus* (RTC) [37] which allows computing and propagating analysis results in a compositional manner, and hence, RTC exhibits expedient features for integration in a modular scheduling framework. Further, we give a brief review on control theory basics and we present the control task model used in this work.

2.1. Real-time tasks

RTC is an analytical framework for worst-case performance analysis of real-time systems. RTC allows modeling of (i) the triggering pattern of tasks (*event model*) which generate execution demands on a resource, and (ii) the service offered by a resource (*resource model*), e.g., a processor, to each task running on it. The mathematical framework of RTC is based on $(\min,+)$ and $(\max,+)$ algebra and has its roots in network calculus [36].

Terms and definitions: Let $\overline{\mathbb{R}} = \mathbb{R} \cup \{+\infty, -\infty\}$ where $\mathbb{R}$ is the set of real numbers and $\mathcal{F}$ be the set of monotonic functions $\mathcal{F} = \{f : \mathbb{R}^+ \to \overline{\mathbb{R}} \mid \forall s < t,\ 0 \leqslant f(s) \leqslant f(t)\}$ where $\mathbb{R}^+$ is the set of non-negative real numbers. Further, the *supremum* (sup) of a set $S \subseteq \mathcal{F}$ is the smallest $U \in \mathcal{F}$ such that $h \leqslant U$ for all $h \in S$. Similarly, the *infimum* (inf) of S is the largest $L \in \mathcal{F}$ such that $h \geqslant L$ for all $h \in S$.

The $(\min,+)$ convolution $\otimes$ and deconvolution $\oslash$ operators are defined as: $\forall f,g \in \mathcal{F},\ \forall t \in \mathbb{R}^+$

$$(f \otimes g)(t) = \inf\{f(s) + g(t-s) \mid 0 \leqslant s \leqslant t\},$$
$$(f \oslash g)(t) = \sup\{f(t+u) - g(u) \mid u \geqslant 0\}.$$

Similarly, the $(\max,+)$ convolution $\overline{\otimes}$ and deconvolution $\overline{\oslash}$ operators are defined as: $\forall f,g \in \mathcal{F},\ \forall t \in \mathbb{R}^+$

$$(f \overline{\otimes} g)(t) = \sup\{f(s) + g(t-s) \mid 0 \leqslant s \leqslant t\},$$
$$\left(f \overline{\oslash} g\right)(t) = \inf\{f(t+u) - g(u) \mid u \geqslant 0\}.$$

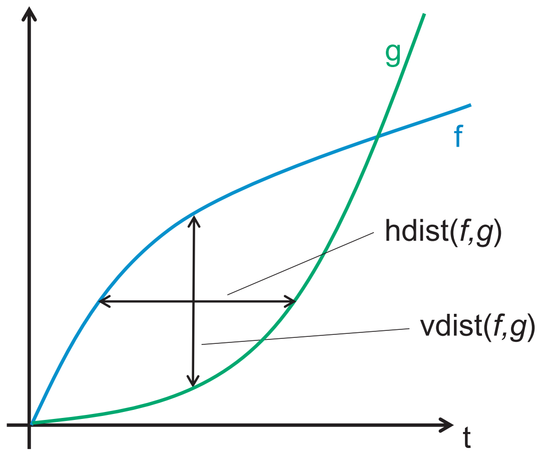

Fig. 2. Delay and backlog computation.

The maximum vertical and horizontal deviation (distance) between two functions $f, g \in \mathcal{F}$ are given by: (see Fig. 2)

$$\mathrm{vdist}(f,g) \overset{\mathrm{def}}{=} \sup\{ f(t) - g(t) \mid t \geqslant 0 \}, \tag{3}$$

$$\mathrm{hdist}(f,g) \overset{\mathrm{def}}{=} \sup\{ \inf\{\tau \geqslant 0 | f(t) \leqslant g(t+\tau)\} \mid t \geqslant 0\}. \tag{4}$$

Further, a function $f \in \mathcal{F}$ is sub-additive iff $f(x+y) \leqslant f(x) + f(y)$ for all x and y in $\mathbb{R}^+$. Similarly, f is super-additive iff $f(x+y) \geqslant f(x) + f(y)$ for all x and y in $\mathbb{R}^+$. In this paper, we assume that all given upper (lower) functions satisfy sub-additivity (super-additivity) before the analysis.

Event model: Triggering patterns of tasks are modeled using a *count-based abstraction* where an arrival pattern of a stream is modeled as a cumulative function $R(t)$ denoting the number of events that arrive during the time interval $(0, t]$. The maximum and minimum number of events that are recorded during *any* time interval of length Δ is represented by a pair of *arrival functions* $\alpha = (\alpha^u, \alpha^l)$ that is defined as

$$\forall \Delta \geqslant 0,\ \forall t \geqslant 0: \quad \alpha^l(\Delta) \leqslant R(\Delta + t) - R(t) \leqslant \alpha^u(\Delta). \tag{5}$$

Arrival curves allow for an expressive characterization of event streams which are able to represent standard event models, e.g., *periodic*, *periodic with jitter* and *sporadic*, as well as arbitrary arrival patterns. Standard event arrival patterns are often specified by the tuple (p, j), where p denotes the period, and j the jitter. The corresponding pair of arrival curves to this specification is modeled as

$$\alpha^l(\Delta) = \left\lfloor \frac{\Delta - j}{p} \right\rfloor, \quad \text{and} \quad \alpha^u(\Delta) = \min\left\{ \left\lceil \frac{\Delta + j}{p} \right\rceil, \left\lceil \frac{\Delta}{j} \right\rceil \right\}. \tag{6}$$

Fig. 3 shows an example of a pair of arrival curves specified as $\alpha = (10, 55)$, i.e., $p = 10,\ j = 55$.

Resource model: Similarly, resource capacities are captured by a cumulative function $C(t)$ denoting the number of events that can be processed by a resource in the time interval $(0, t]$. The maximum and minimum number of events that can be processed in *any* time interval of length Δ is upper- and lower-bounded by a pair of *service functions* $\beta = (\beta^u, \beta^l)$ which are defined as

$$\forall \Delta \geqslant 0,\ \forall t \geqslant 0: \quad \beta^l(\Delta) \leqslant C(\Delta + t) - C(t) \leqslant \beta^u(\Delta). \tag{7}$$

Further, β can also be expressed in terms of the maximum and minimum number of available resource units, e.g., processor cycles.

Compositional analysis: Let us consider an arrival pattern which is bounded by the arrival functions $\alpha = (\alpha^u, \alpha^l)$ and triggering task T on a resource with available service $\beta = (\beta^u, \beta^l)$ as illustrated in Fig. 4(a)). Further, let us assume the buffer that stores arriving events has infinite capacity.

Then, according to (3) and (4) the maximum backlog B at the input buffer, i.e., the maximum buffer space required to buffer this event stream, and the maximum delay D experienced by the input stream α are given by

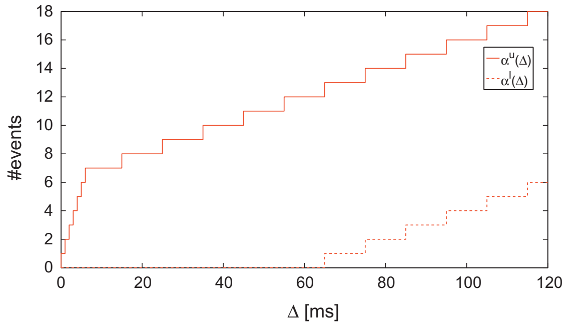

Fig. 3. Arrival curve $\alpha = (10, 55)$.

$$B = \mathrm{vdist}(\alpha^u, \beta^l), \quad \text{and} \quad D = \mathrm{hdist}(\alpha^u, \beta^l). \tag{8}$$

The bounds on the output arrival functions α' and remaining service functions β' for a greedy preemptive processing component are computed as follows [37]:

$$\alpha^{u\prime} = \min\left\{ (\alpha^u \otimes \beta^u) \oslash \beta^l,\ \beta^u \right\}, \tag{9}$$

$$\alpha^{l\prime} = \min\left\{ (\alpha^l \oslash \beta^u) \otimes \beta^l,\ \beta^l \right\}, \tag{10}$$

$$\beta^{u\prime} = (\beta^u - \alpha^l)\ \overline{\oslash}\ 0, \tag{11}$$

$$\beta^{l\prime} = \left(\beta^l - \alpha^u\right) \overline{\otimes}\ 0. \tag{12}$$

Further, the bounds on the processed arrival curves $\alpha\prime = (\alpha^{u\prime}, \alpha^{l\prime})$, and the remaining service $\beta\prime = (\beta^{u\prime}, \beta^{l\prime})$ can be used in a compositional manner. Consider the example in Fig. 4(b)) where the input streams α_1 and α_2 are processed by the tasks T_1 and T_2 on a resource with total service β_1. Assume the output of T_1 is processed by task T_3 using service β_2, and T_1 and T_2 are scheduled according to fixed priority preemptive scheduling (FPS) where T_1 is assigned a higher priory than T_2. Then, the full service β_1 is available to the task with the highest priority (T_1) to process its input stream (α_1). The backlog and worst-case delay are computed as $B_1 = \mathrm{vdist}(\alpha_1^u, \beta_1^l)$ and $D_1 = \mathrm{hdist}(\alpha_1^u, \beta_1^l)$. The bounds on the processed output stream α_1' and the remaining service β_1' are computed using (9)–(12). Now, β_1' is used to compute the performance bounds of task T_2 which processes input stream α_2, i.e., $B_2 = \mathrm{vdist}(\alpha_2^u, \beta_1^{l\prime})$ and $D_2 = \mathrm{hdist}(\alpha_2^u, \beta_1^{l\prime})$. In a similar way, the processed output stream α_1' is used as an input for task T_3 using service β_2. The total end-to-end delay experienced by the input stream α_1 can now be computed as $\tilde{D}_1 = \mathrm{hdist}(\alpha_1^u, \beta_1^l) + \mathrm{hdist}(\alpha_1^{u\prime}, \beta_2^l)$.

2.2. Control tasks

Before describing the control task models, we briefly review some basic concepts from control theory. A feedback control system aims to achieve the desired behavior of a dynamical system by applying appropriate inputs (that are computed based on the feedback signals) to the system. In general a dynamical system is modeled by a set of differential equations called the *state-space model*,

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{13}$$

where $x(t) \in R^n$ is the system *state* and $u(t) \in R$ is the *control input* to the system. $A \in R^{n \times n}$ and $B \in R^{n \times 1}$ are the system and input matrices, respectively. A feedback control loop performs mainly three sequential operations:

- measure the states $x(t)$ (*measure*),
- compute input signal $u(t)$ (*compute*) and,
- apply the computed $u(t)$ to the plant (13) (*actuate*).

Performing these operations in a continuous fashion in any implementation platform requires infinite computational power. Hence, in a digital implementation platform for such a feedback loop, these operations are performed only at discrete-time intervals (sampling instants) t_k for $k \in \mathbb{N}_0$. The time interval between two consecutive executions of a feedback control loop (*control task*) is the sampling period $t_{k+1} - t_k = h$. In an ideal implementation with constant sampling period h, the continuous plant dynamics can be transformed into a discrete-time dynamics,

$$x[k+1] = A_d x[k] + B_d u[k], \tag{14}$$

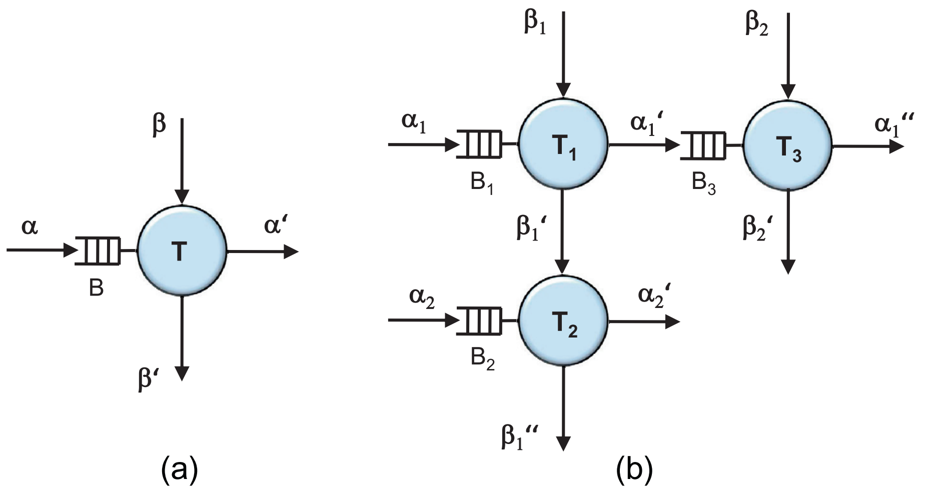

Fig. 4. (a) RTC processing model, (b) Compositional analysis.

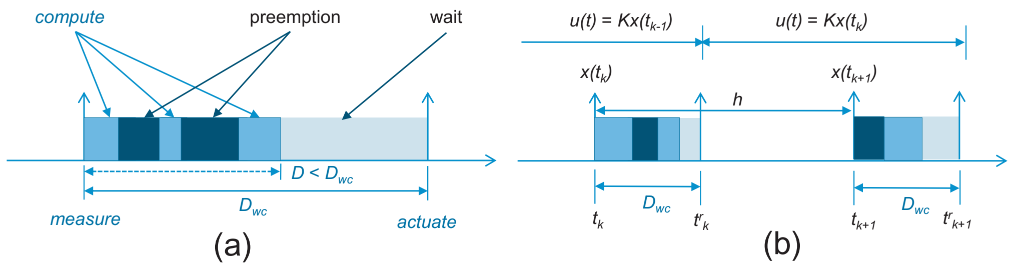

Fig. 5. Control task model.

where

$$A_d = e^{Ah}, \quad B_d = \int_0^h e^{At}dt \cdot B.$$

In general, a feedback control algorithm utilizes the *full* or *reduced-order* states in computing $u[k]$ so as to meet certain high-level design requirements. In the case of full *state-feedback control*, the control law $u[k]$ is given by,

$$u[k] = Kx[k], \tag{15}$$

where K is the state-feedback gain which needs to be designed. Clearly, the closed-loop dynamics becomes as follows in this case,

$$x[k+1] = (A_d + B_d K)x[k].$$

The behavior of the closed-loop system depends on various properties of $(A_d + B_d K)$. The eigenvalues of $(A_d + B_d K)$ are called system *poles*. In this work, a desired set of system poles is achieved by designing the feedback gain K using *pole-placement* technique [38]. Such ideal implementation implicitly assumes *instantaneous* execution of three operations – *measure*, *compute* and *actuate*. The QoC that we achieve with an ideal implementation is referred to as *nominal* performance. In an embedded platform with limited computational capacity, these three operations consume a finite amount of time for their execution.

In general, variable feedback delay or jitter in the delay experienced by the control messages results in multiple *switching* subsystems (depending on the exact duration and sequence of the feedback delays) and the controller must be appropriately designed to *stabilize* the resulting switched system. However, it is very difficult to provide any analytical guarantees on the performance of such switched systems. Since our work mainly targets QoC-critical applications, we use a control task model such that the feedback delay remains constant, and hence, jitter can be avoided.

Our control task model is based on the Logical Execution Time (LET) paradigm [3] where the operations *measure*, *compute*, and *actuate* are performed sequentially in one task as depicted in Fig. 5(a). According to this model, we first analyze the worst-case response times[1] D_{wc} of the control tasks. Next, we ensure that the measure and the actuate operations are separated by exactly D_{wc} time units as depicted in the figure. This essentially means that the actual response times of the *jobs* of a control task may experience delays $D \leqslant D_{wc}$, but the actuation is only performed after D_{wc} time units (see *wait* in Fig. 5(a)) to realize constant delays in the feedback loops. Such a task model has the following advantages which we exploit in this work:

- In contrast to a contoller with *time-varying* feedback delay, a controller with *constant* feedback delay imposes less stringent deadline constraints to ensure stability [30]. Naturally, a longer deadline leads to better schedulablility.
- The fixed timing constraints on the control task implementation are platform independent. Consequently, the I/O behavior, and thus, the QoC behavior of the control application is *time* and *value* deterministic [4].

In the process of executing such a control loop, the control input $u(t)$ is held until the next update (see Fig. 5(b)), i.e.,

[1] The time duration from the start of execution of measure operation and end of execution of actuate operation is known as *sensor-to-actuator delay* or *response time* D of a control task. For the rest of the paper, we will refer to D as *delay* of a control task $T \in \tau_c$.

$$u(t) = Kx(t_k), \quad t_k^r \leqslant t \leqslant t_{k+1}^r. \tag{16}$$

Given the input signal (16) and the delay being $D < h$, the discrete-time system (14) becomes a *sampled-data* system [39],

$$x[k+1] = A_d x[k] + B_0(D)u[k] + B_1(D)u[k-1], \tag{17}$$

where

$$A_d = e^{Ah}, \quad B_0(D) = \int_0^{h-D} e^{At} dt \cdot B, \quad B_1(D) = \int_{h-D}^{h} e^{At} dt \cdot B.$$

Putting (16) in (17), we get the following *closed-loop system*,

$$x[k+1] = A_d x[k] + B_0(D)Kx[k] + B_1(D)Kx[k-1]. \tag{18}$$

In (18), we assume that $u[-1] = 0$ for $k = 0$. Next, we define new system states $z[k] = [x[k-1]\, x[k]]'$ and we obtain,

$$z[k+1] = A_{cl}(h, D)z[k], \tag{19}$$

where

$$A_{cl}(h, D) = \begin{bmatrix} 0 & \Lambda \\ B_1(D)K & A_d + B_0(D)K \end{bmatrix}, \tag{20}$$

where Λ is the unity matrix. Stability of the overall closed-loop system is governed by the properties of $A_{cl}(h, D)$ and for stability, the absolute value of maximum eigenvalue of $A_{cl}(h, D)$ should be less than unity, i.e.,

$$|\lambda_{max}(A_{cl}(h, D))| < 1. \tag{21}$$

The closed-loop system might become unstable when D is long and fails to meet (21). In an ideal implementation, the delay of a control application is $D = 0$. An actual implementation under resource constraints results in $D > 0$ which causes deterioration in the QoC of a control loop. As a QoC measure, we consider *stability margin* [42] of a control loop, i.e.,

$$J(D) = 1 - |\lambda_{max}(A_{cl}(h, D))| \tag{22}$$

as illustrated in Fig. 6. In this work, we assume that the QoC $J(D)$ of a given control system is a monotonically decreasing function of the delay D [27], i.e.,

$$J(D_i) > J(D_j), \quad \forall D_j < D_i \leqslant D_{wc}. \tag{23}$$

The stability margin[2] quantifies how far is the feedback loop from being *unstable*. As a QoC metric, we consider the degradation in QoC due to implementation irregularities (such as $D > 0$) from their performance with an ideal implementation (where $D = 0$). The QoC achieved with an ideal implementation is defined as *nominal* performance J_0, where

$$J_0 = 1 - |\lambda_{max}(A_{cl}(h, 0))|. \tag{24}$$

For a given worst-case delay D_{wc} of a control task, the corresponding QoC is indicated as $J(D_{wc})$ as depicted in the figure. As explained, the delay $D = D_{wc}$ is constant and the resulting QoC is exactly $J(D_{wc})$. Naturally, we aim to reduce D_{wc} by an appropriate priority assignment such that the resulting QoC $J(D_{wc})$ improves. Since our goal is to improve the overall QoC of all control applications (see (2)), the priority assignment highly depends on the relation between D_{wc} and the function $J(D_{wc})$ of all control tasks.

3. Multi-layered schedule synthesis scheme

In this section, now we present our main result. We first formally describe the setting under consideration, followed by the basic scheme of our proposed approach. Next, we outline the details of our algorithm followed by a discussion on the algorithm complexity.

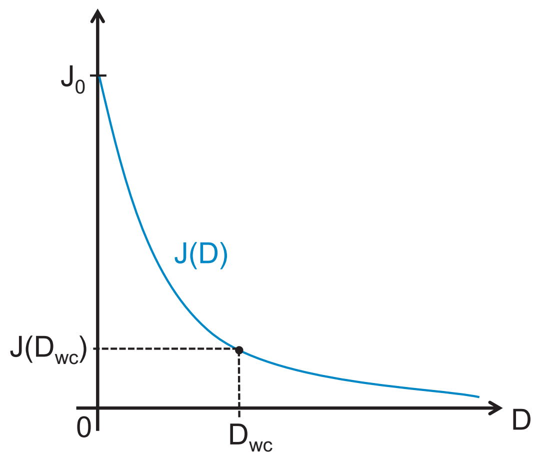

Fig. 6. Quality-of-Control vs. delay.

3.1. System description

We consider an MCCPS where the task set τ is mapped and executed on a single resource r sharing total available service β_r according to FPS. Let $\tau_{rt} \subset \tau$ denote the set of deadline-critical real-time tasks, and $\tau_c \subset \tau$ be the set of QoC-critical control tasks. We assume that all tasks are independent from each other. Further, let the task indices be $i \in I$ with $I = \{1, 2, \ldots, n\}$, $I_{rt} \subset I$ denoting the set of indices related to τ_{rt}, and $I_c \subset I$ indicating the set of indices related to τ_c such that $I_{rt} \cap I_c = \emptyset$. Each task $T_i \in \tau$ is characterized by the tuple $(\alpha_i, e_i, d_i, \pi_i)$ where

- $\alpha_i = (\alpha_i^u, \alpha_i^l)$ denotes a pair of arrival curves according to (5) which triggers the execution of task T_i,
- $e_i = (e_i^u, e_i^l)$ specifies the maximum and the minimum execution demand of T_i, where $e_i^u, e_i^l \in \mathbb{R}^+$,
- $d_i \in \mathbb{R}^+$ represents the relative deadline of $T_i \in \tau_{rt}$. For $T_i \in \tau_c$, d_i specifies the maximum delay at which the control system is still stable, i.e., the maximum D at which inequality (21) is fulfilled,
- $\pi_i \in \Pi$, with $\Pi = \{1, 2, \ldots, n\}$, and π_i denoting the unique priority of $T_i \in \tau$ on resource r. Further, $\pi_i = 1$ represents the highest priority, and $\pi_i = n$ the lowest.

Further, β_r denotes the total available service on resource r (see (7)). The real-time task set τ_{rt} is referred to as *schedulable* iff *all* the tasks $T_i \in \tau_{rt}$ meet their deadlines, i.e., $D_i < d_i, \ \forall i \in I_{rt}$.

On the other hand, the control task set τ_c is schedulable iff *all* the tasks $T_i \in \tau_c$ meet their stability constraints, i.e., $|\lambda_{max}(A_{cl}(h, D_i))| < 1, \ \forall i \in I_c$. However, in contrast to the real-time tasks, it is not sufficient to just meet this constraint but in addition it is necessary to maximize the overall QoC $J^* = \sum_{i \in I_c} J_i$ where J_i is shown in (22).

3.2. Multi-layered scheduling scheme

Before we present the details of our proposed schedule synthesis algorithm we first outline the basic scheme using the example depicted in Fig. 7. As described above, the overall goal is to come up with a priority assignment such that the all real-time tasks $T_i \in \tau_{rt}$ meet their firm deadlines d_i, and at the same time, overall QoC is optimized for the control tasks $T_i \in \tau_c$. For this we present a multi-layered scheduling scheme where tasks of similar criticality are assigned a certain layer in the scheduling engine. This allows

[2] Note that the presented scheme is also applicable to any other QoC-metric e.g., *quadratic error* and *settling time*, that follows the monotonicity property depicted in (23).

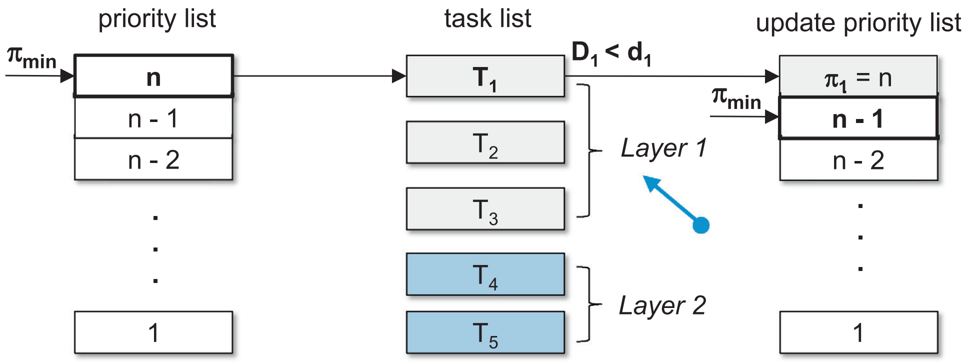

Fig. 7. Multi-layered scheduling scheme, iteration 1.

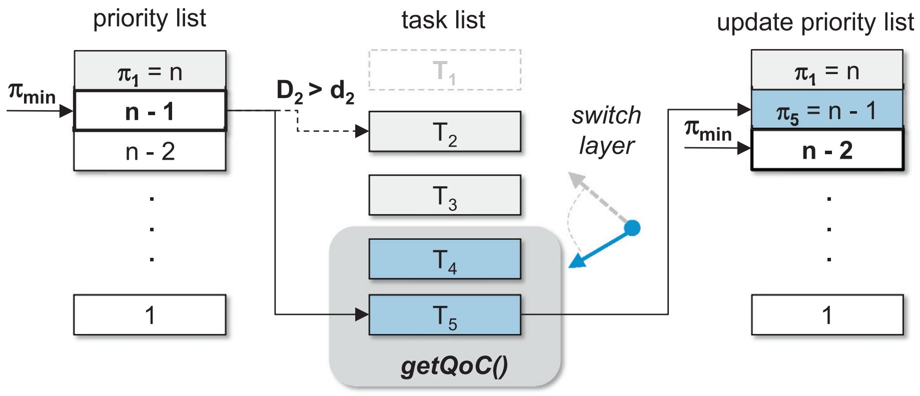

Fig. 8. Multi-layered scheduling scheme, iteration 2.

to implement appropriate scheduling strategies for each type of criticality in an integrated scheduling framework. In the example, we have assigned the real-time task set τ_{rt} the top layer, and the control task set τ_c the bottom layer. Let $\tau_{rt} = \{T_1, T_2, T_3\}$ be the set of real-time tasks and $\tau_c = \{T_4, T_5\}$ be the set of control tasks as illustrated in the figure. Further, let $n = 5$ denote the total number of tasks in the system, the total number of priorities to be assigned, respectively. The available priorities $\pi_i \in \{1, \ldots, n\}$ are indicated by a *priority list*, where the minimum available priority is denoted by π_{min}, i.e., the lowest priority which has not been assigned to any task yet. Similarly, the *task list* represents all the tasks $T_i \in \tau$ that have not been assigned any priorities. Note that the task list basically consists of two lists representing the different layers:

- *Layer 1:* Contains the set of real-time tasks τ_{rt} which is sorted according to deadlines d_i in descending order, i.e., the first task in the list is the task with the longest deadline. Note that tasks $T_i \in \tau_{rt}$ are assigned priorities with the goal to guarantee the deadlines, i.e., $D_i \leqslant d_i$.
- *Layer 2:* Contains the set of unsorted control tasks τ_c. Tasks $T_i \in \tau_c$ are assigned priorities with the goal to guarantee stability as per (21) and to maximize overall QoC J^* according to (2).

Let us look at the example in Fig. 7. We start assigning the minimum available system priority $\pi_{min} = n$ to the first task $T_1 \in \tau_{rt}$ of *Layer 1*, i.e., $\pi_1 = \pi_{min}$ (here $\pi_{min} = 5$), and check whether $D_1 \leqslant d_1$ holds. In the example $D_1 < d_1$, and hence, T_1 meets its deadline if assigned π_{min}. Consequently, we update the priority and task list for the next iteration as illustrated in Fig. 7. This requires (i) fixing the priority assignment $\pi_1 = n$, (ii) decrementing the minimum available system priority $\pi_{min} = n - 1$ for the next iteration, and (iii) removing T_1 from the task list. We repeat this procedure for every task in *Layer 1* until the schedulability test fails. For instance, the worst-case delay experienced by T_2 exceeds the deadline as $D_2 > d_2$ for $\pi_2 = \pi_{min}$. Note that T_2 is currently the real-time task with the longest deadline in the task list. As a result, none of the other remaining real-time tasks will meet their deadlines while being assigned π_{min} as their deadlines are equal or tighter due to the sorting of the task list. Hence, we need not to check the other real-time tasks for schedulablility, and we *switch* to *Layer 2* where we now pursue a QoC-oriented priority assignment strategy. For this we evaluate the QoC (see function *getQoC ()*) in Fig. 8. In particular, we assign π_{min} to the control tasks $T_i \in \tau_c$ for which overall QoC is optimized, e.g., T_5. Subsequently, the priority and task list are updated again, i.e., $\pi_5 = n - 1$ is locked, $\pi_{min} = n - 2$, and T_5 is removed from the task list. This procedure is repeated until all tasks $T_i \in \tau$ have been successfully assigned priorities.

3.3. QoC optimization

As described above the function *getQoC ()* determines which control task is assigned π_{min}. Towards this, we introduce three different QoC-objectives $\mathcal{P}_i$ that can be selected as an optimization objective. Essentially, π_{min} is assigned the control task $T_i \in \tau_c$ for which $\mathcal{P}_i$ is minimum. We explain the QoC-objectives in detail in what follows.

QoC deviation: We define the *QoC deviation* $\mathcal{P}_1$ as

$$\mathcal{P}_1 = \frac{J_0 - J(D_{wc})}{J_0}. \tag{25}$$

Clearly, $\mathcal{P}_1$ captures normalized deviation of QoC due to non-zero delay $D = D_{wc}$ from its nominal value $J_0 = J(0)$. In essence, $\mathcal{P}_1$ measures the deviation of a control task's QoC to the delay. Thus, a higher $\mathcal{P}_1$ implies that the potential improvement in terms of QoC will be more significant in the case of $D_{wc} \to 0$. Consequently, a control task with higher $\mathcal{P}_1$ should be assigned a relatively high priority so that it contributes more to the overall QoC (2). Let us consider Fig. 9(a) which illustrates the QoC functions $J_4(D)$ and $J_5(D)$ of the control tasks of our example in Fig. 8. Let $J_4(0) = 0.8$ and $J_5(0) = 0.4$, and D_{wc} be the worst-case delay experienced by any of the two control tasks for the priority assignments π_{min}. Further, assume $J_4(D_{wc}) = 0.3$ and $J_5(D_{wc}) = 0.25$. According to (25) $\mathcal{P}_1(J_4) = 0.625$ and $\mathcal{P}_1(J_5) = 0.375$, and hence T_5 will be assigned $\pi_5 = \pi_{min}$ as depicted in Fig. 8. Consequently, corresponding to the presented schedule synthesis scheme, T_4 will certainly be assigned a priority $\pi_4 > \pi_5$, i.e., the worst-case delay D'_{wc} experienced by T_4 will certainly be smaller than D_{wc}. This is also depicted in Fig. 9(b) where the improvement in QoC is significantly higher for T_4 ($\Delta J_4 > \Delta J_5$) than for T_5. Note that the actual improvement in QoC depends on the behavior of function $J(D)$, which is not reflected in $\mathcal{P}_1$.

Integral of QoC deviation: The *integral of QoC deviation* is defined as

$$\mathcal{P}_2 = \sum_{D=0}^{D=D_{wc}} \left(\frac{J_0 - J(D)}{J_0} \right). \tag{26}$$

$\mathcal{P}_2$ captures the nature of QoC deviation over the range of possible delays $0 \leqslant D \leqslant D_{wc}$, and thus captures more details of $J(D)$ compared to $\mathcal{P}_1$. For instance, in Fig. 10(a), two QoC curves $J_4(D)$ and $J_5(D)$ have similar $\mathcal{P}_1$ at $D = D_{wc}$, but $J_4(D)$ (see hatched area) will have higher $\mathcal{P}_2$ compared to $J_5(D)$. As a result, T_5 will be assigned $\pi_5 = \pi_{min}$, and consequently T_4 will be assigned a priority $\pi_4 > \pi_5$. Clearly, for the same amount of reduction in D_{wc} the corresponding control task of $J_4(D)$ will contribute more to the overall QoC (2). Thus, the control tasks with higher $\mathcal{P}_2$ should be assigned relatively high priority. In the example, for $D = D_{wc}$, minimizing $\mathcal{P}_1$, $\mathcal{P}_2$, respectively, results in the same priority assignment $\pi_4 > \pi_5$ for T_4 and T_5.

However, as depicted in Fig. 10(a), it can be seen that, e.g., for $D = D'_{wc}$, $\mathcal{P}_1(J_4) = \mathcal{P}_1(J_5)$, whereas the *integral of QoC deviation* $\mathcal{P}_2$ is more expressive than $\mathcal{P}_1$, and clearly $\mathcal{P}_2(J_4) > \mathcal{P}_2(J_5)$.

QoC gradient: The *QoC gradient* is defined as

$$\mathcal{P}_3 = \left(\frac{d}{dD} \frac{J_0 - J(D)}{J_0} |_{D=D_{wc}} \right)^{-1}. \tag{27}$$

$\mathcal{P}_3$ indicates the sensitivity of a control task's QoC to the delay at $D = D_{wc}$ which is the operating point for the control task model under consideration (in Section 2.2). A control task with higher $\mathcal{P}_3$ implies that a small reduction in D_{wc} might results in higher benefit in terms of QoC as depicted in Fig. 10(b) for D_{wc}. Intuitively, assigning a relatively high priority to the control tasks with higher $\mathcal{P}_3$ has more significant impact on the overall performance (2). Similarly to the *integral of QoC deviation* $\mathcal{P}_2$, the *QoC gradient* $\mathcal{P}_3$ strongly depends on the exact shape of $J(D)$ and the operation point D_{wc}.

3.4. Schedule synthesis algorithm

In this section we present the detailed algorithm of our proposed schedule synthesis scheme as illustrated in Algorithm 1. As inputs, the algorithm requires (i) the task set τ which is mapped on resource r, where each task $T_i \in \tau$ is characterized by the tuple $(\alpha_i, e_i, d_i, \pi_i)$ (ii) the service curve β_r which models the processing capacity on r, and (iii) $\mathcal{P}_i$ which specifies the QoC-oriented metric that is used to determine the priority assignment of the control tasks.

Algorithm 1. Priority assignment algorithm

```
Require $\tau$, $\beta_r$, $\mathcal{P}_i$
1: sort $\tau_{rt}$ according to deadlines
2: $\pi_{min} = n$
3: $\pi_i = 0$, $\forall i \in \{1, \dots n\}$
4: While $\pi_{min} > 0$ do
5:   for all $\{i \in I_{rt} | \pi_i = 0\}$ do
6:     $\pi_i = \pi_{min}$
7:     $D_i = computeDelay()$
8:     if $(D_i \leqslant d_i)$ then
9:       $T_i$ is assigned priority $\pi_i = \pi_{min}$
10:      $\pi_{min} = \pi_{min} - 1$
11:    else
12:      $T_i$ is assigned invalid priority $\pi_i = 0$
13:      if $\{\exists j \in I_c | \pi_j = 0\}$ then
14:        if $(selectControlTask() == stable)$ then
15:          $\pi_{min} = \pi_{min} - 1$
16:        else
17:          return (not schedulable)
18:        end if
19:      else
20:        return (not schedulable)
21:      end if
22:    end if
23: end for
24: if$\{\forall i \in I_{rt} | \pi_i \neq 0\}$then
25:   if $(selectControlTask() == stable)$ then
26:     $\pi_{min} = \pi_{min} - 1$
27:   else
28:     return (not schedulable)
29:   end if
30: end if
31: end while
32: return (schedulable)
```

Before the algorithm starts, we perform an initialization which involves:

- Sorting the real-time task set τ_{rt} according to deadlines d_i in descending order (line 1).
- Initialization of π_{min} with the lowest available priority $n = |\tau|$ (line 2).
- Initialization of all task priorities $\pi_i = 0$, $\forall i \in I$ (line 3). $T_i \in \tau$ with priorities $\pi_i = 0$ indicate unscheduled tasks which are member of the task list whereas tasks with $\pi_i = \{1, \dots, n\}$ indicate tasks which have already been successfully assigned feasible priorities, and hence do not belong to the task list.

As long as there exist unassigned priorities (line 4) we iterate over all real-time tasks in the task list, i.e., $T_i \in \tau_{rt}$, $\forall i \in I_{rt}$ such that $\pi_i = 0$ (line 5). We start assigning the first task in the list $T_i \in \tau_{rt}$ the actual minimum priority $\pi_i = \pi_{min}$ (line 6). Next, corresponding to π_i, we compute the worst-case delay D_i calling the function *computeDelay ()* (line 7) using the RTC framework. This involves the two steps outlined in Algorithm 2:

- Computation of the lower remaining service $\beta^l_{r,i}$ for T_i. This is achieved using the relation defined in (12). From the total available service β_r we deduct the execution demands e_j of all event streams α_j of the tasks $T_j \in \tau$ with higher priorities than T_i's, i.e., tasks for which $\pi_j < \pi_i$ (lines 1–5). Recall that $\pi_i = \pi_{min}$, and the priorities of all unscheduled tasks have been initialized with $\pi_i = 0$. In other words, all tasks T_j that have not been assigned

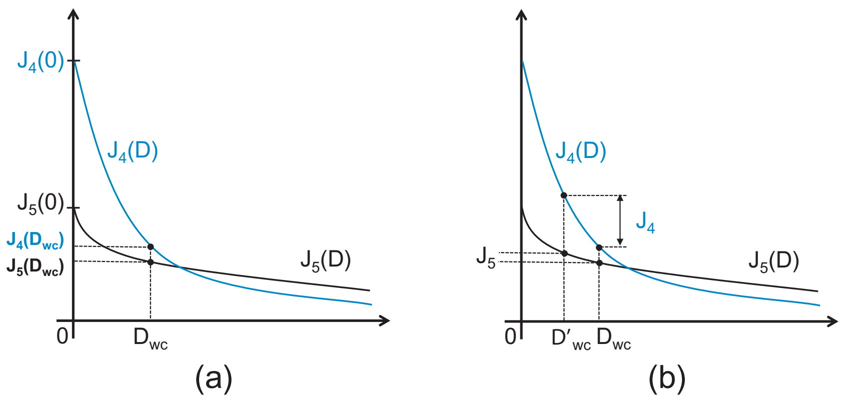

Fig. 9. QoC deviation.

any priorities, i.e., $\pi_j = 0$, are considered as tasks having higher priorities than T_i.

- Computation of the worst-case delay D_i experienced by the event stream α_i triggering T_i (line 6). For this purpose, we compute $D_i = \text{hdist}(\alpha_i^u, \beta_{r,i}^l)$ as defined in (4).

Algorithm 2. *computeDelay ()*

```
1: $\beta^{l\prime} = \beta_r^{l\prime}$
2: for all $\{j \in I | \pi_j < \pi_i\}$ do
3:     $\beta^{l\prime} = \left(\beta^{l\prime} - \alpha_j^u\right) \overline{\otimes}\, 0$
4: end for
5: $\beta_{r,i}^l = \beta^{l\prime}$
6: $D_i = \text{hdist}(\alpha_i^u, \beta_{r,i}^l)$
7: return $(D_i)$
```

Algorithm 3. *selectControlTask ()*

```
1: $stable = false$
2: for all $\{i \in I_c | \pi_i = 0\}$ do
3:     $\pi_i = \pi_{min}$
4:     $D_i = computeDelay()$
5:     if $(|\lambda_{max}(A_{cl}(h_i, D_i))| < 1)$ then
6:         $stable = true$
7:         $\mathcal{P}_i = getQoC()$
8:     end if
9: end for
10: if$(stable == true)$ then
11:     for all $\{i \in I_c | (|\lambda_{max}(A_{cl}(h_i, D_i))| < 1)\}$ do
12:         if $(\mathcal{P}_i = \min_i \mathcal{P}_i)$ then
13:             $\pi_i = \pi_{min}$
14:         else
15:             $\pi_i = 0$
16:         end if
17:     end for
18:     return (stable)
19: else
20:     return (unstable)
21: end if
```

Next, we check if D_i respects the deadline of $T_i \in \tau_{rt}$, and in case $D_i \leqslant d_i$ (line 8) we lock the priority assignment $\pi_i = \pi_{min}$ (line 9), and π_{min} is decremented for the next iteration (line 10). In case the deadline is violated, we reset the priority assignment $\pi_i = 0$ (line 12) because T_i requires a higher priority to guarantee its deadline. Further, in case there exists at least one unscheduled control task (line 13) we call the function *selectControlTask ()* (line 14) which is responsible for the priority assignment to the control tasks $T_i \in \tau_c$. In case no control task is available the task set τ is not schedulable (line 20). Function *selectControlTask ()* involves the steps outlined in Algorithm 3:

- Computation of worst-case delay D_i (line 4) (see Algorithm 2), evaluation of the stability condition in (21) (line 5), and computation of the *QoC* (line 7) according to the implemented QoC objectives in (25)–(27) selected in function *getQoC ()*.
- Check if at least one control task is stable (line 10) and return *unstable* otherwise. Note that if Algorithm 3 returns *unstable*, Algorithm 1 declares the entire task set τ as *unschedulable* (lines 16 and 28) and the algorithm stops. Next, the task with minimum $\mathcal{P}_i$ (lines 11–17) is assigned π_{min}. Further, if there are two tasks with exactly same minimum QoC, then π_{min} is assigned to one of them arbitrarily.

In case a feasible priority assignment was found for a control task (line 13), π_{min} is decremented for the next iteration (line 15). Note that lines 24–30 again represent the priority assignment algorithm for the control tasks in case all real-time tasks already have been scheduled and there are only control tasks left.

3.5. Complexity analysis

In this section, we are concerned with analyzing the complexity of our proposed schedule synthesis algorithm. As discussed previously, it is meaningful to assign priorities to real-time tasks in the system according to DM. For this purpose, the set of real-time tasks τ_{rt} needs to be sorted in order of decreasing deadlines (see Line 1 in Algorithm 1). Recall that our schedule synthesis algorithm starts assigning priorities from the lowest to the highest. This way, if real-time tasks are schedulable at lower priority levels, the algorithm can use higher priority levels to accommodate control tasks and, hence, improve QoC. Such sorting can be performed in $\mathcal{O}(|\tau_{rt}| \log |\tau_{rt}|)$ time where $|\tau_{rt}|$ represents the number of elements in τ_{rt}. Further, our proposed algorithm outlined in Algorithm 1 has to assign n different priorities, i.e., for each priority level, there will be either one real-time or one control task – multiple tasks per priority level are not allowed in our setting. For each priority level, there is one iteration of the while-loop (line 4). Hence, this loop is executed n times starting by $\pi_{min} = n$ (see Line 2) until $\pi_{min} = 0$ is reached (see Line 4).

In each iteration of the while-loop, the algorithm tries to accommodate a real-time task (from the sorted task set τ_{rt}) in the current priority level given by π_{min} (see for-loop, Line 5). For

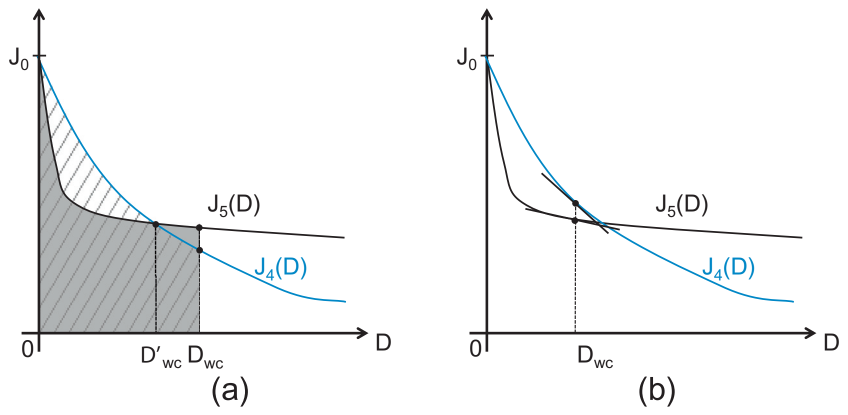

Fig. 10. Integral of QoC deviation and QoC gradient.

this, the delay incurred by the real-time task at the current π_{min} needs to be computed (see *computeDelay ()*, Line 7). The function *computeDelay ()* (shown in Algorithm 2) computes the *minimum* service $\beta^l_{r,i}$ which results from subtracting the arrival curves α^u_j of higher-priority events from the available service. It has been proven that if a deadline is missed by a given schedule, this always happens within the so-called *busy period*, i.e., within the time interval in which the resource is busy without idle time [40]. As a consequence, the computation of $\beta^l_{r,i}$ can be limited to the busy period on the resource. Hence, *computeDelay ()*'s complexity is pseudo-polynomial in the form $\mathcal{O}(n \times Z)$, where n is the total number of tasks in the system and Z is the length of the *busy period* on the shared resource [41].

The function *selectControlTask ()* (shown in Algorithm 3) calls *computeDelay ()*. Since *selectControlTask ()* iterates over the set of control tasks, its complexity is $\mathcal{O}(|\tau_c| \times n \times Z) < \mathcal{O}(n^2 \times Z)$, where $|\tau_c|$ denotes the number of tasks in τ_c. Note that the function *getQoC ()* in Algorithm 3 has constant complexity and hence does not influence *selectControlTask ()*'s complexity.

The for-loop's body in Algorithm 1 is executed at most n times, once for each priority level. Note that, if a real-time task is not be schedulable for a given priority level π_{min}, it will require another iteration of the for-loop. However, the number of iterations in the for-loop is upper-bounded by n. The complexity of executing the for-loop in Algorithm 1 is given by $\mathcal{O}(n \times (n \times Z + n^2 \times Z)$ which can be expressed as $\mathcal{O}(n^3 \times Z)$, i.e., also pseudo-polynomial as a result of the complexity of *computeDelay ()*. Finally, since the while-loop's body is also executed n times, the overall complexity of our proposed algorithm is pseudo-polynomial in the form $\mathcal{O}(n^4 \times Z)$.

4. Experimental results

In this section, we show a number of experimental results (i) using an illustrative task set, and (ii) by performing a sensitivity analysis based on task sets for different utilizations.

4.1. Illustrative example

Let us consider a task set of the form $\tau = \tau_{rt} \cup \tau_c$ that consists of $n = 10$ tasks $T_i \in \tau$. Further, let $T_i \in \tau_{rt}, \ \forall i \in I_{rt}$ with $I_{rt} = \{1,2,3,4,5,6\}$, and $T_i \in \tau_c, \ \forall i \in I_c$ with $I_c = \{7,8,9,10\}$. Each task T_i is characterized by the tuple (α_i, e_i, d_i) where

- α_i denotes T_i's triggering pattern, defined by the tuple (p_i, j_i) with period p_i, and jitter j_i,
- e_i describes the tuple (e^u_i, e^l_i) indicating the maximum and minimum execution time of T_i,
- d_i is the deadline of $T_i \in \tau_{rt}$, the longest delay (in brackets of Table 1) by which the system controlled by $T_i \in \tau_c$ is still stable, respectively.

In order to demonstrate our scheme, we consider an illustrative example with the task set specification as depicted in Table 1. The task parameters and sampling periods of the control tasks are typical values as found in automotive settings. Further, let the task set τ according to Table 1 be mapped on a single processing resource r with a total initial linear service β_r. Each control task $T_i \in \tau_c$ is responsible for regulating the behavior of a dynamical system. In order to ensure the generality of our results, instead of choosing a specific plant model from the automotive domain, we consider a set of inverted pendulums with different parameters as systems to be controlled by the control tasks. Inverted pendulum is a fundamental benchmark in control theory [43] and a classical representative example that relates to many real-life control systems. This is mainly because its plant is naturally unstable with fast and nonlinear dynamics. As a result, the control design imposes typical stability and performance requirements such as settling-time, steady-state error and robustness constraints which are also relevant for many automotive control systems. The linearized version of its continuous-time model has the following parameters:

$$A = \begin{bmatrix} 0 & 1 \\ \frac{9.8}{l} & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{1}{m \times l} \end{bmatrix}. \tag{28}$$

For the sake of illustration, we have chosen different values of m and l in our case study. Towards this, we have designed four controllers as per (16) using a pole-placement technique.

Observations and discussion: We studied and compared the results of four different scheduling strategies which are depicted in Table 2. The table shows the priority assignments $\pi_i(X) = \{\pi_1, \pi_2, \ldots, \pi_{10}\}$ with $X = \{DM, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$, and $\pi_i = 1$ denotes highest priority and $\pi_i = 10$ the lowest. The overall QoC for each of the scheduling strategies is $J^* = \sum_{i \in I_c} J_i(D_i)$. As a reference, we also compute the *nominal* overall QoC $J^*_{nom} = \sum_{i \in I_c} J_i(0) = 1.3$ indicating the maximum overall QoC that can be achieved on an ideal implementation platform with zero delays, i.e., $D = 0$ for all the control tasks. For the sake of comparison, we evaluate for each scheduling strategy, how far is the actual overall QoC (J^*) from the nominal overall QoC (J^*_{nom}) where J^*_{nom} represents a safe upper bound on the maximum realizable QoC. Consequently, we may use J^*/J^*_{nom} as a measure to estimate the quality of the derived results. Note that the computation of the exact optimal solution requires to evaluate $n!$ (i.e., factorial of n) priority assignments. That is, for $n = 10$ we have 3,628,800 combinations to be evaluated, which is not feasible. Finally, we also show typical run times $t(Alg)$ of the different algorithms which have been implemented in Matlab, and were executed on a dual core 1.8 GHz processor with 3 GB RAM. We discuss the results of Table 2 in detail in what follows.

- The column for $\pi_i(DM)$ shows the classical deadline monotonic priority assignment for all tasks $T_i \in \tau$. In this case, the overall QoC has been computed as $J^*(DM) = 0.301137$ which clearly shows the worst QoC compared to the other approaches.As discussed before, if there exists a feasible schedule under fixed priorities, the DM schedule is also feasible. However, it does not allow optimizing the control performance. Under DM, all the control tasks $T_i \in \tau_c$ have been assigned lower priorities $\pi_i(DM) = \{5, 9, 8, 7\},\ \forall i \in I_c$, compared to $\pi_i(\mathcal{P}_1) = \{1, 9, 6, 4\}$. As a result, delays experienced by the control tasks may be high as long as their deadlines are met, and hence, DM is not be very efficient for optimizing overall QoC as $J^*/J^*_{nom} = 23.2\%$.
- The last three columns $\pi_i(\mathcal{P}_1),\ \pi_i(\mathcal{P}_2)$, and $\pi_i(\mathcal{P}_3)$ show results for each of the QoC-oriented metrics implemented in the multi-layered scheduling scheme, i.e., (i)*QoC deviation* $\mathcal{P}_1$ (ii) *integral of QoC deviation* $\mathcal{P}_2$, and (iii) *QoC gradient* $\mathcal{P}_3$, as defined in (25)–(27), respectively. The desired QoC metric is configured and evaluated using the function *getQoC()* of Algorithm 3 which selects the control task with minimum $\mathcal{P}_1,\ \mathcal{P}_2,\ \mathcal{P}_3$, respectively. It can be observed that the priority assignments $\pi_i(\mathcal{P}_1)$ and $\pi_i(\mathcal{P}_2)$ are equal for all tasks $T_i \in \tau$, and achieve a QoC $J^*(\mathcal{P}_1) = 1.173276$ ($J^*(\mathcal{P}_2)$ respectively) which is 90.2% of the nominal value J^*_{nom}.On the other hand, $\pi_i(\mathcal{P}_3)$ results in a lower QoC of $J^*(\mathcal{P}_3) = 0.739011$,$J^*/J^*_{nom} = 56.8\%$. The difference compared to $\pi_i(\mathcal{P}_1)$ and $\pi_i(\mathcal{P}_2)$ is reflected in the priority assignment for the control tasks that is $\pi_i(\mathcal{P}_1) = \{1, 9, 6, 4\}$ compared to $\pi_i(\mathcal{P}_3) = \{4, 9, 1, 6\}, \forall i \in I_c$. This demonstrates that the control tasks exhibit different sensitivity to delay resulting in different QoC depending on the exact priority assignment.

Table 3 shows the delays D_i for each of the control tasks $T_7, T_8, T_9, T_{10} \in \tau_c$ that have been computed for the scheduling strategies under consideration.

4.2. Control task implementation

As discussed in Section 2.2 our scheme employs a time-driven control task model where the time interval between the start of *measure* and end of *actuate* operation is of fixed length D. The value of D is computed by our scheduling framework for every control task (see Algorithm 3, line 4 in Section 3.4), and hence, provides the timing parameters that are necessary to implement the time-driven control task model.

We developed a run-time software framework (see Fig. 11) for integrated *control/plant co-simulation* enabling a time-triggered implementation of the task model described in Section 2.2. Each control task $T_i \in \tau_c$ is defined by the tuple (p_i, π_i, D_i), where p_i is the period, π_i denotes the priority, and D_i is the delay as depicted in Table 3. Our framework runs a scheduler at a fixed base period T_{base} which imposes the following:

- $D_i = \lambda_i \cdot T_{base}$ with $\lambda_i \in \mathbb{Z}^+$ denoting the *release prescaler*,
- $p_i = \sigma_i \cdot T_{base}$ with $\sigma_i \in \mathbb{Z}^+ \leqslant \lambda_i$ denoting the *delay prescaler*.

An example for $T_1 : \lambda_1 = 2,\ \sigma_1 = 1$, and $T_2 : \lambda_2 = 4,\ \sigma_2 = 2$ is illustrated in Fig. 12. When multiple jobs are triggered, the one with the highest priority is executed first. Using our framework, we implemented the four control tasks for the *QoC deviation* $\mathcal{P}_1$ and DM with $T_{base} = 1ms$. The delay values $D_i(DM)$ and $D_i(\mathcal{P}_1)$ for the control tasks have been determined based on the task set of Table 1 and the corresponding priority assignment in Table 2, and are depicted in Table 3. Since $T_{base} = 1ms$, we have

$$\hat{D}_i(DM) = \left\lceil \frac{D_i(DM)}{T_{base}} \right\rceil \quad \text{and} \quad \hat{D}_i(\mathcal{P}_1) = \left\lceil \frac{D_i(\mathcal{P}_1)}{T_{base}} \right\rceil.$$

With the plant model as depicted in (28), we have two system states $x_0(t)$ and $x_1(t)$, and one control input $u(t)$. Fig. 13 and Fig. 14 show the state trajectories and input signals that we obtained from our simulation for the four control applications $T_i \in \tau_c$. Fig. 13 shows the results for the control tasks scheduled according to DM. As all the control deadlines are met all four systems are stable, i.e., the control inputs converge towards zero with increasing time. However, it can be noticed that the plants converge much faster with the control task implementation according to $\mathcal{P}_1$ as depicted in Fig. 14. This is because the delays for the control task implementations are smaller for $T_7,\ T_9$ and T_{10}, and the overall QoC $J^*(\mathcal{P}_1) > J^*(\mathcal{DM})$ (see Table 2) which is reflected in the faster convergence for $\mathcal{P}_1$. Note that the delay for T_8 is same for DM and $\mathcal{P}_1$ which is reflected in the same trajectory as depicted in the figures. Finally, this experiment demonstrates (i) the relation between *schedulability* and *stability*, and (ii) the QoC behavior which is reflected in the convergence of the control inputs.

4.3. Sensitivity analysis

The task set depicted in Table 1 is an illustrative example to demonstrate the applicability of our proposed scheme. We showed that all three presented heuristics clearly outperform the classical DM approach. However, the results for the above task set may not be generalized as different task parameters such as periods, execution times and deadlines may clearly influence the results of the applied priority assignment scheme. To account for this, we next study the sensitivity of the presented metrics to variations in the real-time task parameters and compare the results with the DM approach. Note that we first keep the configuration of the control

Table 1
Task set specification.

T_i	$\alpha := (p_i, j_i)$ (ms)	$e_i := (e_i^u, e_i^l)$ (ms)	d_i (ms)
$T_1 \in \tau_{rt}$	(50,20)	(1.5 0.2)	50
$T_2 \in \tau_{rt}$	(20,15)	(1.3 0.2)	20
$T_3 \in \tau_{rt}$	(10,30)	(0.5 0.2)	15
$T_4 \in \tau_{rt}$	(20,25)	(0.7 0.3)	12
$T_5 \in \tau_{rt}$	(10,15)	(8.0 0.5)	8
$T_6 \in \tau_{rt}$	(10,25)	(1.2 0.1)	5
$T_7 \in \tau_c$	(30,0)	(1.8 0.2)	(15)
$T_8 \in \tau_c$	(40,0)	(1.2 0.3)	(32)
$T_9 \in \tau_c$	(40,0)	(1.7 0.1)	(27)
$T_{10} \in \tau_c$	(30,0)	(1.0 0.2)	(21)

Table 2
Results for different scheduling strategies.

T_i	$\pi_i(DM)$	$\pi_i(\mathcal{P}_1)$	$\pi_i(\mathcal{P}_2)$	$\pi_i(\mathcal{P}_3)$
$T_1 \in \tau_{rt}$	10	10	10	10
$T_2 \in \tau_{rt}$	6	8	8	8
$T_3 \in \tau_{rt}$	4	7	7	7
$T_4 \in \tau_{rt}$	3	5	5	5
$T_5 \in \tau_{rt}$	2	3	3	3
$T_6 \in \tau_{rt}$	1	2	2	2
$T_7 \in \tau_c$	5	1	1	4
$T_8 \in \tau_c$	9	9	9	9
$T_9 \in \tau_c$	8	6	6	1
$T_{10} \in \tau_c$	7	4	4	6
J^*	0.301137	1.173276	1.173276	0.739011
J^*/J^*_{nom}	23.2%	90.2%	90.2%	56.8%
$t(Alg)$	2.7 s	4.9 s	9.4 s	13.9 s

Table 3
Execution delays D for different control task implementations.

T_i	$D_i(DM)$ (ms)	$D_i(\mathcal{P}_1)$ (ms)	$D_i(\mathcal{P}_2)$ (ms)	$D_i(\mathcal{P}_3)$ (ms)
$T_7 \in \tau_c$	12.9	1.8	1.8	10.7
$T_8 \in \tau_c$	22.6	22.6	22.6	22.6
$T_9 \in \tau_c$	21.4	13.1	13.1	1.7
$T_{10} \in \tau_c$	19.2	10.0	10.0	13.1

tasks according to Table 1 and we also use the same controllers C_1, C_2, C_3, C_4. For our sensitivity analysis, we generated a set of $N = 30$ real-time task sets $\tau_{rt} \in \Gamma_U$, where $|\Gamma_U| = N$, for different resource utilizations U where $U = \sum_{i \in I_{rt}} \frac{e_i}{p_i}$. We consider a selected discrete range of resource utilizations $0.3 \leqslant U \leqslant 0.9$ such that $\Gamma_U = \{\tau_{rt,1}, \tau_{rt,2}, \ldots, \tau_{rt,30}\}$ for all U. Each task set τ consists of $n = 10$ tasks, where $|\tau_{rt}| = 6$, and $|\tau_c| = 4$. More specifically, for each real-time task set $\tau_{rt} \in \Gamma_u$ we randomly determine the task parameters period $p_i \in \{10, 20, \ldots, 100\}$, deadline $d_i = p_i$, and execution time e_i that lead to a resource utilization U. We again consider the control task set $\tau_c = \{T_7, T_8, T_9, T_{10}\}$ as defined in Table 1. For the sake of simplicity, we assume jitter to be zero ($j = 0$). For each utilization U we computed the average control performance $\bar{J}^* = \frac{1}{N}\sum_{i=1}^{N} J_i^*$, and plotted $\bar{J}^*$ against U as depicted in Fig. 15. For this, among all generated task sets, only feasible task sets were considered. For the sake of comparison, we perform the priority assignment for each of the generated task sets with the DM approach, and the proposed multi-layered scheduling scheme using the QoC-metrics $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$. The figure shows that up to a utilization of $U = 0.6$ all three heuristics $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$ clearly outperform the DM approach. $\mathcal{P}_1$ and $\mathcal{P}_2$ clearly show the best overall performance. $\mathcal{P}_3$ results in a lower performance compared to $\mathcal{P}_1$ and $\mathcal{P}_2$, but still outperforms the DM approach for small utilizations $U < 0.6$. However, $\bar{J}^*(\mathcal{P}_3)$ significantly deteriorates, and even lies below DM with increasing utilization for $U > 0.7$. Note that $\bar{J}^*(\mathcal{P}_3)$ is not strictly monotonically decreasing. For instance, at $U = 0.65$, $\bar{J}^*(\mathcal{P}_3)$ starts increasing until $U = 0.73$, and subsequently $\bar{J}^*(\mathcal{P}_3)$ heavily deteriorates again. This fluctuating behavior is because the *QoC gradient* $\mathcal{P}_3$ strongly depends on the characteristics of the QoC function $J(D)$ and the operation point $D = D_{wc}$ at which $J(D)$ is evaluated. Hence, $\mathcal{P}_3$ turns out to be not a very reliable optimization subject for QoC optimization. Further, it is interesting to observe that DM shows a similar performance at high utilizations, e.g., for $U = 0.9$, compared to $\mathcal{P}_1$ and $\mathcal{P}_2$. This is because the number of schedulable task sets significantly decreases at high utilizations. Hence, there is less scope for optimizing the overall control performance. Since DM always finds a feasible solution, i.e., one that allows meeting all deadlines, if there exists one, this is most likely the one with the best possible performance. Here, the QoC heavily depends on the control tasks' deadlines.

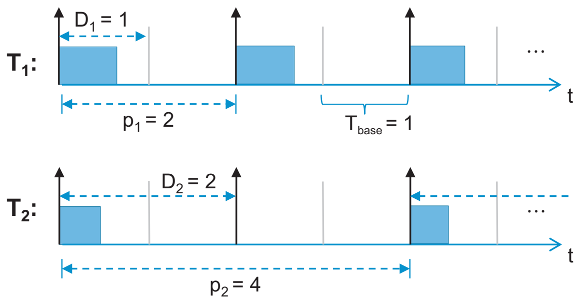

Fig. 12. Examples of time-triggered control task implementations.

4.4. Schedulability vs. QoC

In this experiment, we investigate the performance of our proposed *multi-layered* scheduling scheme with respect to schedulability and QoC optimization. For this purpose, we compare the results of $\mathcal{P}_1$ which turned out to perform well for our control task specification (see Fig. 15) with (i) the DM approach which is known to be optimal with respect to schedulability, and (ii) a *bandwidth reservation* (BR) scheme that realizes optimal overall QoC J^*. Next, we explain the BR approach in more detail.

The idea of the BR scheme is to reserve the *high-priority bandwidth* for the control tasks and the remaining bandwidth for the real-time tasks. That is, we assign (i) the highest available priorities to the control tasks such that they experience reduced delays and achieve a maximum QoC J^*, and (ii) the remaining lower priorities to the real-time tasks. Consider a task set $\tau = \tau_{rt} \cup \tau_c$ consisting of n tasks. Then, the optimal priority assignment (with respect to QoC) is the one which assigns the set of *high priorities* $\Pi_c = \{1, \ldots, |\tau_c|\}$ to the control tasks $T_i \in \tau_c$, and the remaining (lower) priorities $\Pi_{rt} = \{|\tau_c| + 1, \ldots, n\}$ to the real-time tasks $T_i \in \tau_{rt}$ such that the mapping $\Pi_c \mapsto \tau_c$ results in maximum QoC J^*. Naturally, the mapping $\Pi_c \mapsto \tau_c$ depends on the delay sensitivity of the control tasks, i.e., on the characteristics of $J(D_i)$, and the actual delays D_i experienced by the tasks. Note that for computing the delays D_i we do not need to consider interference from the real-time tasks as all tasks in τ_{rt} have lower priorities. Hence, the number of possible choices for assigning priorities to the control tasks is reduced to $|\tau_c|!$, i.e., factorial of $|\tau_c|$. In our example, $|\tau_c| = 4$, and hence, the optimal priority assignment $\Pi_c \mapsto \tau_c$ may easily be obtained by combinatorial search. In order to achieve best possible schedulability of the real-time task set τ_{rt},

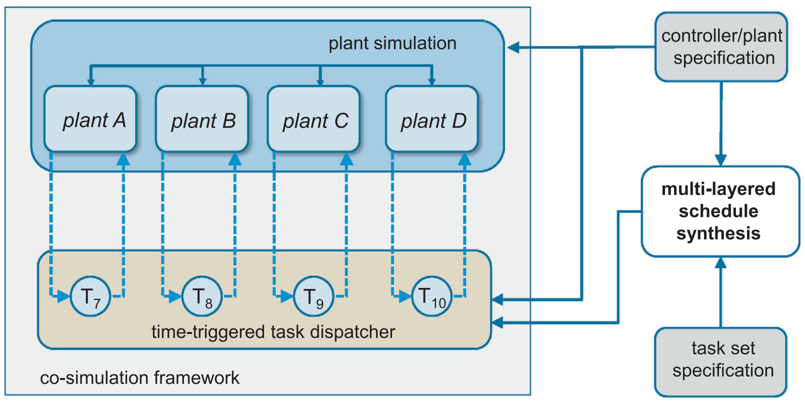

Fig. 11. Overview of experimental setup.

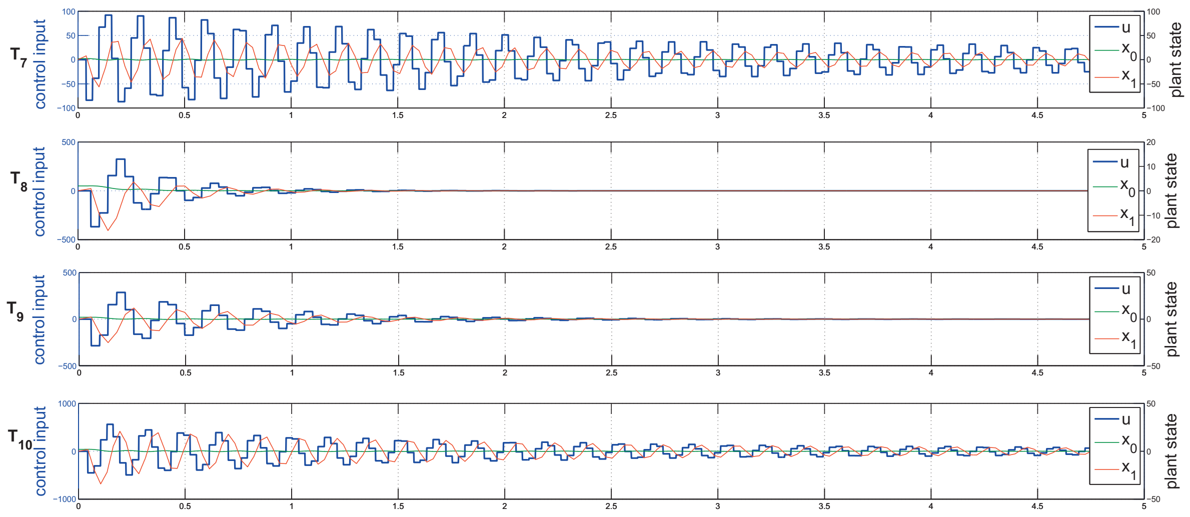

Fig. 13. Observed plants regulated by the control tasks $T_7, T_8, T_9, T_{10} \in \tau_c$ scheduled according to DM.

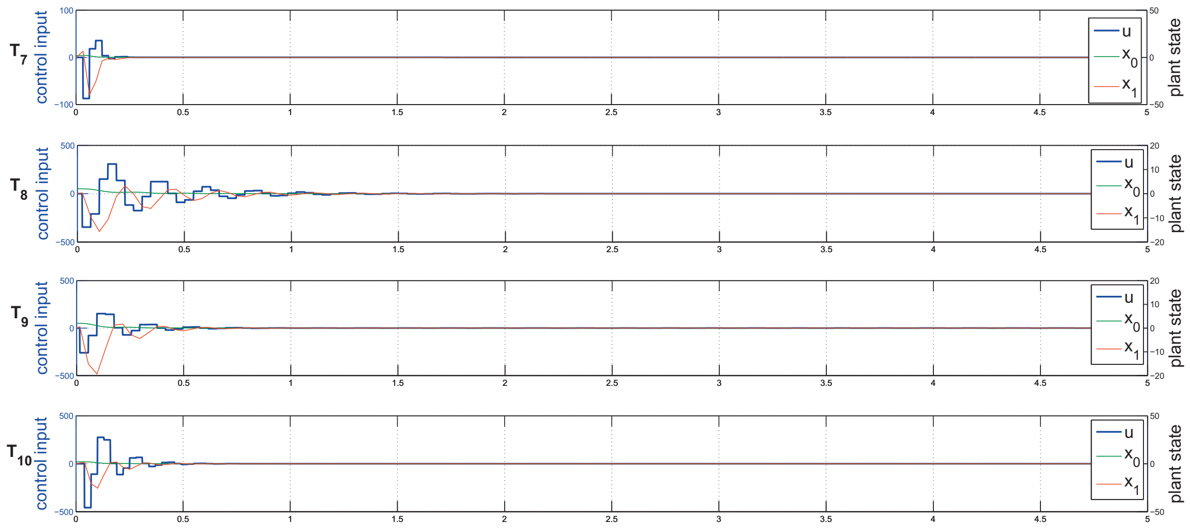

Fig. 14. Observed plants regulated by the control tasks $T_7, T_8, T_9, T_{10} \in \tau_c$ scheduled according to $\mathcal{P}_1$.

we perform the mapping of the remaining (lower) priorities to real-time tasks $\Pi_{rt} \mapsto \tau_{rt}$ in DM fashion. In summary:

- The BR approach is used as a reference for QoC optimization. It achieves optimal QoC J^* but suffers from non-optimal schedulability. This is because the *high-priority bandwidth* is always shared among the control tasks and the real-time tasks are more likely to miss their deadlines when always being assigned low priorities.
- The DM approach is optimal with respect to schedulability, i.e., if there exists a feasible schedule, DM is feasible, too. However, as already shown in Section 4.3, DM is not optimal with respect to QoC optimization.

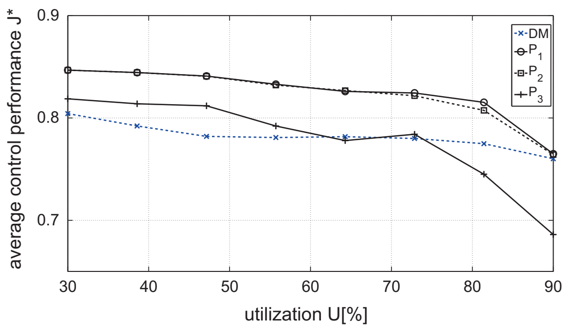

Fig. 15. Control performance vs. utilization.

Towards this, we will show that our *multi-layered* scheduling scheme combines the advantages of DM and BR leading to both good schedulability and QoC optimization.

Observations: For our experiments, we synthesized 200 task sets generating high utilizations $U > 80\%$ (according to the procedure described in Section 4.3), and we evaluated schedulability and QoC for DM, BR, and $\mathcal{P}_1$. The results are summarized and explained in what follows:

- *BR:* Out of the 200 task sets, $N_{BR} = 136$ have been determined as feasible for BR. By definition, if BR is feasible, then the corresponding priority assignment $\Pi(BR)$ results in optimal QoC, i.e., $J^* = 1.27013458$. As a result, $\Pi(BR) = \Pi^*$, and

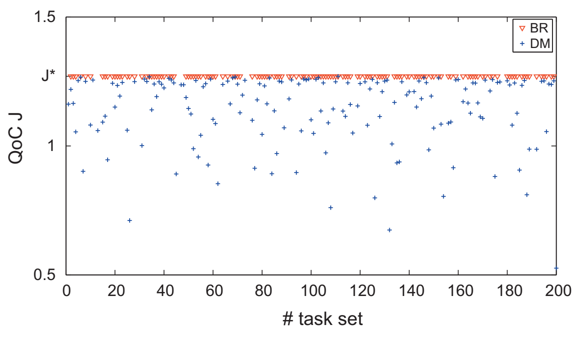

Fig. 16. BR and DM.

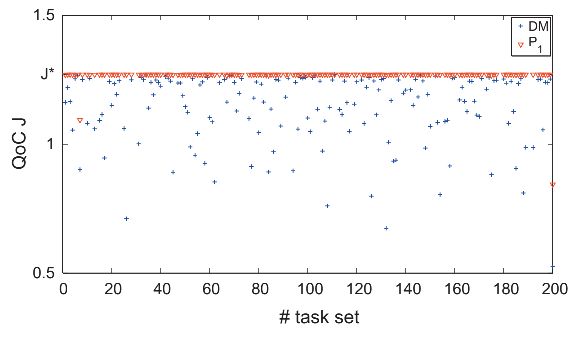

Fig. 17. $\mathcal{P}_1$ and DM.

Table 4
Task set generating $U = 86.75\%$.

T_i	p_i	e_i	d_i	$\Pi(BR) : D_i(\pi_i)$	$\Pi(DM) : D_i(\pi_i)$	$\Pi(\mathcal{P}_1) : D_i(\pi_i)$
$T_1 \in \tau_{rt}$	45	2.7	43	29.7 (10)	29.7 (10)	29.7 (10)
$T_2 \in \tau_{rt}$	5	0.9	**5**	**5.2 (5)**	0.9 (1)	4.0 (4)
$T_3 \in \tau_{rt}$	10	0.8	10	6.9 (6)	1.7 (2)	6.9 (6)
$T_4 \in \tau_{rt}$	20	5.2	20	17.7 (9)	13.9 (6)	17.7 (9)
$T_5 \in \tau_{rt}$	10	0.4	10	7.3 (7)	2.1 (3)	7.3 (7)
$T_6 \in \tau_{rt}$	10	1.1	10	8.4 (8)	3.2 (4)	8.4 (8)
$T_7 \in \tau_c$	30	1.4	(15)	1.4 (1)	4.6 (5)	1.4 (1)
$T_8 \in \tau_c$	40	1.2	(32)	4.3 (4)	17.7 (9)	6.1 (5)
$T_9 \in \tau_c$	40	0.7	(27)	3.1 (3)	16.5 (8)	3.1 (3)
$T_{10} \in \tau_c$	30	1.0	(21)	2.4 (2)	14.9 (7)	2.4 (2)
Schedulabe				No	Yes	Yes
Overall QoC J^*				N/a	0.915	1.269

$J^*(BR) = J^* > J^*(DM)$ for all 136 feasible task sets. This is also depicted in Fig. 16 where the triangles indicate $J^*(BR)$, and the crosses denote $J^*(DM)$ for all the feasible task sets.

- *DM:* Due to the optimality of DM, $N_{DM} = 170 > N_{BR}$ task sets turned out to be feasible which is an improvement of 34 compared to BR. This can also be observed at points in Fig. 16 where there exists a feasible DM schedule (cross) but no corresponding BR schedule (no triangle). However, as expected $J^*(DM) < J^*$ for all feasible task sets as depicted in the figure. The maximum QoC deviation from the optimal value J^* over all feasible task sets is computed as $\Delta J_{max}(DM) = J^* - \min_{i=1\ldots N_{DM}} J_i^*(DM) = 0.7442$ (58.59%), the average QoC deviation from J^* is computed as $\Delta J_{avg}(DM) = J^* - \frac{\sum_{i=1}^{N_{DM}} J_i(DM)}{N_{DM}} = 0.1275$ (10.04%).
- $\mathcal{P}_1$: Our approach is always feasible if DM is feasible, and hence, $N_{\mathcal{P}_1} = N_{DM}$ such that $J^*(\mathcal{P}_1) > J^*(DM)$ (see Fig. 17). Further, for task sets where both BR and DM are feasible, $\mathcal{P}_1$ finds the optimal priority assignment. That is, $\Pi(\mathcal{P}_1) = \Pi(BR) = \Pi^*$, and $J^*(\mathcal{P}_1) = J^*(BR) = J^*$. In cases where BR is not feasible but $\mathcal{P}_1$ is feasible, the achieved QoC is very close to the optimum J^* as $\Delta J_{avg}(\mathcal{P}_1) = 0.0037$ (0.29%). Further, the maximum deviation from the optimum QoC is computed as $\Delta J_{max}(\mathcal{P}_1) = 0.4264$ (33.6%) which is significantly less than in the case of DM.

Table 4 illustrates an example for one of the generated task sets at $U = 86.75\%$. As shown there, BR fails to meet the deadline of real-time task T_2 whereas DM and $\mathcal{P}_1$ satisfy all deadlines. This is because BR assigns the highest priorities to the control tasks T_7, T_8, T_9, T_{10}, in particular, $\pi_7 = 1, \pi_8 = 4, \pi_9 = 3$, and $\pi_{10} = 2$, and the remaining (lower) priorities have been assigned to the real-time tasks $T_i \in \tau_{rt}$ in DM manner. In the example, for $T_2, d_2 = 5$ which is the shortest deadline among all real-time tasks, and hence, $\pi_2 = 5$ (see bold numbers in brackets). However, the corresponding delay experienced by T_2 is computed as $D_2(\pi_2) = 5.2 > d_2$ which results in a deadline violation. In contrast, the other schemes DM and $\mathcal{P}_1$ are feasible, however, as shown in Table 4, the priority assignments $\Pi(DM)$ and $\Pi(\mathcal{P}_1)$ are different. In particular, for $\mathcal{P}_1$ the priorities assigned to the control tasks $T_i \in \tau_c$ are higher than for DM. As a result, the delays experienced by the control tasks are significantly shorter for $\mathcal{P}_1 : D_7 = 1.4, D_8 = 6.1, D_9 = 3.1, D_{10} = 2.4$ compared to DM: $D_7 = 4.6, D_8 = 17.7, D_9 = 16.5, D_{10} = 14.9$. Consequently, the overall QoC $J^*(DM) = 0.915$ is significantly less than $J^*(\mathcal{P}_1) = 1.269$.

5. Concluding remarks

This paper presented a multi-layered schedule synthesis algorithm for MCCPS where we tackle the problem of jointly scheduling deadline-critical real-time tasks and QoC-critical control tasks. The proposed approach integrates RTC-based timing analysis techniques in a multi-layered scheduling framework. That is, real-time tasks are scheduled according to a deadline-driven scheduling policy in the top layer, and control tasks are assigned priorities subject to QoC optimization in the second layer. For this purpose, we presented three different QoC-oriented metrics that we implemented in our framework. We compared our results to classical DM scheduling and a BR approach, and we show that our approach efficiently combines the advantages of DM and BR which significantly improves overall QoC while guaranteeing schedulability. Future work envisages to extend the scheduling framework in two directions. That is (i) integrating the concept of criticality *levels* and building a bridge to conventional MCs theory, and (ii) by exploring the applicability of the presented scheduling scheme to multi-processors which constitute highly relevant processing platforms in future high-end MCCPSs. There is a lot of work on scheduling and schedulability analysis for multiprocessor and multicore platforms. So far, our work only considers a uniprocessor setup and extending it to multiprocessors will be interesting.

References

[1] E.A. Lee. Cyber physical systems: design challenges, in: ISORC, 2008.
[2] R. Obermaisser, C. El Salloum, B. Huber, H. Kopetz, From a federated to an integrated automotive architecture, TCAD 28 (7) (2009) 956–965.
[3] C.M. Kirsch, A. Sokolova, The Logical Execution Time Paradigm, Advances in Real-Time Systems, Springer, 2012 (pp. 103–120).
[4] P. Derler, S. Resmerita, Flexible Static Scheduling of Software with Logical Execution Time Constraints, CIT, 2010.
[5] J. Rushby. New challenges in certification for aircraft software, in: EMSOFT, 2011.
[6] R. Palin, D. Ward, I. Habli, R. Rivett. ISO26262 Safety Cases: Compliance and Assurance, in: System Safety, 2011.
[7] S. Baruah, L. Haohan, L. Stougie, Towards the design of certifiable mixed-criticality systems, in: RTAS, 2010.

[8] S. Baruah, A. Burns, R. Davis. Response-time analysis for mixed criticality systems, in: RTSS, 2011.
[9] J. Lehoczky, L. Sha, J. Strosnider, Enhanced aperiodic responsiveness in hard real-time environments, in: RTSS, 1987.
[10] B. Sprunt, L. Sha, J. Lehoczky, Aperiodic task scheduling for hard real-time systems, Real-Time Syst. 1 (1) (1989) 27–60.
[11] J. Strosnider, J. Lehoczky, L. Sha, The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments, IEEE Trans. Comput. 44 (1) (1995) 73–91.
[12] T. Ghazalie, T. Baker, Aperiodic servers in a deadline scheduling environment, Real-Time Syst. 9 (1) (1995) 31–67.
[13] J. Lehoczky, S. Ramos-Thuel, An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems, in: RTSS, 1992.
[14] R. Davis, K.W. Tindell, A. Burns, Scheduling slack time in fixed priority pre-emptive systems, in: RTSS, 1993.
[15] S. Baruah, V. Bonifaci, G. D'Angelo, A. Marchetti-Spaccamela, S. Van Der Ster, L. Stougie, Mixed-criticality scheduling of sporadic task systems, in: ESA, 2011.
[16] W. Burns, A. Burns, A.J. Wellings, Dual priority assignment: a practical method for increasing processor utilisation, in: Euromicro Workshop on Real-Time Systems, 1993.
[17] R. Davis, A. Wellings, Dual priority scheduling, in: RTSS, 1995.
[18] S. Islam, R. Lindstrom, N. Suri, Dependability driven integration of mixed criticality SW components, in: ISORC, 2006.
[19] K. Lakshmanan, D. de Niz, R. Rajkumar, G. Moreno, Resource allocation in distributed mixed-criticality cyber-physical systems, in: ICDCS, 2010.
[20] S. Vestal, Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance, in: RTSS, 2007.
[21] D. de Niz, K. Lakshmanan, R. Rajkumar, On the scheduling of mixed-criticality real-time task sets, in: RTSS, 2009.
[22] K. Lakshmanan, D. de Niz, R. Rajkumar, Mixed-criticality task synchronization in zero-slack scheduling, in: RTAS, 2011.
[23] K.E. Arzen, A. Cervin, J. Eker, L. Sha, An introduction to control and scheduling co-design, in :CDC, 2000.
[24] S. Samii, A. Cervin, P. Eles, Z. Peng, Integrated scheduling and synthesis of control applications on distributed embedded systems, in: DATE, 2009.
[25] H. Voit, R. Schneider, D. Goswami, A. Annaswamy, S. Chakraborty. Optimizing hierarchical schedules for improved control performance, in: SIES, 2010.
[26] A. Aminifar, S. Samii, P. Eles, Z. Peng, A. Cervin, Designing high-quality embedded control systems with guaranteed stability, in: RTSS, 2012.
[27] E. Bini, A. Cervin, Delay-aware period assignment in control systems, in: RTSS, 2008.
[28] B. Jia, E.P. Eyisi, Q. Fan, X. Yuan, X.D. Koutsoukos, Optimal cross-layer design of sampling rate adaptation and network scheduling for wireless networked control systems, in: ICCPS, 2012.
[29] Y. Wu, G. Buttazzo, E. Bini, A. Cervin, Parameter selection for real-time controllers in resource-constrained systems, IEEE Trans. Ind. Inf. (TII) 6 (4) (2010) 610–620.
[30] A. Cervin, Stability and worst-case performance analysis of sampled-data control systems with input and output jitter, in: ACC, 2012.
[31] R. Majumdar, I. Saha, M. Zamani, Performance-aware scheduler synthesis for control systems, in: EMSOFT, 2011.
[32] D. Goswami, R. Schneider, S. Chakraborty, Co-design of cyber-physical systems via controllers with flexible delay constraints, in: ASP-DAC, 2011.
[33] R.Schneider, D. Goswami, A. Masrur, S. Chakraborty. QoC-oriented efficient schedule synthesis for mixed-criticality cyber-physical systems, in: FDL, 2012.
[34] D. Goswami, M. Lukasiewycz, R. Schneider, S. Chakraborty. Time-triggered implementations of mixed-criticality automotive software, in: DATE, 2012.
[35] J.Y. Leung, On the complexity of fixed-priority scheduling of periodic real-time tasks, Perform. Eval. 2 (4) (1982) 237–250.
[36] J.-Y. Le Boudec, P. Thiran, Network Calculus - A Theory of Deterministic Queuing Systems for the Internet, LNCS, vol. 2050, 2001.
[37] S. Chakraborty, S. Künzli, L. Thiele, A general framework for analyzing system properties in platform-based embedded system designs, in: DATE, 2003.
[38] K. Zhou, J.C. Doyle, K. Glover, Robust and Optimal Control, Prentice Hall Upper Saddle River, NJ, 1996.
[39] A.Y. Bhave, B.H. Krogh, Performance bounds on state-feedback controller with network delay, in: CDC, 2008.
[40] C. Liu, J. Layland, Scheduling algorithms for multiprogramming in hard real-time environments, J. ACM 20 (1) (1973) 46–61.
[41] S. Baruah, A. Mok, L. Rosier. Preemptively scheduling hard real-time sporadic tasks on one processor, in: RTSS, 1990.
[42] R.C. Dorf, R.H. Bishop, Modern Control Systems, Prentice Hall, 2001.
[43] O. Boubaker, The inverted pendulum: a fundamental benchmark in control theory and robotics, in: ICEELI, 2012.

Reinhard Schneider is a PhD candidate in the EE department of TU Munich, from where he obtained his Master's degree in 2006. He interned at TTC in Ann Arbor, Michigan, in the summer of 2011. His work received a best paper award at EUC, and he is also recipient of an Intel Doctoral Student Honor Award for the 2012–2013 academic year.

Dip Goswami is an assistant professor at Technical University of Eindhoven, Netherlands. He obtained his PhD in Electrical and Computer Engineering from the National University of Singapore (NUS) in 2009. Subsequently, he was a post-doctoral fellow at the School of Computing of NUS, where he worked on a General Motors funded project. During 2010–2012, he was an Alexander von Humboldt Postdoctoral Fellow at TU Munich, Germany. His research focuses on various design aspects of embedded control systems in resource-constrained domains such as automotive and robotics. He has published in several international journals and conferences in the fields of control systems, robotics and cyber-physical systems. He has also served on the TPCs of several conferences, such as RTSS, RTAS, DSD, SIES and EUC.

Alejandro Masrur is an assistant professor since August 2013 in the Department of Computer Science at TU Chemnitz, Germany. He obtained his doctoral degree in Electrical and Information Engineering from TU Munich in 2010 and was a senior research associate at the Institute for Real-Time Computer Systems at TU Munich for over two years. His research interests cover real-time systems and scheduling, high-level design of embedded systems, and automotive software among others.

Martin Becker is a PhD candidate at the EE department of TU Munich. He has several years of experience at EADS in the domain of avionic embedded systems and software. He obtained his Master's degree in EE from TU Munich in 2012. His current research is broadly in the area of computer systems, with a focus on embedded and reconfigurable computing.

Samarjit Chakraborty is a Professor of Electrical Engineering at TU Munich, where he holds the Chair for Real-Time Computer Systems. Prior to joining TU Munich, he was an Assistant Professor of Computer Science at the National University of Singapore from 2003–2008. He obtained his PhD from ETH Zürich in 2003. He works on various aspects of system level design and analysis of embedded systems and has more than 150 publications in this area.