2014 IEEE Real-Time Systems Symposium

Formal analysis of timing effects on closed-loop properties of control software

Goran Frehse
Université Joseph Fourier Grenoble 1
Laboratoire Verimag
Gières, France

Arne Hamann
Robert Bosch GmbH
Stuttgart, Germany

Sophie Quinton
Inria Grenoble
Rhône-Alpes
St Ismier, France

Matthias Woehrle
Robert Bosch GmbH
Stuttgart, Germany

***Abstract*—The theories underlying control engineering and real-time systems engineering use idealized models that mutually abstract from central aspects of the other discipline. Control theory usually assumes jitter-free sampling and negligible (constant) input-output latencies, disregarding complex real-world timing effects. Real-time systems theory uses abstract performance models that neglect the functional behavior and derives worst-case situations with limited expressiveness for control functions, e.g., in physically dominated automotive systems. In this paper, we propose an approach that integrates state-of-the art timing models into functional analysis. We combine physical, control and timing models by representing them as a network of hybrid automata. Closed-loop properties can then be verified on this hybrid automata network by using standard model checkers for hybrid systems. Since the computational complexity is critical for model checking, we discuss abstract models of timing behavior that seem particularly suited for this type of analysis. The approach facilitates systematic co-engineering between both control and real-time disciplines, increasing design efficiency and confidence in the system. The approach is illustrated by analyzing an industrial example, the control software of an electro-mechanical braking system, with the hybrid model checker SpaceEx.**

I. Introduction

In automotive systems, software is executed on embedded control units to control physical processes with complex dynamics using sensors and actuators. Examples include engine control, active safety systems (anti-lock braking system and electronic stability control), and braking systems, to mention only a few. Usually, the execution platform executes several applications in parallel. As a consequence, concurrently executed applications compete for processor time. Since only one application can be executed at the same time, a real-time kernel arbitrates the access to the processor according to a scheduling policy, leading to increased and varying response times for the applications due to preemptions, blocking effects, etc.

Worst-case timing analysis techniques that are usually agnostic to the underlying application's functionality are of limited use for industrial control applications. The reason is that the correctness of the application is primarily determined by the functional behavior of the control software instead of only focusing on satisfying deadlines. In particular, the functional properties of industrial control software must consider various goals such as responsiveness, small overshoot, and bounded dynamics.

On the other hand, control design is usually performed assuming idealized timing assumptions, including sampling without jitter and negligible delay from controller input (sensing) to output (actuation). Of course, these idealized assumptions do not hold in practice: the functional behavior is influenced by the above stated timing effects.

For industrial control applications, we need new interdisciplinary techniques that integrate results from timing analysis with functional analysis of control software. The aim of this paper is to introduce a practical approach for co-design between control engineering and real-time systems engineering. To this end, we investigate two prominent models from real-time systems research, arrival curves and typical worst-case models, and integrate them with functional verification techniques. In particular, we propose a system model that allows an engineer to integrate specific timing models into a closed-loop functional model. This functional model is a hybrid model with discrete parts, such as mode switches and clock cycles, and continuous parts such as the physics of the plant. We rely on functional verification of hybrid systems to prove properties of the closed-loop functional model. In particular, this paper uses the hybrid system state space explorer SpaceEx for closed-loop reachability analysis. We validate our approach based on an industrial example, the control software of an electro-mechanical braking system, and two different timing models.

The main *contributions* of this paper can be summarized as follows:

- We propose a novel approach integrating timing and closed-loop verification that is based on a combination of LET (logical execution times) and TWCA (typical worst-case analysis).
- We show how to analyze closed-loop properties of our new model using reachability analysis of hybrid automata.
- We show the effectiveness of our approach on an industrial example of an electro-mechanical brake.

The remainder of this paper is structured as follows: First, we present our system model and introduce our approach for closed loop control software analysis. Second, we describe existing formal analysis of timing models in Section III and discuss them in relation to our proposed closed loop analysis approach. In Section IV we motivate our approach based on an industrial case study. Section V provides background in related work before concluding the paper in Section VI.

II. System model and proposed approach

A. System model

Automotive software systems typically consist of several dozens functionalities that are scheduled using OSEK [1] compliant operating systems. In this kind of operating systems we deal with static priority preemptive scheduling policies. Scheduled entities are called *tasks* that are repeated cyclically with a fixed period. Each task contains several *processes* that

1052-8725/14 $31.00 © 2014 IEEE
DOI 10.1109/RTSS.2014.28

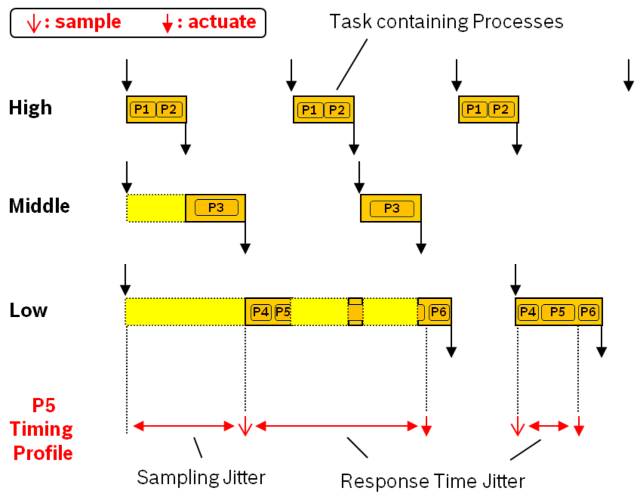


Fig. 1. Sample schedule of a system consisting of three tasks with different priorities. Tasks are mere containers for processes that contain the actual code, e. g., control algorithms. Preemptions by higher priority tasks lead to sampling jitter and response time jitter for process P5 influencing its functional behavior.

contain the functional code. Processes are assigned to tasks according to the continuous dynamics of the physical process they control, e. g., injection control in motor management has higher dynamics than exhaust gas control, and therefore requires a shorter period. In the simplest case that we assume in this paper, one functionality (e. g., one controller) is realized using a single process. However, more complex functionalities are realized using several communicating processes possibly distributed over several tasks. Please note that the approach presented in this paper can be easily extended to this more general case.

Processes are functions without arguments and return values that only communicate over shared memory. For data consistency reasons, all data that is required by processes inside a task, is copied from shared memory into local buffers. Note that this data copy mechanism is not performed directly inside the interrupt releasing the task, but after the releasing interrupt and before executing processes. This is important to mention, since this leads to *sampling jitter* (variable time between task release and process execution) in case the task is directly preempted after release, as can be seen in Figure 1. Data written by processes into shared memory, e. g., updated actuator values as result of a control algorithm, become visible immediately, since there is no equivalent copying mechanism for data written by processes. It is obvious that this semantic leads to a *response time jitter* of a process (variable time from process start to end), a well-known and well-studied effect in real-time systems research. Note that for control this is problematic as one may be working with data read after the nominal sampling time (sampling jitter) and applying the control decisions too late (response time jitter). Both effects might lead to decreased performance of the control software.

For systematically considering timing effects in closed-loop verification, we propose the system model presented in Figure 2(c). This novel approach leverages the two approaches on the left of the figure: Classical timing analysis determines the effect of scheduling on software timing models as shown in Fig. 2(a). Here the abstract scheduler model denotes a policy specifying how the execution platforms activates and preempts tasks and processes. The software timing model is a description of the activation patterns and computation demands of tasks and processes. The results of such an analysis are timing properties such as response time and jitter.

Functional verification of hybrid systems as shown in Fig. 2(b) checks that a closed-loop system comprised of software and plant has certain specified properties. As shown in the figure, verification is often performed on an abstract, continuous model of the software that is agnostic of discretization, scheduling and timing effects. In contrast to timing properties, closed-loop properties concern functional behavior. Examples include control properties, where an internal state should be steered towards a desired set point, e. g., the rise time ("How responsive is my system to a jump in the desired value?") and settling time ("After what time does the internal state stay close to the desired value"). A concrete example for rise time is given in the case study in section Sec. IV-B, where we have the requirement that the brake reacts swiftly (small rise time) to braking requests (jump in set point of braking position).

To consider the impact of timing effects on closed-loop performance, our closed-loop model explicitly includes a scheduler property model derived from timing analysis. *The scheduler property model specifies the points in time where data values are read from the plant and written back to the actuator*. We can use this for our closed-loop analysis by composing the scheduler property models with a refined, discretized software model (and the plant) as shown in Fig. 2(c). We derive our scheduler property models from existing timing analysis techniques, namely busy window-based approaches using arrival curves and Typical Worst Case Analysis [2]. Before detailing on specific timing analyses and how to derive scheduler property models, we explain our approach for closed-loop verification.

B. Timing-aware, closed-loop verification

We use model checking to verify the specified closed-loop properties as follows. Hybrid automata as a modeling formalism combine continuous-time and discrete-event dynamics. Discrete-time systems can be modeled by generating a discrete event at the corresponding sampling times. Hybrid automata allow us to combine models from different domains by embedding them into the same continuous-time, discrete-event framework, e. g., an ordinary differential equation (ODE) model of the plant, a discrete-time model of the controller, and an abstract timed model of the scheduler. So the plant, the controller, and the scheduler are each modeled by one or more hybrid automata, which are then composed into a hybrid automata network. The model checker symbolically computes the set of reachable states for the entire network, which is sufficient to decide safety properties (nothing bad ever happens) and bounded liveness properties (something good happens within a given time frame). The set of reachable states can be queried within the model checker to obtain conservative bounds on the variables, e. g., maximum overshoot, or to check whether a given set of state is reachable or not. This mechanism can be used to check more complex properties by adding a so-called monitor automaton to the network. A monitor automaton features a fail state that is reachable if the property is violated, and thus encodes the property as a reachability problem. If the model checker can show that the fail state is not reachable when the monitor is composed with

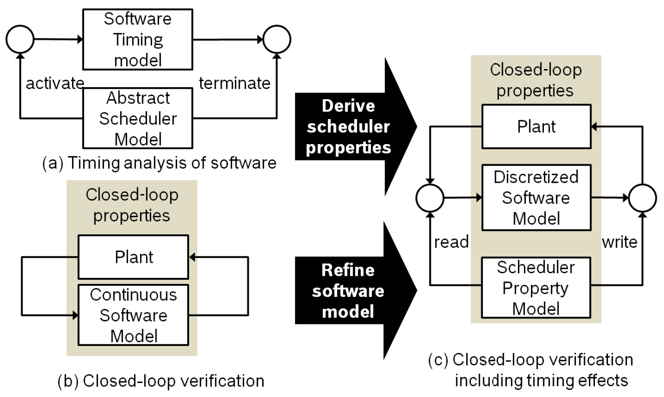

Software Timing model
activate
terminate
Abstract Scheduler Model
(a) Timing analysis of software
Closed-loop properties
Plant
Continuous Software Model
(b) Closed-loop verification
Derive scheduler properties
Refine software model
Closed-loop properties
Plant
Discretized Software Model
read
write
Scheduler Property Model
(c) Closed-loop verification including timing effects

Fig. 2. (a) Classic timing analysis focuses on the effect of schedulers on software timing, e. g., meeting certain deadlines. (b) Hybrid system verification uses closed-loop models and specification of properties on the closed-loop behavior to prove correct functional behavior. (c) Our proposed approach joins timing analysis with hybrid system verification. We leverage a scheduler property model describing all possible behaviors of the discretized system on the execution platform, such that the verification can check that the closed-loop properties hold.

the closed-loop model, then the property is guaranteed to be satisfied. For example, to show that the system reaches a target set within a given time frame, the monitor consists of an initial state, a fail state, and a clock. The fail state is connected to the initial state with a transition that is enabled if the system has not reached the target set at the end of the time frame, which is measured by the clock.

The heterogeneous closed-loop model described above is not trivial to analyze. Model checkers for hybrid systems traditionally struggle with systems that feature a lot of continuous variables, but recently developed approximation techniques allow one to handle certain systems with hundreds of variables at an acceptable loss of accuracy, e. g., [15]. Nonetheless, the approach needs to handle a combinatorial explosion both in terms of the number of reachable discrete states and in terms of the reachable sets of continuous variables. One of the key challenges of the work presented in this paper is to develop an abstract scheduler property model that is expressive enough to yield useful information, and for which closed-loop model checking is possible with reasonable computational cost.

C. Existing approaches for timing analyses

Different possibilities have been previously proposed to quantify sampling jitter and response time jitter in order to make them accessible for systematic analysis. Note that these approaches are all further detailed in the related work in Section V.

1) Traces: Use scheduling simulation to derive traces describing the points in time where data is read and written back.
2) Probabilistic approaches: Model sampling and response time jitter as random variables with known expectations.
3) Analytic models such as arrival curves.

While method 1 can be used in simulation contexts to get a first impression about the behavior of a controller under scheduling influences, it only considers single scenarios and cannot be used to give any formal guarantees about the behavior of the control algorithm. Probabilistic approaches such as method 2 require to model the distribution of the jitter which is practically infeasible to derive. The reason is hidden dependencies between tasks and processes, that are not only due to scheduling and communication but also due to digital hardware mechanisms such as caching and pipelining. Method 3 captures the scheduling influences safely, and thus represents a sound basis for formal verification approaches. However, the safe over-approximation of BCRT (best-case response time) and WCRT (worst-case response time) by the upper and lower arrival curve leads to a large set of possible behaviors that hinders efficient and realistic analysis. Additionally, it is not clear how the information captured by the arrival curves can be leveraged constructively for systematic, timing-aware controller design.

D. Novel approach for integrating timing and closed-loop verification

In this paper we prefer to use an alternative approach for design and analysis of timing-aware controllers. The approach allows for a safe approximation and formal proof of controller properties, while being constructive, in the sense that control engineers can account for timing effects during controller design, and thus design more efficient, yet still robust controllers.

In order to combine real-time analysis methods constructively with control engineering, we propose to employ the well known concept of *Logical Execution Times (LET)* [3]. LETs can be used to enforce deterministic data read and data write semantics[1], which is very useful for designing and composing control software.

The determinism provided by the LET abstraction greatly reduces the effort for closed-loop analysis and integrates well with scheduling analysis for correct-by-construction system synthesis. More precisely, using platform mechanisms implementing the LET semantics, data produced by each process can be written back at its WCRT. By this means, the WCRTs can be interpreted as systematic and constant latencies during control design. This greatly reduces the set of possible timing behaviors to a single deterministic schedule, thereby facilitating controller design and functional verification. The reason is that we do not need to consider reading and writing as separate non-deterministic models, but can integrate them into one joint scheduler property model.

Unfortunately, such a straight-forward approach is not feasible in practice. The reason is that WCRTs are usually far

[1]Jitters are eliminated at the cost of increased latency

too large (up to 90% of the period in automotive systems) to be useful for designing controllers that are robust against model errors and external disturbances. A much more practical approach is combining the LET semantic with the *Typical Worst-Case Analysis (TWCA)* method [2].

E. Typical Worst-Case Analysis (TWCA)

TWCA exploits the fact that worst-case scheduling situations for a process under analysis are rare and usually do not occur repetitively for subsequent executions. Based on this observation, TWCA derives a typical worst-case response time (TWCRT) that is far smaller than the actual WCRT, as well as error bounds on the number of violations of that TWCRT in a given time window. More precisely, the output of TWCA consists of (*i*) a TWCRT bound on the response time and (*ii*) a function err such that, for $k \geq 1$, it holds that out of k consecutive executions, *at most* $err(k)$ response times may be larger than the computed TWCRT. Table I shows an example of such an error (number of deadline misses) for some values of k (consecutive executions). For example, the second line reads as follows: out of 18 consecutive executions, at most 3 may have a response time that is larger than TWCRT. Note that the error computed by TWCA is similar to the properties verified on *weakly-hard* systems [4].

The complete theory behind TWCA is beyond the scope of this paper. Informally, TWCA is based on the identification of some specific activation patterns, e. g., for interrupt service routines, which clearly have a typical case and an exceptional case — think of periodic bursts as an example [5]. TWCA performs two response-time analyses instead of one: the first analysis is the standard WCRT analysis mentioned before, while the second analysis uses the same approach on a different model in which the non-typical activation scenarios are ignored. This being done, TWCA uses some information obtained during the WCRT analysis to determine the error associated to the TWCRT. Basically, the error is computed by considering the fact that the impact of a non-typical activation cannot last forever in a schedulable system, and can in fact be bounded by the worst-case busy window (called busy period in its first mention in [6]). As a consequence, knowing the frequency of non-typical activation scenarios, which can be represented in the simple case using an arrival curve, one can determine how many executions at most may be impacted by a non-typical activation and thus experience a response time above the TWCRT.

For system synthesis we propose to assume that data is written back deterministically at the TWCRT. Thus, the timing determinism provided by the LET semantic can be exploited without inflicting prohibitively high input-output delays for the controller. However, compared to the approach using the WCRT as discussed above, we now have to account for the effect of TWCRT violations during verification. This is a trade-off: For controller design, we can leverage a smaller dead time (TWCRT $\ll$ WCRT), while for verification the resulting set of timing behaviors to analyze will increase slightly. However, we will show that this increase is very small compared to the number of behaviors in systems with jittering response times.

In this paper we assume that when the process violates the TWCRT, its execution is discarded, and the last written data remains valid. From a control design perspective, this is a valid assumption, since there exists a vast amount of control approaches (notably in the domain of networked control) that can systematically account for sampling losses and prove whether or not the controller is stable and adheres to required performance criteria [7], [8]. From the scheduling analysis perspective, note that the work by Quinton et al. [2] is based on a different assumption, namely that executions which miss their deadline run until completion. For static priority preemptive schedulers, using directly the original analysis provides conservative results w. r. t. our new hypothesis: killing an execution which has missed its deadline will directly reduce the delay inflicted to subsequent executions of the same process while having no impact of the interference from higher priority tasks.

A more formal and general comparison of the two scheduling strategies regarding weakly-hard guarantees is left for future work.

III. Formal analysis of timing models

In this section, we consider two different timing models – arrival curves and typical worst-case models – and show how to use them within our system framework for closed-loop analyses in III-B. Please note that both models are inherently different. We will present both models and discuss their differences in Section III-C.

Before diving into the concrete models for the analyses we provide an overview of the relevant background on hybrid and timed automata.

A. Background: Hybrid and timed automata

We model the interaction of discrete events and continuous, time-driven dynamics with hybrid automata and follow the model of Alur et al. [9]. A hybrid automaton is a tuple $\mathcal{H} = (\mathcal{L}, \mathcal{V}, Init, \mathcal{F}, \mathcal{I}, \mathcal{T}, \mathcal{B})$ consisting of a labeled graph with vertices $\mathcal{L}$, the so-called locations, and edges $\mathcal{T}$, the so-called transitions. It describes the nondeterministic evolution of a finite set of continuous variables $\mathcal{V}$ over time. We associate each of the n variables with a dimension in $\mathbb{R}^n$. A state is a tuple $(l, x) \in \mathcal{L} \times \mathbb{R}^n$. $Init \in \mathcal{L} \times \mathbb{R}^n$ describes the initial state. In every state (l, x), the time-driven evolution of the continuous variables, the so-called flow, is described by a set of differential equations or inclusions, i. e., $\dot{x} \in \mathcal{F}(l, x) \subseteq \mathbb{R}^n$. The system may remain in location l, evolving according to the flow $\mathcal{F}(l)$, only as long as the state is inside the invariant $\mathcal{I}(l)$.

A transition $(l, b, \mu, l') \in \mathcal{T}$, with label $b \in \mathcal{B}$ describes the discrete transition (edge) from location l to location l', $l, l' \in \mathcal{L}$. A transition is guarded by a guard $g \in \mathcal{G}$, i. e., the transition may be taken if for the current state (l, x), $x \in g$. When taking the transition the state can nondeterministically jump to any successor in the *jump relation* $\mu \subseteq \mathbb{R}^n \times \mathbb{R}^n$, i. e., it can jump from (l, x) to (l', x') iff $(x, x') \in \mu$. The set of states that can jump is called the *guard* of the transition, and consists of the projection of μ onto x. In the following we use SpaceEx for formal verification of hybrid systems models. SpaceEx accepts hybrid automata with piecewise affine dynamics (PWA), i. e., where the flow $\mathcal{F}$ is described by linear ordinary differential equation (ODE). Invariants, jump relations, and initial states are described by linear predicates (polyhedra) over the state variables.

Hybrid automata can be combined to form complex models by running them in parallel. This *parallel composition* allows the hybrid automata to share variables (details are omitted for lack of space), and synchronize on shared transition labels: a transition may take place in one of the automata only if

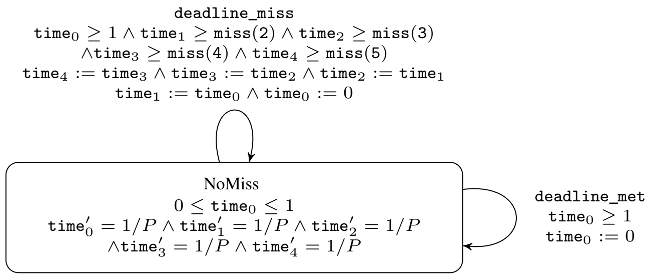

deadline_miss
$\mathtt{time}_0 \geq 1 \wedge \mathtt{time}_1 \geq \mathtt{miss}(2) \wedge \mathtt{time}_2 \geq \mathtt{miss}(3)$
$\wedge \mathtt{time}_3 \geq \mathtt{miss}(4) \wedge \mathtt{time}_4 \geq \mathtt{miss}(5)$
$\mathtt{time}_4 := \mathtt{time}_3 \wedge \mathtt{time}_3 := \mathtt{time}_2 \wedge \mathtt{time}_2 := \mathtt{time}_1$
$\mathtt{time}_1 := \mathtt{time}_0 \wedge \mathtt{time}_0 := 0$
NoMiss
$0 \leq \mathtt{time}_0 \leq 1$
$\mathtt{time}_0' = 1/P \wedge \mathtt{time}_1' = 1/P \wedge \mathtt{time}_2' = 1/P$
$\wedge \mathtt{time}_3' = 1/P \wedge \mathtt{time}_4' = 1/P$
deadline_met
$\mathtt{time}_0 \geq 1$
$\mathtt{time}_0 := 0$

Fig. 3. Scheduler property model based on TWCA as modeled in SpaceEx.

all other automata that feature the same label can also take a transition with this label. Synchronizing transitions are then executed simultaneously. A more detailed description on how to model control systems in SpaceEx is described in [10].

A simple example of a hybrid automaton as it is used in this paper is shown in Figure 3. It has a single location, i. e., $\mathcal{L} = \{\mathtt{NoMiss}\}$. There are 5 continuous variables $\{\mathtt{time}_0, \mathtt{time}_1, ..., \mathtt{time}_4\} \in \mathbb{R}^5$. Each of these variables describes a clock with constant derivative. The invariant $\mathcal{I}(\mathtt{NoMiss})$ describes that we can stay in location NoMiss for at most one period. There are two possible transitions, one with label deadline_met (on the right) and one with label deadline_miss. Note that in this case deadline_met corresponds to the point in time where data is read from the plant. Writing data back to the plant is then performed deterministically after the TWCRT deadline. In other words, in this model we decide at reading time whether a deadline will be met or missed. We can take the deadline_met transition after exactly one period, since the guard checks that $\mathtt{time}_0 \geq 1$ and the invariant imposes that $\mathtt{time}_0 \leq 1$. In this case, we reset $\mathtt{time}_0$:=0, i. e., start a new period. The other transition has a more restrictive guard (with a conjunction of predicates on the clock variables) and more elaborate assignment of the clock variables that is detailed below.

Note that in previous timing analysis approaches using state-based analysis, timed automata are used, especially in the context of Uppaal. Timed automata are a special class of hybrid automata, where the continuous variables represent clocks, that have a constant flow and can only be reset to zero. Obviously, all timed automata can be represented as hybrid automata and due to the larger expressiveness of hybrid automata models may be modeled in a simpler, more concise manner.

B. Scheduler property models as hybrid automata

We use arrival curves and TWCA to derive scheduler property models as hybrid automata. We use a compositional approach, where we can cleanly separate scheduler property models that generate events that in this case trigger data reads from the plant model.

1) Arrival curve models: For arrival curves, we follow the results of Lampka et al. [11], [12]. They present how to convert abstract stream models such as PJD (periodic with jitter) or time-interval based models to a network of co-operating timed automata (TA). As described TA are a special subclass of HA. As such, it is straight-forward to integrate these TA models into our framework.[2]

Note that we used arrival curves solely for modeling the sampling jitter in this paper. Generally, we also need to model response time jitter. However, in our first evaluation focusing on verification efficiency, already the sampling jitter arrival curve model rendered closed-loop analysis infeasible for state-of-the-art verification tools.

2) Typical worst-case models with LET: In the following, we describe an approach to derive scheduler property models using TWCA for systems with LET semantics. These weakly-hard models are called *TWCA models* for simplicity in the following. As a running example, we use the specific TWCA results in Table I, where we denote the maximum number of deadline misses in a given number of subsequent sampling periods. The corresponding model is shown in Figure 3. The basic idea of the approach is that we have one clock that models the periodic schedule of the controller, here $\mathtt{time}_0$. Note that we model clocks with a period P that we use as a normalization in the flow. The clock $\mathtt{time}_0$ together with the guards on the two transitions model the fixed period, i. e., each period we have to take one of the transitions and reset the clock accordingly.

For each row in Table I, we need to measure the interval over which we have missed a specific number of deadlines. As we can see in the figure, we need a dedicated clock to measure each of these intervals, concretely $\mathtt{time}_1$, the interval over which we had one miss, up to $\mathtt{time}_4$, the interval over which we had four misses. We use these measured intervals to decide whether another schedule deadline may be missed over the current interval. Therefore these clocks are used in the transition on the bottom of Figure 3 in the guard condition. As an example, we are allowed to miss a third deadline in the current period if there were only two misses over at least the last 18 consecutive executions. As such, we use in this case $\mathtt{time}_2$ with the threshold for 3 misses miss(3). Analogously, the same holds for all other model thresholds. Please note that taking the transition at the top of Figure 3 with label deadline_miss corresponds to a deadline miss. Therefore, this transition is guarded by measuring the time intervals in which a certain number of deadlines were missed before, i. e., using miss(2), miss(3), etc. as discussed above.

Intuitively, the TWCA model controls writing of the data. When the deadline is missed, data is not written to the actuators. For verification, we simplify the model by realizing that in these cases, where we do not write the data to the plant, we do not need to compute new actuator values in the first place and therefore do not need to sample the sensors in this case. As we have a deterministic, fixed-timing relationship between a read and a write operation in our model, we synchronize deadline_met with the read operation. This is a safe simplification of the model to facilitate verification. Note that the LET that we assume for the execution of the controller is modeled by a fixed delay between reading and writing. To further improve efficiency, we use a variation of the hybrid automaton in Figure 3 in which clock values are no longer computed when they are not needed. This is achieved by having a location each with one, two, three, and four active

[2]While these models can be simplified in the more powerful hybrid automaton formulation, e. g., due to not being limited to variable reset to zero, this is out-of-scope of this work.

# deadline misses	consecutive executions
2	2
3	18
4	20
5	56

TABLE I. TYPICAL WORST-CASE ANALYSIS MODEL FOR A GIVEN PROCESS THAT IS EXECUTED PERIODICALLY EACH 1 ms. THE WORST CASE RESPONSE TIME IS 0.8 ms. THE TYPICAL CASE RESPONSE TIME IS 0.4 ms WITH THE GIVEN BOUNDS ON MISSES PER CONSECUTIVE EXECUTIONS.

clocks. The automaton selects the locations depending on how many events are being tracked, i. e., how many clocks are still within the largest time frame (here `miss(5)`).

3) Formal definition of TWCA automaton: Formally, we construct a hybrid automaton from the TWCA model as follows. Let P be the sampling period, and let N be the maximum number of deadline misses considered. The TWCA model consists of an N-dimensional integer vector `miss` which encodes that a maximum of i deadline misses can take place in any $\mathtt{miss}(i)$ consecutive executions (sampling periods). The hybrid automaton is

$$\mathcal{H}_{\text{TWCA}} = (\mathcal{L}, \mathcal{V}, \mathit{Init}, \mathcal{F}, \mathcal{I}, \mathcal{T}, \mathcal{B}),$$

with a single location $\mathcal{L} = \{\mathtt{NoMiss}\}$. The variables $\mathcal{V} = \{\mathtt{time}_0, \ldots, \mathtt{time}_{N-1}\}$, where $\mathtt{time}_i$ models the time since i misses have been observed. The initial states are

$$\mathit{Init} = \mathtt{NoMiss} \times \{\mathtt{time}_0 = 0 \wedge \bigwedge_{1 \le i < N} \mathtt{time}_i = \mathtt{miss}(N)\},$$

where the $\mathtt{time}_i$ for $i \ge 1$ are set to a value high enough that they initially do not restrict the occurrence of events. The flow is given by

$$\mathcal{F}(\mathtt{NoMiss}) = \{ \bigwedge_{0 \le i < N} \mathtt{time}'_i = 1/P\},$$

i. e., all variables are clocks normalized on the period P, and the invariant is

$$\mathcal{I} = \{\mathtt{NoMiss}\} \times \{0 \le \mathtt{time}_0 \le 1\}.$$

The set of transition labels is $\mathcal{B} = \{\mathtt{deadline\_met}, \mathtt{deadline\_miss}\}$. The transition label `deadline_met` serves to trigger the control actions in the rest of the model via synchronization of transitions that have this label. The label `deadline_miss` synchronizes with a dummy automaton that allows transitioning at all times. There are two transitions in $\mathcal{H}_{\text{TWCA}}$. First, the occurrence of an event is modeled by a transition $(\mathtt{NoMiss}, \mathtt{deadline\_met}, \mu_{\mathtt{deadline\_met}}, \mathtt{NoMiss}) \in \mathcal{T}$, where

$$\mu_{\mathtt{deadline\_met}} = \{\mathtt{time}_0 \ge 1 \wedge \mathtt{time}'_0 = 0 \wedge \bigwedge_{1 \le i < N} \mathtt{time}'_i = \mathtt{time}_i\},$$

i. e., $\mathtt{time}_0$ is reset while the other clocks do not change. The transition guard and the invariant together ensure that the `deadline_met` transitions take place when $\mathtt{time}_0 = 1$. Since $\mathtt{time}_0$ is reset to zero, the `deadline_met` transitions can only occur exactly at the sampling period P. Second, missing an event is modeled by a transition $(\mathtt{NoMiss}, \mathtt{deadline\_miss}, \mu_{\mathtt{deadline\_miss}}, \mathtt{NoMiss}) \in \mathcal{T}$, where

$$\mu_{\mathtt{deadline\_miss}} = \{\mathtt{time}_0 \ge 1 \wedge \bigwedge_{1 \le i < N} \mathtt{time}_i \ge \mathtt{miss}(i+1)$$
$$\wedge\, \mathtt{time}'_0 = 0 \wedge \bigwedge_{1 \le i < N} \mathtt{time}'_i = \mathtt{time}_{i-1} \},$$

i. e., $\mathtt{time}_0$ is reset and the other clocks take the value that corresponds to the previously missed event. Due to the guard constraints, the transition is only enabled if the time since the i-th miss is greater or equal to $\mathtt{miss}(i+1)$, which enforces the semantics of the TWCA model.

C. Differences between both scheduler property models

Arrival curves are used in timing analysis methods to model the activation patterns of tasks and processes. They do not make any assumptions about underlying platform mechanisms (e. g., scheduling policy) or other concurrently executed tasks and processes. TWCA models, on the other hand, are synthesized from more information, and are already the result of an analysis step taking into account platform mechanisms, timing constraints, and other concurrent tasks and processes. It is, therefore, clear that the arrival curve model is more general than the TWCA model.

However, the restricted TWCA model exactly covers many relevant industrial scenarios, and makes the verification question addressed in this paper a much simpler task that scales considerably better than using arrival curve models (compare Section III-E). In particular, the scheduler property model for TWCA is discrete (number of deadlines misses in a given number of consecutive executions), whereas the space of possible activation patterns described by arrival curves is continuous. Therefore, TWCA models represent a suitable trade-off between accuracy, generality and scalability for combining timing analysis with closed-loop functional verification. The same is not true for arrival curve models: Integrating arrival curves into closed-loop models leads to a computationally expensive verification task for realistic systems (with state-of-the-art methods). This is not surprising as the computational complexity already occurs in state-based timing analysis using arrival curves with Timed Automata [13].

D. Reachability analysis for closed-loop, hybrid models

To check whether the specification is satisfied, we compute the set of reachable states of the system, i. e., all states that lie on one of the trajectories. The computation is exhaustive, so that all states are covered independently of the number of transitions or the time spent in locations. Computing the reachable states is similar to numeric simulation, except that sets of values are used instead of numbers. One can regard this as a generalization of interval-based ODE solvers that are guaranteed to contain the exact solution, but with one difference: While even guaranteed ODE solvers only compute the ranges of reachable values at specific discrete points in time, computing all reachable states involves also covering the states in the continuum between those time points. In our tool SpaceEx, this is achieved using first order Taylor approximations, whose remainder is bounded conservatively. Reachability is an undecidable problem for all except the most simple dynamics, so one tries to compute a conservative cover that is precise enough to prove that the specification is satisfied.

While numerical simulation is fast and can be carried out for arbitrarily complex, nonlinear dynamics, it is limited to computing one single behavior (trajectory) at a time, in a deterministic scenario in which all side conditions (inputs, disturbances, initial conditions) are attributed a fixed numerical value. Reachability computation, on the other hand, is carried out with sets, and one can therefore take into account a wide variety of nondeterminism, even continuous nondeterminism such as disturbances that act continuously over time and are only known up to some bounds. A single run of reachability computation can therefore replace a very large number of simulations, which would otherwise be necessary to cover all combinations. The downside of reachability analysis is the computational complexity, namely in the form of state space explosion, and the fact that scalable methods are so far known mainly for simple dynamics such as linear ODEs.

One of the main difficulties in reachability analysis is the computation with high-dimensional continuous sets. The key lies in finding suitable set representations. The representation must be easy to compute with, in particular the operations for solving the ODE (equivalent to low-order ODE solver code) and the operations to cover the continuum between time points (convex hull and Minkowski sum). Furthermore, geometric operations such as union and intersection are necessary, e. g., to apply invariant and guard constraints. Polyhedra in constraint representation are suitable for some of these operations, but scale terribly for others. Recently, scalability breakthroughs have been achieved by combining suitable algorithms on the ODE side with "lazy" set representations (support functions) and template polyhedra [14]. Template polyhedra are polyhedra whose facet normals have fixed directions, and approximating the solution of a number of set operations with template polyhedra can be done very efficiently. The downside of templates is that they are fixed in advance and may work more or less well depending on the model and its dynamics. For the results presented in this paper, we use templates that correspond to computing a bounding box on the states at each point in time, which is somewhat similar to solving the ODEs with interval arithmetic.

E. Advantages of Typical Worst-Case Analysis over Arrival Curves for Closed-Loop Models

While arrival curves present an elegant way to model uncertainty in the scheduling of controller events, the nondeterminism that they introduce in the timing turns out to be very challenging for the type of reachability analysis we consider. In the arrival curve model, events can arrive at any time over a certain interval. This means that one must compute the reachable states over that interval before the event, and then the set of successors of those states after the event. This is inherently difficult for our type of analysis, since the reachable states over an interval of time is generally a non-convex set, while successor computations are carried out on convex sets. So the nonconvex set forcibly needs to be covered with a collection of convex sets before the successor computations can be carried out. This is also known as convexification, and the process produces a number of convex sets that increases with the length of the interval over which the event can take place. Since convexification is carried out for each occurrence of an event, this can quickly lead to an exponential increase in the number of sets that have to be computed. To take the case study presented in the next section as an example, we could compute the reachable states using a conservative arrival curve model only up to three (!) time units, while the typical worst-case analysis can be carried out over 14 time units, which is sufficient for the specification.[3]

Modeling the scheduler property with timed arrival curves leads to a sequence of events that occur nondeterministically on the time axis. In the closed-loop model, this corresponds to the discrete controller updates taking place at any time over a relatively large time interval. Combining this nondeterminism with the plant dynamics leads to a difficult model for a hybrid model checker. Informally speaking, the model checker needs to evaluate an exponential function (the solution of the ODEs of the plant) over time intervals of increasing length. The influence of the timing uncertainty is therefore highly nonlinear. In our experiments, we encounter a massive explosion in the number of symbolic states and an accumulation of the approximation error over time.

Modeling the scheduler property with the TWCA model, on the other hand, leads to a closed-loop model in which the discrete controller updates take place at regular intervals. While there may be nondeterminism in terms of the occurring events, and in terms of the states of the plant states, the timing of events is deterministic and thus easier to handle by the model checker. Informally speaking, the model checker needs to evaluate an exponential function (the solution of the ODEs of the plant) at only discrete points in time. Since the systems are piecewise linear, the influence of any nondeterminism in terms of the plant states is linear. Experimentally, we see only a mild state explosion in our model, and the approximation error remains small over the considered time horizon.

IV. Motivating case study: EMB

A. EMB and control systems in industry

The basic mechanical structure of an electro mechanic braking system (EMB) contains the following components:

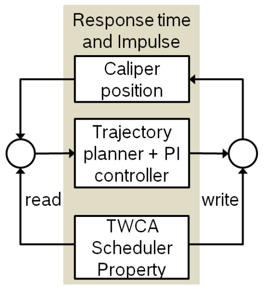
- DC Motor
- Gear box
- Brake caliper
- Brake disk
- Brake pedal as interface to the driver

Figure 4(a) shows the concretization of our general approach from Fig. 2(c): our focus is the software that controls the electro-mechanical brake. We have two requirements on response time and impulse that we need to verify on the closed-loop model. We have a scheduler property model based on TWCA as shown in Fig.3.

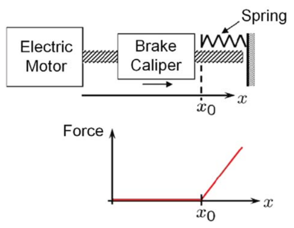
The EMB model consists of several parts: The electric DC motor, driving the brake caliper, comprising of the rotating mass of the rotor and the spindle, the gearbox between the rotational mass of the rotor and the translation mass of the caliper, as well as the translational mass of the caliper including a stiff spring model for the brake disk.

A brake request is realized using two different modes of operations as shown in Fig. 4(b). First, the DC motor moves the brake caliper into position x_0, denoting the position at the brake disk. Once position x_0 is reached, the DC motors applies the requested braking force. In this paper we focus on the positioning of the caliper at the braking disk. This part of

[3]The TWCA analysis exhaustively computes a cover of all reachable states up to 14 time units in 40 s, giving a total of 1423 symbolic states. The arrival curves analysis can compute all states for a bounded horizon of 2 time units within 30 s, giving a total of 316 symbolic states. For a bounded horizon of 3 time units, we obtain 2659 symbolic states in 814 s. For a bounded horizon of 4 time units, the analysis times out after two hours.



(a) Concretization of system model: Using a TWCA scheduler property model to analyze the closed-loop properties of an electro-mechanical brake (EMB).



(b) Simplified EMB plant model for required functionality. A brake cycle consists of: 1) positioning of the caliper into position x_0 (brake disk), and 2) applying force to the brake disk.

Fig. 4. EMB case study: concrete timing-aware closed-loop analysis and plant functionality

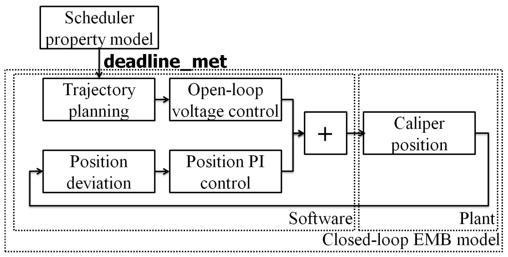


Fig. 5. Schematic of the closed-loop model consisting of (discretized) software and plant, the scheduler property model and the synchronization using the `deadline_met` label.

the functionality is of interest for several scenarios: braking, disk wiping, and pre-crash preparations.

Figure 5 shows a schematic of our model in SpaceEx. The figure indicates the typical closed-loop model of software and plant and the synchronization with the scheduler property model from Fig. 3 using the label `deadline_met`.[4] Please note that the discretized software model includes data read at activation (by the `deadline_met`) as well as writing to the actuators at the TWCRT using LET semantics.

B. EMB requirements

For the correct functioning of the braking functionality the following requirements must be enforced:

- *Response time* for reaching the braking position x_0 ($t_{R_{max}}$): the caliper must reach $x_0 = 0.05\ dm$ after the braking request is issued within 20 ms with a precision of 4%. This requirement ensures the reactiveness of the system.
- *Small impulse*: If the caliper hits the braking disk with a high impulse, force closure occurs and the driver feels an abrupt deceleration. While this might be acceptable for braking, it must not happen in other scenarios where the caliper is positioned close to the braking disk (e. g., disk wiping). The caliper speed at contact must be below 2 mm/s.

[4]Similarly we composed the system with a TA model of arrival curves.

C. Analysis in SpaceEx

We are able to show that the system satisfies the specification by computing the reachable states of a model in which we stop the evolution of the system when within 4% of contact ($x \leq 0.048$). As initial conditions we fix that the brake caliper is at rest in position $x = 0$. A global timer t is added to the model in order to measure the response time. We model the TWCA property model based on Table I. The control output is applied with LET semantics 0.4 ms after the typical case response time.

We use the tool SpaceEx [14] with the STC algorithm from [15] to compute a finite cover of the reachable states. The computation is exhaustive, i. e., it terminates with a fixed point without any bounds on the number of transitions or on the time horizon. The result nonetheless ranges over finite time since the evolutions are stopped in the model by the constraint $x \leq 0.048$, i. e., trajectories beyond this level are disregarded. On a standard laptop (MacBook Pro), the computation takes 40 s, giving a total of 1423 symbolic states. The memory consumption is less than 50 MB.

Figure 6(a) shows the projection of the reachable states from the 12-dimensional state space onto the t/x plane. It shows that the response time of the braking position is at around 12.5 ms, which satisfies the specification by a good margin. The figure also shows the branching of trajectories caused by skipping actuator updates due to violations of the typical case response time as specified in Table I. To illustrate a case where the caliper response time is violated, we consider a modified version where the controller setpoint is chosen deliberately close to violating the specification. As shown in Figure 7, only one single case fails the contact specification. The failure is drastic, since the limit on the response time is exceeded by a factor of more than two. Note that the problematic timing scenario corresponds to neither the slowest nor the fastest response, but to one of the middle cases. This shows that our method can detect the presence of realistic problems in control software considering timing.

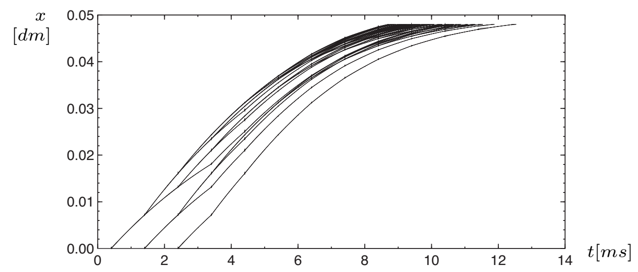
For the second requirement on small impulse, we observe the current I, which in our model is proportional to the caliper velocity, i. e., $\dot{x} = cI$, with $c = 0.008$. The requirement is therefore satisfied if $I \leq 1.25\ A$. This is verified by intersecting the reachable states with the plane of contact, $x = 0.048$, and projecting onto the variable I. This gives the bounds $I \in [0.38, 0.99]$, which satisfies the specification. As an illustration, Figure 6(b) shows the reachable values of I at the point of contact, for all possible time instants at which contact can occur under the given typical case response time.

This first proof-of-concept application with its positive results shows the great potential of our timing-aware closed-loop analysis. Nevertheless, there are several questions that remain for future work, such as (*i*) inclusion of uncertainties (e. g., in initial position) and disturbances and (*ii*) detailed performance assessment on additional examples.

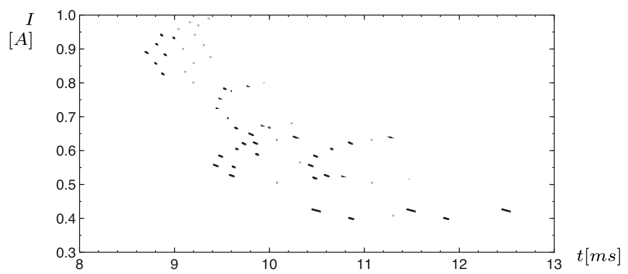
V. Background and Related Work

A. Timing analysis

Classical work in real-time systems is mainly concerned with the so-called *schedulability* of a system. The system models considered by many contributions usually assume application models consisting of periodic and sporadic tasks with precedence constraints. Tasks (sometimes also task chains) are associated with deadlines that must be fulfilled in order



(a) Position over time as analyzed in SpaceEx for the typical worst-case analysis. The different branches that we see in the the position evolution are the result of skipping actuator updates due to violations of the typical case response time as specified in Table I.



(b) Current over the different possible time instants at which the brake caliper may get in contact with the brake disks, as analyzed in SpaceEx. Since the current is proportional to the velocity of the brake caliper, this provides bounds on the impulse of the collision between the caliper and the disk.

Fig. 6. Visualization of closed-loop behavior of the EMB from SpaceEx reachability analysis.

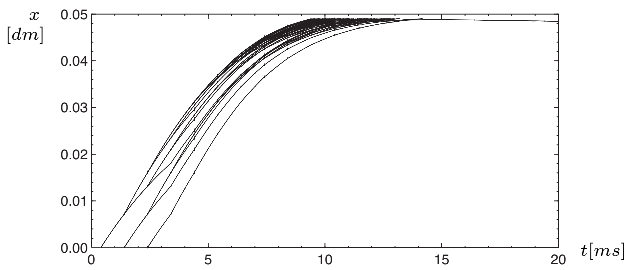


Fig. 7. Position over time as analyzed in SpaceEx for a controller set-point deliberately close to violating the specification. Out of the many timing scenarios, only one single case fails the contact specification.

for the system to be *schedulable*. The concrete techniques applicable for answering the question whether or not a given system is schedulable strongly depend on the assumed platform model and the scheduling strategy. In cases that the application and platform models are simple enough, *maximum utilization bounds* can be derived to decide whether or not a given system is schedulable. Prominent examples for this kind of analyses are, for instance, the work of Liu and Layland for independent periodic task sets under rate-monotonic fixed priority scheduling on a single core platform [16], or the work of Dertouzos on earliest deadline first (EDF) scheduling [17]. An overview of extensions on those basic works can be found under [18].

When it comes to more complex application and platform models as they are used, for instance, in automotive systems, methods based on the so-called *busy window approach* combined with reasoning about the *critical instant* as proposed by Lehoczky [6] must be used in order to successfully analyze a system's real-time behavior. As of today there exist industry strength tools, such as SymTA/S[5], that are capable of analyzing complex distributed systems consisting of platforms with heterogeneous scheduling strategies and bus arbitration protocols with high precision.

Lampka et al. [11] showed a hybrid analysis based on a general approach to model arrival curves as (networks of) timed automata. We leverage their work as scheduler property models for real-time calculus. However, different to our work they do not consider the functional dynamics of the software, but the schedulability of a system.

[5]www.symtavision.com

Until today the above described methods have only found little attention in the development of real-world automotive applications. The reason is that automotive systems strongly interact with the environment to sense and control physical processes. This has been neglected by those approaches. In fact reasoning about real-time scheduling of task sets in isolation is of limited practical use, since it represents only a small building block for proving correct functional behavior in automotive systems.

B. Co-engineering approaches

Of course, we are not the first to recognize the need of integrating real-time scheduling approaches with functional behavioral models. One example is the work on the *TrueTime* toolbox [19] that enhances Simulink with scheduling simulation for simple scheduling policies. This enables function developer to assess the functional behavior of control applications under temporal effects that are due to real-time scheduling (latency and jitter) already on model level. While this is very useful in practice, our work focus on a comprehensive approach leveraging formal verification rather than on single simulations. Another approach in this line of research is implemented in the *JitterBug* toolbox [20]. Here, timing effects are described using probabilistic models that can elegantly be integrated into control engineering system theory to prove stability and control performance. The drawback is that probabilistic models are rarely adequate to describe the timing behavior in automotive systems. The main reason are unknown dependencies due to digital hardware mechanisms that are impossible to capture.

Kumar et al. [21] extend the work in [11] and use the above-detailed hybrid analysis for controller design. The authors use a similar strategy as our TWCA-based approach by using a delay threshold that is smaller than the worst-case yet typically met. Then, in controller design, the sporadic misses that can be quantified by the hybrid analyses, are used in the stability analysis of the controller. In contrast, we do not focus on stability of a single controller, but want to verify general properties of a closed-loop system such as responsiveness and overshoots of a system that even may include different controllers and mode switches.

Alur et al. [22] synthesize an automaton that includes all schedules that satisfy given stability and settling-time requirements. This is complementary to our approach as (*i*) they consider stability properties (and settling time) of the closed-loop system, while we are interesting in general safety properties and (*ii*) their starting point are the properties from which they derive the class of acceptable schedules, while we describe the set of possible schedules and evaluate all system evolutions given these schedules. Both Alur et al. [22] and our approach use a logical execution time abstraction.

Related to our system model approach is the work on cyber-physical system design contracts [23]. These contracts are very similar to our scheduler property models and describe several similar scheduling analysis approaches that may be useful for contracts including the LET approach that we generalize using TWCA by including deadline misses. Derler et al. [23] share our vision of a co-engineering approach between control design and software design. In contrast to their work, we focus in this work on computational methods and corresponding modeling for systematic verification of closed-loop functional properties instead of relying on simulations.

VI. Conclusion

For industrial embedded applications controlling physical processes, systematic co-engineering is indispensable. Focusing on timing effects, we need co-engineering approaches that integrate state-of-the-art timing analysis and closed-loop, functional analysis. This allows software engineers to verify the correctness and performance of their control software early in the design process, while still considering timing effects introduced by scheduling on the digital ECU hardware. In this paper, we motivate and present such a co-engineering approach.

To this end, we propose to integrate scheduler property models from existing timing analysis techniques into closed-loop system models to systematically analyze the closed-loop properties of control software. We present a system model that allows software engineers to evaluate the effect of timing on closed-loop systems using a model-based approach. We showcase this approach based on two different timing analysis techniques. We discuss the industrial relevance of these models and identify weakly-hard models based on typical-case analysis as very suitable for design and analysis of closed-loop system models. We present a relevant industrial case study of an electro-mechanical brake, where we used these weakly-hard (TWCA) models to verify on a model level that the closed-loop system satisfies its specification, here with respect to two different performance criteria of the controller. Additional to timing, there are other effects that might influence the closed-loop behavior on the target platform, e. g., sensor and actuator variances. Our approach based on functional verification easily allows us to extend our models to integrate these uncertainties in future work.

VII. Acknowledgments

This research was partly funded by the German Federal Ministry of Education and Research (BMBF), under grant "SPES XTCore" (01IS12005).

References

[1] OSEK VDX, "Open systems and the corresponding interfaces for automotive electronics," http://www.osek-vdx.org.

[2] S. Quinton, M. Hanke, and R. Ernst, "Formal analysis of sporadic overload in real-time systems," in *DATE*, 2012, pp. 515–520.

[3] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 84–99, 2003.

[4] G. Bernat, A. Burns, and A. Llamosí, "Weakly hard real-time systems," *IEEE Trans. Computers*, vol. 50, no. 4, pp. 308–321, 2001.

[5] S. Quinton, M. Negrean, and R. Ernst, "Formal analysis of sporadic bursts in real-time systems," in *DATE*, 2013, pp. 767–772.

[6] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadlines," in *Proceedings of IEEE Real-Time Symposium (RTSS)*, 1990, pp. 201–209.

[7] L. Schenato, B. Sinopoli, M. Franceschetti, K. Poolla, and S. S. Sastry, "Foundations of control and estimation over lossy networks," *Proc. IEEE*, vol. 95, no. 1, pp. 163–187, 2007.

[8] R. Blind and F. Allgöwer, "On the stabilizability of continuous-time systems over a packet based communication system with loss and delay," in *World Congress of the International Federation of Automatic Control (IFAC)*, 2014.

[9] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical computer science*, vol. 138, no. 1, pp. 3–34, 1995.

[10] A. Donzé and G. Frehse, "Modular, hierarchical models of control systems in spaceex," in *Proc. European Control Conf. (ECC'13)*, Zurich, Switzerland, 2013.

[11] K. Lampka, S. Perathoner, and L. Thiele, "Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems," in *Proceedings of the seventh ACM international conference on Embedded software*. ACM, 2009, pp. 107–116.

[12] ——, "Component-based system design: analytic real-time interfaces for state-based component implementations," *International Journal on Software Tools for Technology Transfer*, vol. 15, no. 3, pp. 155–170, 2013.

[13] ——, "Analytic real-time analysis and timed automata: A hybrid methodology for the performance analysis of embedded real-time systems," *Design Automation for Embedded Systems*, vol. 14, no. 3, pp. 193–227, 2010.

[14] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *CAV*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 379–395.

[15] G. Frehse, R. Kateja, and C. Le Guernic, "Flowpipe approximation and clustering in space-time," in *Proc. Hybrid systems: computation and control*. ACM, 2013, pp. 203–212.

[16] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," in *Journal of the ACM*, vol. 20(1), 1973, pp. 46–61.

[17] M. L. Dertouzos, "Control robotics: the procedural control of physical processes," in *Proceedings of the IFIP Congress*, 1974, pp. 807–813.

[18] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-Time Systems*, vol. 28, no. 2-3, pp. 101–155, Nov. 2004.

[19] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. Arzen, "How does control timing affect performance? analysis and simulation of timing using jitterbug and truetime," *Control Systems, IEEE*, vol. 23, no. 3, pp. 16–30, June 2003.

[20] B. Lincoln and A. Cervin, "Jitterbug: A tool for analysis of real-time control performance," *Proceedings of the 41st IEEE Conference on Decision and Control*, vol. 2, pp. 1319–1324, 2002.

[21] P. Kumar, D. Goswami, S. Chakraborty, A. Annaswamy, K. Lampka, and L. Thiele, "A hybrid approach to cyber-physical systems verification," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 688–696.

[22] R. Alur and G. Weiss, "Regular specifications of resource requirements for embedded control software," in *Proceedings of the 2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, ser. RTAS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 159–168. [Online]. Available: http://dx.doi.org.acm.rb-han.de.bosch.com/10.1109/RTAS.2008.13

[23] P. Derler, E. A. Lee, S. Tripakis, and M. Törngren, "Cyber-physical system design contracts," in *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*, ser. ICCPS '13. New York, NY, USA: ACM, 2013, pp. 109–118. [Online]. Available: http://doi.acm.org.acm.rb-han.de.bosch.com/10.1145/2502524.2502540