Analytic Real-Time Analysis and Timed Automata: A Hybrid Method for Analyzing Embedded Real-Time Systems

Kai Lampka, Simon Perathoner, and Lothar Thiele
Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland
{lampka,perathoner,thiele}@tik.ee.ethz.ch

ABSTRACT

This paper advocates a strict compositional and hybrid approach for obtaining key (performance) metrics of embedded systems. At its core the developed methodology abstracts system components by either flow-oriented and purely analytic descriptions or by state-based models in the form of timed automata. The interaction among the heterogeneous components is modeled by streams of discrete activity-triggers. In total this yields a hybrid framework for the compositional analysis of embedded systems. It supplements contemporary techniques for the following reasons: (a) state space explosion as intrinsic to formal verification is limited to the level of isolated components; (b) computed performance metrics such as buffer sizes, delays and utilization rates are not overly pessimistic, because coarse-grained purely analytic models are used for components only which conform to the stateless model of computation. For demonstrating the usefulness of the presented ideas we implemented a corresponding tool-chain and investigated the performance of a two-staged computing system, where one stage exhibits state-dependent behavior only coarsely coverable by a purely analytic and stateless component abstraction.

Categories and Subject Descriptors

C.4 [**Performance of systems**]: Modeling techniques

General Terms

Performance, Design, Verification

Keywords

Performance Analysis, Timed Automata, Real-Time Calculus, Hard Real-Time Systems

1. INTRODUCTION

When it comes to guarantees w. r. t. system behaviors, the latter must be analyzed by thorough and sound mathematical methods. Testing and simulation are in general not sufficient since they are not exhaustive. Timed automata [1] are well accepted for analyzing real-time systems, see e. g. [14]. However, the finite state/transition system to be derived from some high-level model tends to grow exponentially w. r. t. the number of clocks and clock constants. Therefore, the detailed analysis of a complex system may be hampered in practice, if not impossible at all. In contrast, purely analytic (or stateless) methods such as provided by the Real-Time Calculus [17, 19], SymTA/S [10] or MAST [8] solely depend on solutions of closed form expressions, yielding a very good scalability w. r. t. the size of systems to be analyzed. But, this advantage leads to serious drawbacks: (a) analytic methods are limited to the computation of a few specific system measures and (b) each method is restricted to a specific model to which the system specification under analysis must be translated, which in general may lead to overly conservative analysis results. To overcome these shortcomings, this paper aims to combine purely analytic and state-based performance analysis methods. Employing state-based evaluation approaches only for those system components, where state-less analysis delivers overly pessimistic results will maintain scalability.

In the present work we have chosen to combine TA (Timed Automata) and RTC (Real-Time Calculus), as the former is widespread for verification of real-time systems and RTC is a very general analytic performance analysis approach, see [17, 5, 19]. However, we would like to point out that the presented method is not limited to RTC. The coupling of TA with other analytic performance evaluation frameworks such as any method from classical real-time analysis or SymTA/S can be reduced to a special case of what is discussed here.

Coupling the Modular Performance Analysis framework (MPA) [19] which is based on RTC [17] and Uppaal [18] for the joint analysis of embedded real-time systems is far from trivial, since (a) the RTC lacks a concrete execution semantics unlike TA, (b) TA can not be verified by evaluating closed form expressions, nor can one in general derive an analytic description from them and (c) RTC and TA not even share the same time domain. TA operate on the conventional time-line, whereas the RTC operates on stream abstractions that are defined on time intervals. To overcome this obstacles this paper provides the following contributions:

- A pattern is described allowing to convert abstract stream models such as PJD (periodic with jitter) or time-interval based models used in RTC to a network of co-operating TA (Sec. 4.1).

EMSOFT'09, October 12–16, 2009, Grenoble, France.

- Correctness and exhaustiveness is proven, i. e., the TA solely generate traces conforming to the abstract stream model, and does this for all conformant traces (Sec. 4.1.3).
- A pattern is described to automatically derive abstract stream models (such as PJD or those used in the context of RTC) from a TA-based system model (Sec. 4.2).
- Finally the paper also presents an implementation and analyzes an exemplarily chosen system (Sec. 5).

2. RELATED WORK

There are several other approaches known which also tackle the combination of RTC-based analytic and state-based models for system-wide performance analysis. The authors of [16] bridge the gap between a state-based methodology and the RTC-method as discussed in Sec. 3. However, contrary to this paper the work of [16] is based on event count automata (ECA) [6]. With ECA the user must specify the minimum and maximum number of event arrivals taking place while the ECA resides in the respective location. For translating an RTC-based abstract stream representation into an ECA the authors use the principle of a ring buffer. Each counter represents the number of events associated with the respective number of unit intervals. When it comes to the interfacing of ECA with RTC, i. e., one needs to derive abstract stream representations as used in RTC-curves from ECA specifications, the authors of [16] suggest the usage of observer ECA. They use binary search for extracting the maximum and minimum number of events seen in a window size Δ via reachability analysis. In [15] it is shown how the above approach can be employed within a hybrid framework, allowing to obtain key performance metrics of embedded systems by combining RTC and ECA-based analysis. However, our usage of TA appears to be more beneficial, since contrary to ECA they have an explicit notion of time, whereas ECA advance in a lock-step fashion. In addition, in our work we solely require one observer automaton for a complete staircase function defined over all time intervals Δ rather than one observer per discrete window size Δ.

The authors of [7] present an approach, where a system to be analyzed is mapped to a process network which is analyzed via a compositional response time analysis [10]. The resulting periodic event stream models and the computed response times serve as parameters for pre-defined TA. The high-level descriptions of system properties to be checked are also transformed into TA. Finally, the use of standard model checking procedures allows to check, whether the system model fulfils the desired properties or not. The approach differs from the new results in this paper as the system is not decomposed into components which exhibit substantial state-dependent behavior and those which can be analyzed using analytic approaches. Instead, state-based behavior is not explicitly taken into account.

The authors of [11] also address the combination of RTC-based components and TA. For including the abstract stream representation used in RTC into TA-based system models one operates on an array of clocks. Each clock is associated with the number of events produced so far, as well as with a minimal and maximal number of events to be generated within the respective time interval length. For deriving RTC-based stream representations from the combined model, the authors suggest the usage of observer automata and binary search on the maximal and minimal number of events that appear within any time interval of length Δ, which is in fact similar to the idea of [16]. As one operates on a finite set of time-interval lengths Δ only, it is not clear when to stop with the translation of an abstract event stream representation into a TA and vice versa. The use of observer automata that investigate single time-interval lengths only implies that one either needs one observer automaton with its local clock for each interval length, or one must execute a full state space exploration for each of the interval lengths. Also on the side of the event generating automaton, the number of clocks may be prohibitively large because one basically needs one clock per upper and lower bound for the number of events seen on the stream within the resp. time interval Δ. The approach described in this paper attempts to overcome these limitations by using a compositional leaky bucket representation of event streams.

3. BACKGROUND THEORY

Definitions: In the following, we will make use of a few notations that are described next:

(a) A timed action is a pair (t, a) where a is some edge label and $t \in \mathbb{R}^{\geq 0}$ some non-negative time stamp. In particular, we will consider the edge-label *event*, either signalling the incoming/outgoing of some event into/from some TA.

(b) A timed trace $\tau := (a_1, t_1); (a_2, t_2); \ldots$ is a sequence of timed actions ordered by increasing time stamps, s.t. $t_i \leq t_{i+1}$ for $i \geq 1$.

(c) For addressing the evaluation of a clock x or a counter b at some time t we will use the notation $x(t)$ or $b(t)$. We will also use the clock identifier instead, namely in cases where the concrete point in time t is clear.

Real-Time Calculus (RTC) [17, 5] extends the classical Network Calculus, see e. g. [4], towards analyzing distributed (hard) real-time systems. Contrary to other analysis techniques, streams and their counting functions are not defined on the time domain, but on the time-interval domain. At first we define now the differential counting function $R : \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0}$ which yields the number of events seen on a timed trace in the interval $[s, t]$. Such functions can be bounded by upper and lower arrival curves α^u and α^l defined in the time-interval domain:

$$\alpha^l(t-s) \leq R(s,t) \leq \alpha^u(t-s) \text{ with } 0 \leq s \leq t \qquad (1)$$

Thus, $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ bound the maximal and minimal number of events arriving in *any* interval $[s, t]$ of length $\Delta := t - s$. We denote such functions defined on the time-interval domain as *RTC-conformant*. Analogously one may also define an upper and lower bound for the availability of resources, where $C(s, t)$ is the corresponding differential accumulative function for the amount of available resource units in the time interval $[s, t]$. $\beta^u(\Delta)$ and $\beta^l(\Delta)$ are the corresponding upper and lower bounds in time-interval domain, denoted as upper and lower service curves. They determine upper and lower bounds on the available resource in *any* interval $[s, t]$ of length $\Delta := t - s$.

In a pure RTC-based modeling approach event arrival and service curves provide the inputs to a single analysis component. For computing the corresponding bounds $\alpha'^{\{u,l\}}$ for the outgoing event stream and $\beta'^{\{u,l\}}$ for the remaining resources one commonly applies operators of min-plus and max-plus algebra, see [17, 19]. Overall, this component-

based analysis methodology allows to obtain (hard) bounds on job delays, buffer sizes and utilization of hardware units, either for a single component or for complex systems.

With TA one can only make use of discrete variables, rather than continuous ones. Hence, we consider systems with discrete numbers of events or resource units, where on the level of RTC this refers to staircase functions. One of the major results of this paper is the transformation between the time-interval domain to the time domain, independent of whether the corresponding curves represent events or resources units. In the following, we will therefore generically speak of upper and lower curves γ^u and γ^l, addressing RTC-conformant event-based or resource-based staircase functions. The complexity we will face is the fact, that the bonding functions γ^u and γ^l in time-interval domain implicitly define a possibly infinite set of timed traces, namely all traces the differential counting function R or C of which is bounded in the sense of Eq. 1.

Timed Automata: We follow the concept of timed safety automata as found in the model checker Uppaal [2, 3]. We briefly re-capitulate some related constructs.

(a) Cooperation via shared variables: Variables can be declared on the level of a network of TA, allowing the individual TA to read and manipulate them.

(b) Rendez-vous mechanisms: Uppaal makes use of channels and signals. In fact this implements different rendez-vous mechanisms for jointly executing edges within different TA and thus allowing the composition of individual TA into a network of co-operating TA. By following Uppaal's nomenclature we will also often speak of sender and receivers when referring to the synchronization of TA. One may distinguish the following concepts:

- Binary synchronization: A sending and a receiving TA synchronizes on the joint execution of two dedicated edges, one from the sender, whose edge is labeled by a `channel id` and an exclamation mark, and one from the receiver, whose edge is labeled by the same `channel id`, but extended with a question mark. For simplicity we speak of sending- and receiving edges (see the *event*! and *event*?-labeled edges in the TA of Fig. 2(B) and 2(C)).
- Broadcast channels: One sender synchronizes with up to n receivers. This refers to the situation where one sender executes a sending edge, which can be understood as the emission of a signal and where between 0 and n receivers execute a receiving edge, which can be interpreted as the instantaneous reception of this broadcast signal. It is important to note, that all at the time enabled receiving TA have to execute their respective receiving-edge (see the *event*! and *event*?-labeled edges in the TA of Fig. 3(A–C)).

(c) Location invariants: A location can only be entered once its invariant holds. For the execution of an edge this yields that besides the enabling condition also the invariant of the target location must hold.

(d) Urgent and committed locations: Within urgent locations no time passes. Thus the system has to execute any downstreamed transition in zero time once the urgent location is entered. In case this is not possible the TA-system deadlocks. In Uppaal urgent locations are marked accordingly (see the right location of the TA of Fig. 3(C); the locations marked with a circle are the initial ones).

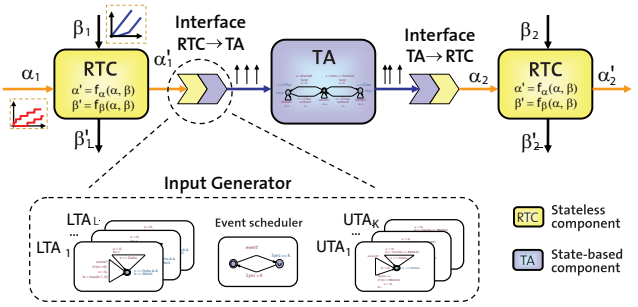


Figure 1: Overview of hybrid analysis method

(e) Customized operators: The TA depicted in this paper are not always syntactically correct, e. g. the `if`-statement employed by us has to be implemented in Uppaal with the `?`-operator of ANSI C. For the max- and min-operator one needs to define individual functions as part of Uppaal's system declaration.

4. THE APPROACH

In the following we will develop a scheme for interfacing RTC-conformant staircase curves with TA-based system descriptions. The developed strategy consists of two independent parts (Fig. 1), which will be discussed separately: (i) we show how-to implement staircase **input** curves by a network of TA denoted as input generator. (ii) we discuss how to derive staircase **output** curves from TA-based system specifications. This way, upstream as well as downstream system components could be analyzed on the basis of RTC.

4.1 Input generator

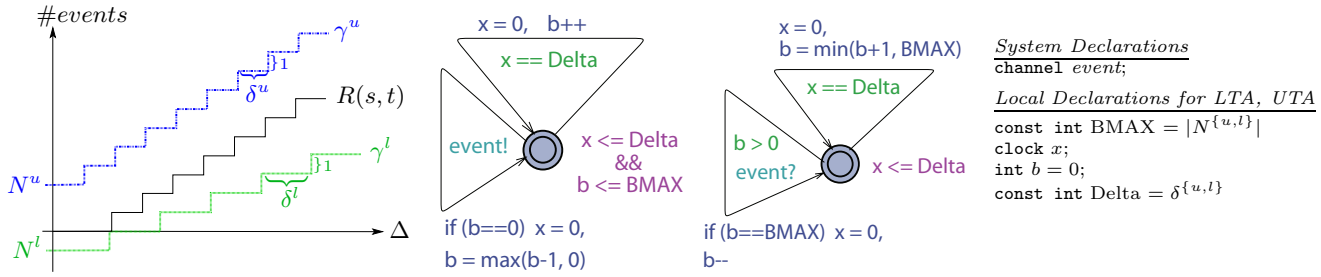
For interfacing RTC-conformant curves to TA the approach has to translate time-interval-based functions into TA-based representations of possibly infinitely many timed streams. The main idea for achieving this is based on the observation that the interval-based bounding functions γ^u and γ^l can be modeled by sets of individual staircase functions combined by the (nested) application(s) of minimum or maximum; details on modelling input curves with staircase functions follow in Sec. 4.1.5. Each of the involved staircase functions $\gamma_i^{\{u,l\}}$ is guarded by its own TA, where LTA i guards *lower* curve γ_i^l and UTA i guards *upper* curve γ_i^u. The network of co-operating UTA and LTA emits a dedicated, instantaneous signal *event* to the environment, where this signal can be used for stimulating a user-defined TA-based model description which represents some component of the system under analysis. Emission of the *event*-signal has to be done in such a way that one is capable of producing each timed trace $tr := ((event, t_0), \ldots (event, t_n) \ldots)$ of *event*-signals whose differential counting function is bounded by $\gamma^{\{u,l\}}$. For keeping the discussion as simple as possible, we will start now with the most simple case, where $\gamma^{\{u,l\}}$ are defined by a single staircase-function each.

4.1.1 *Linear input pattern*

We define an upper or lower staircase function as follows:

$$\gamma^{\{u,l\}}(\Delta) := N^{\{u,l\}} + \left\lfloor \frac{\Delta}{\delta^{\{u,l\}}} \right\rfloor \qquad (2)$$

where in our approach each is guarded by its own timed automaton denoted UTA, LTA respectively. Binary syn-



(A) Upper and lower input curves (B) LTA for guarding γ^l (C) UTA for guarding γ^u (D) Declarations

Figure 2: Linear pattern: Input curves and their TA-based implementation

chronization enforces UTA and LTA to produce only those traces which contain at least $\gamma^l(\Delta)$ and at most $\gamma^u(\Delta)$ *event* signals for any interval of length $\Delta \in \mathbb{R}^{\geq 0}$.

For exemplification please refer to the curves depicted in Fig. 2. The parameter N^u can be understood as burst capacity, which describes the number of events producible in zero time according to curve γ^u. The parameter $\delta^{\{u,l\}}$ specifies the minimum/maximum distance of two successive events with respect to curve $\gamma^{\{u,l\}}$. The absolute values of parameter N^l of the lower curve can be understood as the fictitious numbers of δ^l delays before event emission has to be enforced every δ^l time unit. The above constants provide the values for the free parameters in UTA and LTA.

The implementation of UTA and LTA is shown in Fig. 2(B) and 2(C). Each of them employs its own clock x, counter b, and its constants $BMAX$ and $Delta$ (Fig. 2(D)). The edge-guards (green) steering the execution of the different edges, clock resets (blue), variable updates (blue) and signal sending and reception (light blue) are specified next to the respective edge. Location invariants (light purple) are stated next to the affected location. UTA and LTA cooperate by synchronizing on sending and receiving signal *event*, which enforces the joint execution of the *event*! and the *event*?-labeled edges in the two TA. The non-deterministic emission of events (= sending and receiving of the *event*-signal by LTA and UTA) is possible as long as for the UTA $b > 0$ holds. On the other hand emission of the signal *event* has to take place once the local variable b of LTA reaches its local threshold $BMAX$, which is enforced by the location invariant $b <= BMAX \wedge x <= Delta$. It is also important to note that for $b = 0$ and the production of an event LTA resets its local clock x, whereas UTA does so once $b = BMAX$ holds and the signal *event* is sent.

The proof of correctness and completeness is briefly sketched now, where concise details will follow once we are dealing with more complex RTC-conformant event curves.

(A) Correctness w. r. t. the upper and lower bound:

1. Binary synchronization enforces that the linear input generator can only produce traces with at most $N^u + \lfloor \frac{\Delta}{\delta^u} \rfloor$ timed *event*-actions seen on any interval of length Δ. This is because event emission is blocked once b of UTA equals 0 and because for $b = N^u$ and an event emission UTA's local clock x is reset.

2. The invariant defined on LTA's initial location enforces that after $(N^l + 1) \cdot \delta^l$ time unit an event is emitted and from now on at least every δ^l time unit. Therefore, every trace produced by the input generator contains at least $\gamma^l(\Delta)$ timed *event*-actions for any interval Δ.

(B) Completeness: The input generator allows non-deterministic production of events, bounded by γ^u and γ^l as indicated above. By contradiction it can easily be shown, that all timed *event* traces where UTA blocks clearly violate the upper bound if one would have seen an extra *event* on any of these positions. Thus all traces which are not producible by the input generator are clearly not contained in the set of traces defined by γ^u and γ^l.

4.1.2 Convex and concave input pattern

Now we extend the discussion to cases where the input functions γ^l or γ^u are modeled as the maximum or minimum of a set of staircase functions:

$$\gamma^u(\Delta) := \min_i(\gamma_i^u(\Delta)) \quad ; \quad \gamma^l(\Delta) := \max_i(0, \gamma_i^l(\Delta)) \qquad (3)$$

where $\gamma_i^{\{u,l\}}(\Delta)$ is defined as stated in Eq. 2 but now with their individual pairs of parameters $N_i^{\{u,l\}}, \delta_i^{\{u,l\}}$. For exemplification one may refer to the curve(s) depicted in Fig. 3. In accordance with Eq. 3, as well as with the scenario of Fig. 3 it is required that for all $i < j$ holds:

$$|N_i^{\{u,l\}}| < |N_j^{\{u,l\}}|;\ \delta_i^{\{l,u\}} > 0;\ \delta_i^l > \delta_j^l \text{ and } \delta_i^u < \delta_j^u \qquad (4)$$

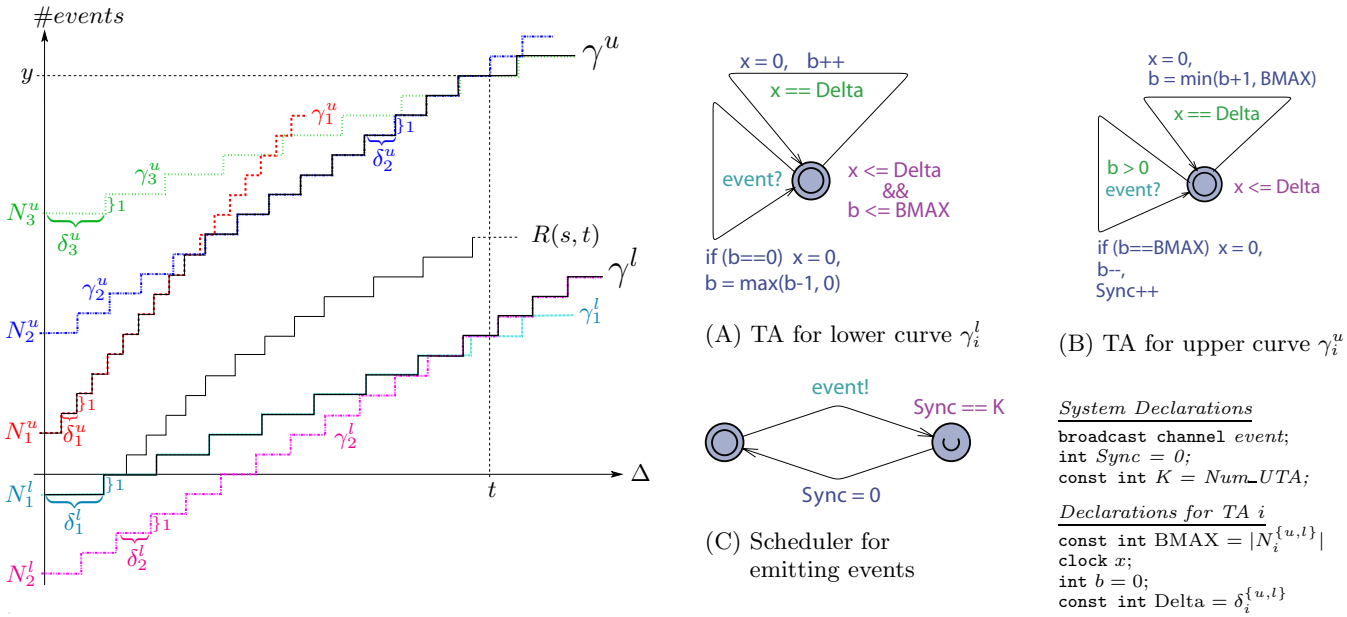
Basic idea of the approach: The bound imposed by curve $\gamma_i^{\{u,l\}}$ will be guarded by UTA and LTA i, respectively. Cooperation among the UTA and LTA has to be organized in such a way, that it complies with the minimum and maximum-operation as employed in Eq. 3. Due to the usage of minimum and maximum the following conditions apply:

(1) Minimum-condition: The input generator **may** emit timed action $(event, t) \iff \forall b_i(t) \in \mathcal{B}^u : b_i(t) > 0$, where $\mathcal{B}^u$ is the set of the UTA-local b-variables such that $i \in \{1, K\}$ with K as the number of UTA.

(2) Maximum-condition: The input generator **has to** emit timed action $(event, t) \iff \exists b_i \in \mathcal{B}^l \wedge \exists x_i \in \mathcal{C}^l$ such that $b_i(t) = |N_i^l| \wedge x_i(t) = \delta_i^l$, where $\mathcal{B}^l$ is the set of the LTA-local b-variables, $\mathcal{C}^l$ is the set of their local clocks and δ_i^l the period for incrementing x_i, with $i \in \{1, L\}$ where L is the number of LTA.

The above operations are implemented on the basis of broadcast channels and location invariants as explained now.

The implementation is shown in Fig. 3. To be as generic as possible we make use of Uppaal's concept of templates, s. t. clock x, constants $BMAX, Delta$, and the counter b of the TA shown in sub-figure (A) and (B) are local entities only, and will be indexed accordingly. The instances of sub-figure (A) and sub-figure (B) implement the single LTAs and UTAs of the network, respectively. The TA shown in sub-figure (C) is the scheduler employed for governing *event*-



(A) TA for lower curve γ_i^l

(B) TA for upper curve γ_i^u

(C) Scheduler for emitting events

Figure 3: Convex,concave pattern: Input curves and their TA-based implementation

emission, which is necessary since instead of binary synchronization we are using now Uppaal's concept of broadcast channels, where full synchronization among the UTAs and LTAs has to be enforced. For bounding the number of producible events each instance of an UTA and LTA periodically increments its local counter b_i as before, namely every $\delta_i^{\{u,l\}}$ time units by executing the respective edge, which we denote as clock-tick edge from now on; it is the top edge of TA template (A) or (B). The emission of an event can now only take place if the minimum-condition applies, whereas event emission is enforced if the maximum-condition holds. In any case event emission yields an update of local counter b_i which allows each UTA and LTA to track event production accordingly. Following these principles, minimum and maximum-condition are implemented as follows:

(1) Implementation of the minimum-condition has to do with location invariant $Sync = K$ defined for the target location of the *event*!-edge of the scheduler (Fig. 3) and the increment of global variable *Sync*, as done by each UTA when executing its *event*?-edge.

(2) Implementation of the maximum-condition has to do with the location invariants defined for each LTA, where the violation can only be circumvented by the scheduler executing its *event*!-edge at the respective point in time.

The usage of the unique label *event* within all TA guarantees the joint execution of the *event*!-edge of the scheduler and all *event*?-edges of the LTA and UTA, respectively. Thus either all *event*-edges are jointly executed or none, which yields the nice feature that an input generator of the above kind deadlocks if upper and lower bounding functions are not consistent. This case can easily be verified by model checking a corresponding query. Finally one may note that as before LTA i resets its local clock x_i once $b_i = 0$ and event emission takes place. UTA i does so once $b_i = BMAX_i$ holds and event emission occurs. In the following we will reason formally about the correctness and completeness of this more complex pattern.

4.1.3 Correctness and completeness of the pattern

THEOREM 1. *The input generator represents each timed event trace tr the differential counting function $R^{tr}(s,t)$ of which is bounded according to Eq. 1 with the upper and lower bounding functions γ^u and γ^l as defined by Eq. 3 and 4.*

This will be shown by proving the invulnerability of upper and lower bound (A) and by proving the completeness of the input generator w.r.t. the set of producible timed *event*-traces (B).

(A) Invulnerability of the upper and lower bound

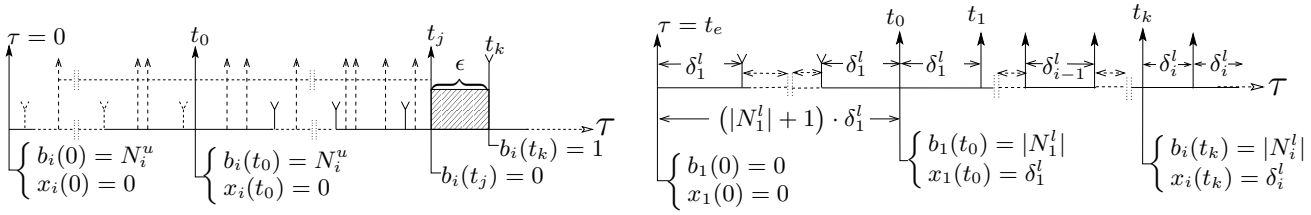
LEMMA 1. *Any trace tr producible by the input generator which is a network of fully synchronizing and parameterized UTA, is bounded by the staircase function γ^u as defined in accordance to Eq. 3 and 4.*

PROOF. According to the above discussion the parameters N_i^u and δ_i^u of an UTA i correspond to the parameters of the respective step-function γ_i^u. It is easy to see that *UTA i* allows the production of at most $N_i^u + \left\lfloor \frac{\Delta}{\delta_i^u} \right\rfloor$) events and that with $b_i = 0$ it blocks event production. Minimum-condition as defined above gives, that for any $\Delta \in \mathbb{R}^{\geq 0}$ the max. number of events producible is bounded by $\min_i(N_i^u + \left\lfloor \frac{\Delta}{\delta_i^u} \right\rfloor)$. It is important to note that for $b_i(t_0) = |N_i^u|$ an event generation resets clock x_i, such that the above equation yields an upper bound for the number of producible events. This is exactly what was defined for γ_u in Eq. 3 and 4. □

For exemplification please refer to the graph of Fig. 3: for intervals of size t and with y events produced UTA 3 blocks event production until its clock x expires. From now on event production is bounded by γ_3^u (green curve) due to the minimum-operation realized by synchronizing the UTA.

LEMMA 2. *The network of LTAs enforces the generation of events s. t. for any trace tr producible by the input generator $R^{tr}(s,t) \geq \gamma^l(t-s)$ for $0 \leq s \leq t$ holds with $R^{tr}(s,t)$ as the differential event-counting function of tr and $s,t \in \mathbb{R}^{\geq 0}$.*

PROOF. Due to the maximum-operation it seems appropriated to argue over the index of the LTA currently enforcing the generation of events and the size of the intervals:



(A) Trace generated by UTAs (B) Trace enforced by LTAs

Figure 4: Timed *event*-traces and evaluations of $b_i(\tau)$ and $x_i(\tau)$

- $0 \leq \Delta < (|N_1^l| + 1) \cdot \delta_1^l$: For intervals of this length event emission does not need to be enforced. Thus by choosing the parameters of LTA 1 accordingly the input generator is capable of delaying event emission up to $(|N_1^l| + 1) \cdot \delta_1^l$ time units, since starting with $b_1 = 0$ it is exactly this amount of time which it takes for LTA 1 to reach its event generation threshold ($b_1 = BMAX_1$ with $BMAX_i = |N_i^l|$) and the local clock x_1 expiring ($x_1 = Delta_1$, with $Delta_i = \delta_i^l$) given that no event has been emitted before. This implies also why clock x_1 needs to be reset when emitting an event in case of $b_1 = 0$ (cf. Fig. 3(A))
- $(|N_k^l|+1) \cdot \delta_k^l \leq \Delta < (|N_{k+1}^l|+1) \cdot \delta_{k+1}^l$: For each interval of this length LTA k bounds the minimum number of emitted events to $N_k^l + \left\lfloor \frac{\Delta}{\delta_k^l} \right\rfloor$, with $N_k^l \leq 0$. This is because when holding its threshold ($b_k = BMAX_k = |N_k^l|$) and with the local clock x_k expiring ($x_k = \delta_k^l$) LTA k enforces the generation of an event and form now on every δ_k^l time units (see location invariant of the UTA of Fig. 3(A)). This goes on until LTA $k+1$ holds its threshold $BMAX_{k+1}$ and its local clock x_{k+1} expires.
- $(|N_{k+1}^l| + 1) \cdot \delta_{k+1}^l \leq \Delta < \infty$: For intervals of this size we may use the same argument as above, but now starting with LTA $k+1$.

The last question to think of is whether states where $b_i = |N_i^l|$ and $x_i = \delta_i^l$ holds are reachable. Given the initial configuration with $b_i = 0$ and for $|N_{i-1}^l| < |N_i^l|$ and $\delta_{i-1}^l > \delta_i^l$ this is evident. □

For exemplification refer to Fig. 4(B) which illustrates a trace produced by the LTAs. Let event generation take place at time t_e and let $b_1(t_e) = 0$ afterwards s. t. clock x_1 is set to zero. After another $(|N_1^l| + 1) \cdot \delta_1^l$ time units where no event emission took place $b_1 = |N_1^l|$ and $x_1 = \delta_1^l$ will hold which immediately enforces event generation (here at time t_0). From now on this is done at least every δ_1^l time units, until $b_2 = |N_2^l|$ and $x_2 = \delta_2^l$ holds which enforces event production now every δ_2^l time units. In general this means that once started LTA $i-1$ enforces the event generation every δ_{i-1}^l time units. This goes on until $b_i = |N_i^l|$ and $x_i = \delta_i^l$ holds, which forces LTA i to take over event production, e. g. at time t_k as shown in Fig. 4(B).

(B) Completeness of the pattern

This issue needs only to be discussed w. r. t. the UTA; the completness w. r. t. γ^l is already contained in lemma 2.

LEMMA 3. *The network of co-operating UTAs is capable of producing any trace tr the cumulative function of which is bounded by γ^u.*

This will be shown by contradiction.

PROOF. One may assume that there is a timed trace tr bounded by γ^u, but it can not be produced by the network of UTAs. From this it follows that there must be a timed action (t, e) where the UTAs are blocked, but the production of an additional event would not be contradictory to γ^u. Let $(t, e) \in tr$ but $(t, e) \notin tr^{TA}$, where tr^{TA} is a trace producible by the input generator with the same prefix as tr a priori to the occurrence of (t, e). Let $t := t_j + \epsilon$, it must hold that there $\exists b_i \in \mathcal{B}$, s.t. $b_i(t_j + \epsilon) = 0$, otherwise one would be able to produce an event. Let t_j be now the earliest point in time for tr where $b_i(t_j) = 0$ holds and let $t_0 < t_j$ be the last point in time where $b_i(t_0) = N_i^u$ was satisfied, which is exactly what we have illustrated in Fig. 4. The choice of i and the blocking of events implies now that for the blocking period ϵ it must hold $t_j + \epsilon < t_k$, where t_k is the next time b_i is incremented and the generation of an event would not be blocked anymore. From this it follows that for the number of events $R^{tr}(t_0, t_j + \epsilon)$ seen on tr in the interval $[t_0, t_j + \epsilon]$ it must hold:

$$R^{tr}(t_0, t_j+\epsilon) = N_i^u + \left\lfloor \frac{(t_j + \epsilon - t_0)}{\delta_i^u} \right\rfloor + 1 = \gamma_i^u(t_j+\epsilon-t_0)+1,$$

otherwise b_i would not be 0. Obviously this violates the bound imposed by γ^u, since at time point t_j the number of events is bounded by γ_i^u which is the current minimum and truly smaller than $R^{tr}(t_0, t_j + \epsilon)$. Thus from $R^{tr}(0, t_j + \epsilon) > \gamma^u$ we can conclude that such a trace tr does not exist. Concerning the assumption that t_0 was the last point in time where $b_i(t_0) = N_i^u$ was satisfied and that for t_k: $b_i(t_k) = 0$ must hold, one may note that for the initial point in time we have $b_i(0) := N_i^u$ and that b_i must be zero at t_j, since otherwise UTA would not block event emission by violating the minimum-condition. □

4.1.4 Extended input generators

In practice systems may not show strictly concave or convex patterns. For example, the overall upper input or output curves may have decreasing step widths, or the lower curve may be characterized by increasing ones. Due to space restrictions we will briefly indicate how one can resolve such situations when coupling RTC-conformant curves with down-streamed TA.

In cases where non-concave and non-convex patterns occur only finitely often within an arriavl curve, one can handle this by simply making use of subsets of UTA, LTA and local synchronization for obtaining local minima and maxima. For implementing this one solely needs to encapsulate the respective sets of co-operating LTA or UTA in their own sub-system. These subsystems can be implemented analogously to the pattern illustrated above, but requiring slightly adapted TA-specifications w. r. t. the employed thresholds.

For RTC-conformant curves with periodic repetition of non-

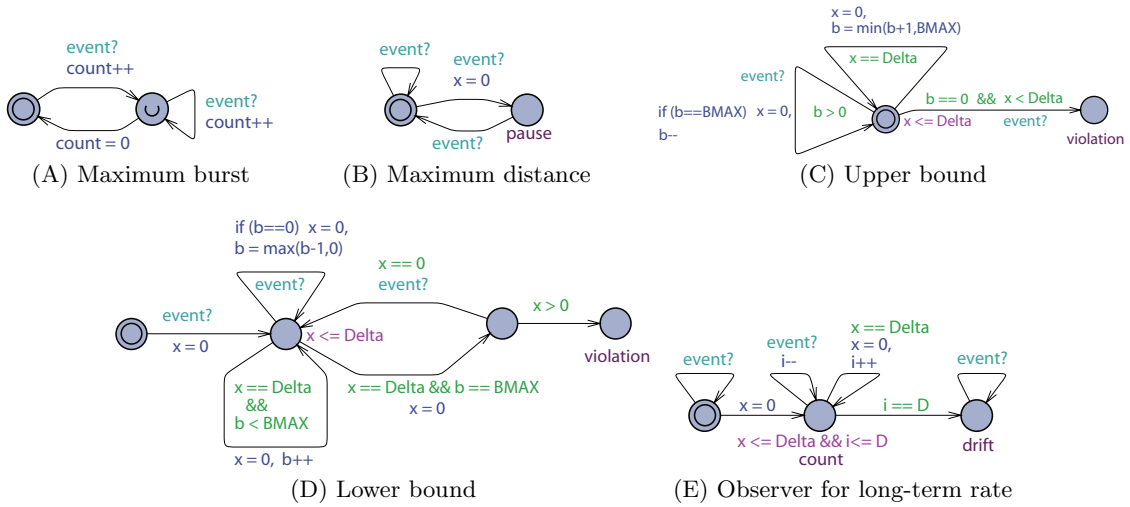


Figure 5: Observer automata for deriving upper and lower output curves

concave and non-convex patterns, one may employ an UTA, LTA which implement the upper and lower long term rate of the respective streams.

4.1.5 Complexity issues related to input modelling

Complexity of model checking TA is exponentially bounded by the number of clocks and clock constants [1]. Thus it is straight forward to see that the efficiency of the approach is closely related to the number of staircase functions employed for modeling lower and upper input curves.

In the following we propose a simple method that permits to approximate a general arrival curve α with the convex/concave combination of just a few staircase functions. The approach first approximates the arrival curve by a so called periodic with jitter event arrival model, and then derives the parameters for the corresponding staircase curves $\gamma_i^{\{u,l\}}$. The periodic with jitter event model (or PJD model in short) is commonly used in literature and is a simple representation for the arrival of events in streams [19]. It is specified by a parameter triple (p, j, d), where p denotes the period, j the jitter and d the minimum inter-arrival time of events in the modeled stream.

Arrival curves are in general more expressive than PJD models. However, every arrival curve can be conservatively approximated by a PJD model [12]. Given an arrival curve α to feed into a TA-based component, we first use the algorithm described in [12] to approximate α with a PJD model. Subsequently, we convert the PJD parameters to a set of appropriate parameters $N_i^{\{u,l\}}$ and $\delta_i^{\{u,l\}}$ that are used to specify the input generator for the TA-based component as described in Section 4.1.2.

The upper bound described by a PJD model can be represented by the minimum of at most two staircase functions γ_1^u and γ_2^u. In particular, two staircase functions are needed if $d > 0 \ \wedge \ d > p - j$, while only one staircase function suffices otherwise. For the lower bound of a PJD model one staircase function γ^l is always sufficient. The parameters of the staircase functions are computed as follows:

- Case $d = 0 \ \vee \ d \leq p - j$:
  $N^u := \left\lceil \frac{j}{p} \right\rceil + 1; \quad N^l := -\left\lceil \frac{j}{p} \right\rceil; \quad \delta^u := \delta^l := p$
- Case $d > 0 \ \wedge \ d > p - j$: $N_1^u := 1; \quad \delta_1^u := d;$
  $N_2^u := \left\lceil \frac{j}{p} \right\rceil + 1; \quad N^l := -\left\lceil \frac{j}{p} \right\rceil; \quad \delta_2^u := \delta^l := p$

Note that an exact representation of a PJD model by means of staircase functions γ_1^u, γ_2^u and γ^l is not always possible under the assumptions that the parameters $N_i^{\{u,l\}}$ are integers and that at each event arrival the counters b_i are decreased by one unit: In such a representation the first step of a staircase function has always the same width as all following steps, or in other words, we cannot horizontally shift the staircase functions $\gamma_i^{\{u,l\}}$ with respect to each other. However, in such cases the staircase curves derived by means of the above formulae will still provide a conservative approximation of the PJD model. If we generalize the assumption such that at each event arrival the counters b_i are decreased by e_i units, where e_i is a positive integer, an exact representation of any PJD model by means of at most three staircase functions is always possible. For the sake of conciseness we omit the formulae for the parameters of γ_1^u, γ_2^u and γ^l for this more general case.

While the approximation of arrival curves with PJD models represents a simple way to coarsely bound an event stream with few staircase functions, in the presented hybrid analysis approach the interface between RTC and TA is of course not limited to PJD curves. Any other algorithm that correctly bounds an arrival curve α with an arbitrary number of staircase functions $\gamma_i^{\{u,l\}}$ can be used as an interface between the two domains.

Now that we have described the interfacing from RTC-based model descriptions to TA, we will discuss the interfacing from TA-systems back to RTC-conformant performance models.

4.2 Obtaining output curves from TA systems

We extract a set of staircase functions $\gamma_i'^u$ and $\gamma_i'^l$ which allow to construct an overall output curve $\gamma'^{\{u,l\}}$ by means of minimum and maximum operators, respectively. For achieving this goal, we couple the system under analysis including the input generator with a set of observing TA. Checking reachability queries for these TA-systems allows to derive the parameters $N_i^{\{u,l\}}$ and $\delta_i^{\{u,l\}}$ that uniquely characterize $\gamma_i'^u$ and $\gamma_i'^l$. In the following we will first describe the TA that are used to verify individual staircase parameters. After that, we will describe the overall composition strategy for constructing a valid output curve $\gamma'^{\{u,l\}}(\Delta)$.

For implementing the above procedure we will employ the observers TA depicted in Fig. 5:

(a) Maximum burst size: An upper bound for the maximum number of events that the system can generate simultaneously can be verified by means of the observer depicted in Fig. 5(A) and the query[1]: `A[] (count<=estimate)`.

(b) Maximal distance between two successively emitted events: We can verify a bound on the maximum pause time between two output events by employing the observer shown in Fig. 5(B) and the query `A[] (pause imply x<=estimate)`.

(c) Arbitrary upper staircase curve $\gamma_i'^u$: For obtaining an individual staircase function we employ the observer TA of Fig. 5(C) which witnesses the violation or invulnerability of the respective curve. The witnessing TA moves into the location `violation`, once the respective curve is violated, i.e. the actual system produces too many events. Thus one simply needs to query for the reachability of location `violation` (`A[] (not violation)`). In other words, given some staircase parameters N_i^u and δ_i^u, we can determine whether the corresponding staircase function is a valid upper bound in time-interval domain of the produced event stream.

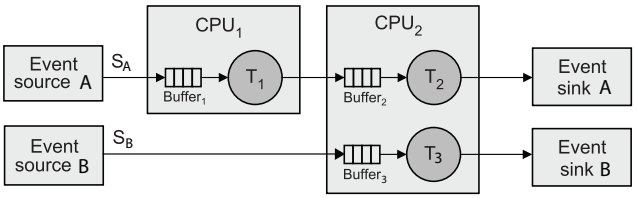
(d) Arbitrary lower staircase curve $\gamma_i'^l$: For obtaining an individual lower staircase function we employ observer TA of Fig. 5(D) and use the same principle as described above: Given some parameters N_i^l and δ_i^l, we can determine whether the corresponding staircase function is a valid lower bound in time-interval domain of the produced event stream.

(e) Long-term rates: In order to construct output curves $\gamma'^{\{u,l\}}$ that approximate the system behavior well also for large time intervals, we need to make sure that we follow the long-term event output rate. The largest δ_i^u and the smallest δ_i^l of *any* valid upper and lower output staircase function, respectively, denote upper and lower bounds on the long term rate of the output. The principle of efficiently verifying that a given staircase function represents this upper or lower bound will be explained by means of $\gamma_i'^u$. The procedure for $\gamma_i'^l$ is analogous and is omitted for conciseness. We will verify that for all intervals the difference between the maximum number of events allowed by $\gamma_i'^u$ and the maximum number of events producible by the system is bounded by a constant `D`. If this is the case then a tight upper bound on the long-term rate of the system is found. To do so one may employ the observer depicted in Fig. 5(E). In this automaton the variable `i` (initialized with N_i^u) tracks the difference between $\gamma_i'^u$ and the system output. If there is a trace that never reaches the location `drift`, it means that $\gamma_i'^u$ represents a tight upper bound for the long-term rate. Such a trace can be found as counterexample to the query: `count -> drift`.[2]

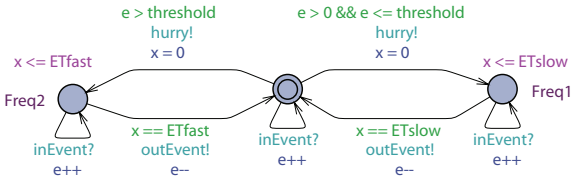
Now, we will describe how the TA introduced above can be used to construct valid upper and lower bound curves $\gamma'^{\{u,l\}}(\Delta)$ of the system output. There are many possibilities and the following list summarizes only a few of them, especially those used in the Case Study:

- A binary search on `estimate` (see (a) and (b)) yields the maximum burst size and the maximal pause time, respectively.

[1]In Uppaal `A[]` stands for 'always invariantly'.
[2]In Uppaal `->` stands for 'always eventually leads to'.



(A) System architecture



(B) TA model for CPU_1

Figure 6: Layout and TA-based submodel

- These values can be used to determine one degree of freedom (of the available two) of an upper or lower staircase automaton, see (c) and (d). For example, using the maximal burst size from (a) and a binary search on the remaining parameter δ_i^u in the automaton of (c) yields an upper staircase function $\gamma_i'^u$ that characterizes the maximal burst rate.

- Fixing any of the two free parameters in the automata of (c), (d) and performing a binary search on the other yields a valid upper and lower staircase bound, respectively.

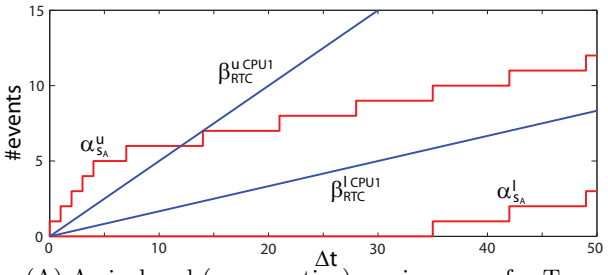
- Using the automaton of (c) with a large initial burst capacity N_i^u and performing a binary search on δ_i^u leads to a tight upper bound on the maximal long term rate.

- One may determine a convex hull of the lower and a concave hull of the upper (unknown) RTC-curves of the event stream by calculating a sequence of staircase functions. For example in case of an upper curve, we start with the curve that characterizes the maximal burst rate. The interval Δ where it just matches the system behavior can be determined using a counterexample construction. This point is then the starting point of the next staircase function. The sequence ends if the long term rate is met, see (e).

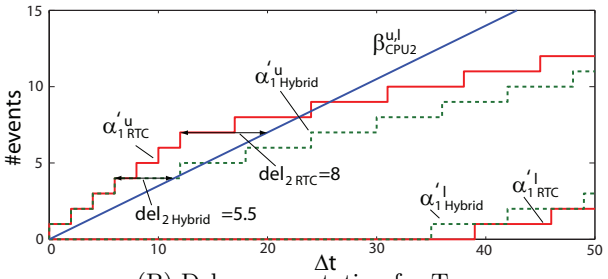
- All constructed valid upper and lower staircase functions can be combined to a valid bounding curve by minimum and maximum operations, respectively.

5. CASE STUDY

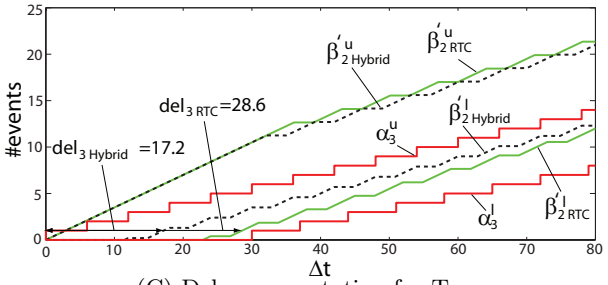
The considered system is shown in Fig. 6(A). It consists of three tasks T_1, T_2 and T_3 that run on two distinct processors CPU_1 and CPU_2. The three tasks process two incoming event streams S_A and S_B which are periodic streams with large jitters that lead to bursts. S_A and S_B are specified by the parameter triples $p_A = 7$ms, $j_A = 28$ms, $d_A = 1$ms and $p_B = 7$ms, $j_B = 23$ms, $d_B = 6$ms, respectively. CPU_2 implements a preemptive fixed-priority scheduling policy with T_2 having higher priority than T_3. The execution of each task on its respective CPU takes 10^6 cycles. CPU_2 operates at a constant frequency of 350MHz. CPU_1 implements a load-dependent frequency adaptation. In particular, it operates at 166MHz if there are less than 4 events in its input



(A) Arrival and (conservative) service curve for T_1



(B) Delay computation for T_2



(C) Delay computation for T_3

Figure 7: Curves associated with the case study

buffer, and at 500MHz otherwise. The goals of the performance analysis are to characterize the event output of T_1, to determine the maximum delays and backlogs that events can experience at the single tasks, and to find the maximum end-to-end delay for stream S_A. In this case study we will compare three different approaches: First we analyze the described system with the abstraction of RTC only using the MPA Toolbox [13, 19]. Subsequently, we carry out the analysis with the presented hybrid analysis approach, where we model the state-dependent behavior of CPU_1 as TA and analyze CPU_2 with RTC. Finally, we verify the performance of the system by means of a dedicated TA model according to the method described in [9], which permits to exploit the simple periodic nature of the input streams.

For the *hybrid analysis approach*, we first represent the input stream S_A by the combination of three staircase functions γ_1^u, γ_2^u and γ^l. Using the equations of Sec. 4.1.5 we get the parameters $N_1^u := 1$, $\delta_1^u := 1$, $N_2^u := 5$, $\delta_2^u := 7$, $N^l := -4$ and $\delta^l := 7$ for the staircase functions. The corresponding event curve α_{S_A} is shown in Fig. 7(A). Given these parameters we automatically create the input generator as described in Sec. 4.1.2. In order to increase the efficiency of the analysis, we merge the input generating network of TA into a single automaton and simplify it slightly by considering that $N_1^u = 1$, that is, for γ_1^u we do actually not need a counter variable b, but just a clock to enforce a minimum distance δ_1^u between consecutive events. This input generator is then coupled with the automaton shown in Fig. 6(B) which models the load-dependent behavior of CPU_1. In this automaton we use the signals *inEvent* and *outEvent* to distinguish between ingoing events coming from the Event Source A and outgoing events sent to T_2. *Buffer1* of CPU_1 is modelled by means of a local counter variable *e*. The two locations *Freq1* and *Freq2* represent the processing of events at low and high frequency, respectively, with corresponding processing times *ETslow* and *ETfast*. The signal *hurry* belongs to an urgent channel which is always ready for synchronization. This construct enforces greedy event processing. At this point we apply the heuristic of Sec. 4.2 to get arrival curves that bound the output of the TA subsystem, where we choose to represent the upper bound as the minimum of three staircase functions and the lower bound with just one staircase function. The resulting pair of arrival curves is then used as input for the RTC analysis of CPU_2. For the analysis of the maximum delay on CPU_1 in the hybrid setting, we customize the automaton of Fig. 6(B) following the ideas of [9].

	Max delay [ms]				Max buffer [events]		
	T_1	T_2	T_3	EE_A	T_1	T_2	T_3
RTC	29	8	28.6	31.9	5	3	5
Hybrid	25	5.5	17.2	30.5	5	2	3
TA	25	4.6	14.3	27.9	5	2	3

Table 1: Results of Performance Analysis

	RTC	Hybrid	TA
Total run-time	< 1s	11min	1h

Table 2: Run-times for Performance Analysis

Table 1 summarizes the results of the performance analysis. The worst-case end-to-end delay of stream S_A is denoted as EE_A. Note that in general for a sequence of components the worst-case end-to-end delay can be smaller than the sum of the individual worst-case delays. While in the abstractions of RTC and TA this phenomenon can be captured for EE_A, this is not possible in the hybrid approach.

As can be seen in the table, in terms of accuracy the *hybrid approach* is clearly better than the pure RTC analysis. In particular, the conservativeness of the results is highly reduced, with a maximum delay and backlog at T_2 that are 31% and 33% lower with respect to the RTC analysis, respectively. For the delay and the backlog at T_3, the hybrid approach achieves values that are 40% lower compared to the pure RTC analysis.

The reason for the better results can be understood by looking at Fig. 7, where we employed the RTC-related labeling, i.e. the α-curves refer to input streams of events, the α'-curves to their outgoing counterparts and the β- and β'-curves to the ingoing and outgoing streams of available resources, respectively. A *pure RTC-based analysis* of the above scenario cannot capture the load-dependent behavior of CPU_1. Hence, one has to assume that the processor always operates at 500MHz in the best case and at 166MHz in the worst case. This assumption corresponds to using the service curves β_{RTC}^{uCPU1} and β_{RTC}^{lCPU1} (cf. Fig. 7(A)) for the analysis of CPU_1. This yields conservative worst-case processing load predictions captured by $\alpha'^{u}_{1,RTC}$ for T_2. However, a TA-based analysis of CPU_1 produces tighter input bounds captured by $\alpha'^{u}_{1,Hybrid}$ for the RTC analysis of T_2. This leads to smaller worst case delay guarantees, as shown in Fig. 7(B) and 7(C).

The last line of Table 1 contains the exact values for the worst-case performance of the system. These values are determined by means of the *dedicated TA model for the entire system*. As can be seen in the table, the results for the hybrid analysis approach are slightly more conservative. The reason is that the concave (convex) hull determined as bound for the output event stream of T_1 does slightly over- (under)-approximate the real behavior of the system. The graphs of Fig. 7 do not show arrival and service curves for the exact internal behavior of the system, as these interfaces are not intrinsic to the dedicated TA model.

The higher degree of accuracy of the hybrid analysis method in comparison to the pure analytic RTC approach has its price, namely a substantially longer run-time, as can be seen in Table 2. This becomes worse if one keeps in mind that we already decided to bound the output curves by a convex (concave) pattern of three staircase functions only. In case of requiring a higher degree of accuracy one needs to adapt the proposed scheme in order to detect non-convex and non-concave patterns and its additional staircase functions. But this once again comes along with clearly higher computation times. Nevertheless, the run-times achieved for the hybrid approach are still significantly better compared to the verification of the pure TA model.

6. CONCLUSION

In this paper we developed a hybrid analysis methodology by coupling analytic (state-less) RTC- and state-based TA analysis. The presented technique is based on the observation that the stream abstractions via RTC-curves can be obtained by composing individual staircase functions linked through minimum and maximum operations. When constructing TA representations of RTC-curves, the developed technique guards each staircase function by its own TA, where the building of maximum and minmum is implemented by synchronizing the respective groups of TA. In the reverse direction, the parameters of tight staircase functions must be found by employing the respective observer TA in a binary search based algorithm.

The proposed methodology limits state space explosion as intrinsic to formal verification to the level of isolated (sub)-components, since losely coupled TA-based component descriptions can be analyzed in isolation, combined by interfacing the respective RTC-conformant output and input curves (cf. Fig 1). For maintaining scalability state-based analysis should only be applied for those components, where RTC-based analysis delivers overly pessimistic results. As demonstrated by the case-study such cases are found when dealing with components showing strict state-dependent behaviours. Overall such a strategy will thereby avoid to pessimistically over-approximate system's overall performance metrics, but still maintaing scalability of the proposed methodology.

As the RTC-based stream abstractions are more general than the widely used PJD (periodic with jitter) abstractions, the proposed method also leads to hybrid methods that combine TA-based timing verification of single components with classical real-time analysis, e.g. MAST [8] or Symta/S [10]. In particular, PJD-models can directly be modeled using 3 staircase functions.

Acknowledgement: This work is funded by the European Union project COMBEST under grant number 215543.

7. REFERENCES

[1] R. Alur and D. L. Dill. Automata For Modeling Real-Time Systems. In M. Paterson, editor, *Proc. of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 322–335. Springer, 1990.

[2] G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.

[3] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, volume 3098 of *LNCS*, pages 87–124. Springer, 2004.

[4] J.-Y. L. Boudec and P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050 of *LNCS*. Springer, 2001.

[5] S. Chakraborty, S. Künzli, and L. Thiele. A General Framework for Analyzing System Properties in Platform-Based Embedded System Designs. *Design, Automation and Test in Europe Conference and Exhibition*, 1, 2003.

[6] S. Chakraborty, L. T. X. Phan, and P. S. Thiagarajan. Event count automata: A state-based model for stream processing systems. In *Proc. of the 26th IEEE International Real-Time Systems Symposium (RTSS'05)*, pages 87–98, 2005.

[7] H. Dierks, A. Metzner, and I. Stierand. Efficient Model-Checking for Real-Time Task Networks. In *Int. Conf. on Embedded Software and Systems 2009*, 2009. Accepted for publication.

[8] M. González Harbour, J. J. Gutiérrez García, J. C. Palencia Gutiérrez, and J. M. Drake Moyano. Mast: Modeling and analysis suite for real time applications. In *Proc. of 13th Euromicro Conference on Real-Time Systems*, pages 125–134. IEEE Computer Society, 2001.

[9] M. Hendriks and M. Verhoef. Timed automata based analysis of embedded system architectures. In *Proc. of the 20th Int. Parallel and Distributed Processing Symposium (IPDPS 2006)*. IEEE, 2006.

[10] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System Level Performance Analysis-The SymTA/S Approach. *IEEE Proceedings-Computers and Digital Techniques*, 152(2):148–166, 2005.

[11] P. Krcal, L. Mokrushin, and W. Yi. A tool for compositional analysis of timed systems by abstraction (extended abstract). In *Proc. of NWPT07, the 19th Nordic Workshop on Programming Theory*, October 2007.

[12] S. Künzli, A. Hamann, R. Ernst, and L. Thiele. Combined Approach to System Level Performance Analysis of Embedded Systems. In *Proc. of the 5th Int. Conf. on Hardware/Software Codesign and System Synthesis 2007*, pages 63–68, New York, NY, USA, 2007. ACM.

[13] Modular Performance Analysis Framework and Matlab Toolbox. www.mpa.ethz.ch.

[14] C. Norström, A. Wall, and W. Yi. Timed automata as task models for event-driven systems. In *Proc. of the 6th Intl. Conference on Real-Time Computing Systems and Applications*, page 182. IEEE Computer Society, 1999.

[15] L. Phan, S. Chakraborty, and P. Thiagarajan. A Multi-mode Real-Time Calculus. In *Proc. of the 28th IEEE Real-Time Systems Symposium (RTSS 2008)*, pages 59–69. IEEE Computer Society, 2008.

[16] L. T. X. Phan, S. Chakraborty, P. S. Thiagarajan, and L. Thiele. Composing functional and state-based performance models for analyzing heterogeneous real-time systems. In *Proc. of the 28th IEEE Real-Time Systems Symposium (RTSS 2007)*, pages 343–352. IEEE Computer Society, 2007.

[17] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proc. Intl. Symposium on Circuits and Systems*, volume 4, pages 101–104, 2000.

[18] The Uppaal timed model checker. www.uppaal.com.

[19] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System Architecture Evaluation Using Modular Performance Analysis: A Case Study. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(6):649–667, 2006.