Analytic Real-Time Interfaces for state-based Components

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

TIK Report 327: ftp.tik.ee.ethz.ch/pub/publications/TIK-Report-327.pdf

August 13, 2010

ABSTRACT

Interface theories provide the foundation for formally defining and exploiting often implicitly made assumptions about the environment a component is interacting with. This paper develops a procedure for computing analytic input/output bounds for event streams consumed/emitted by the individual components of a system design, allowing to define state-less assume/guarantee (A/G) real-time interfaces for each component. However, contrary to existing work this paper extends interface definitions with properties to be met invariantly by any component that implements it. This allows to compute bounds of key performance metrics of the overall system design by solely considering the information provided by the enriched interfaces. This is important as it supports incremental, i. e. component-wise evolution of system designs for the following reason: given a consistent interface-based system description the interface-derived properties are invariant w. r. t. composition and substitution of components, as long as each state-based component implementation is conformant to its interface and the interfaces are compatible. This paper establishs the required consistency and conformance criteria and develops the machinery for carrying out the checks in an automatic fashion. Thereby it advocates a strictly compositional design approach and improves the scalability of state-based analysis methods.

1. INTRODUCTION

1.1 Motivation and Contribution

Compositionality appears to be a helpful paradigm for coping with complexity. Hence it could be the silver bullet when designing, analyzing and implementing embedded systems with real-time constraints. However, in the context of state-based analysis methodologies, e. g. real-time UML state charts, Timed Petri-Nets or Timed Automata Timed, to name only few of them, component interaction commonly prohibits a compositional analysis of the overall system design. For limiting component's mutual interference and maintaining compositionality we restrict component interaction to streams of uniform, discrete activity-triggers. Such event streams can be abstractly characterized by so called event arrival curves [11] or event models [9]. This paper employs arrival curves for abstractly capturing component behaviour, as the curves bound all streams of activty-triggers possibly consumed or emitted by a component. This implicitly defines what kinds of traffic patterns the component is willing to accept and, on the level of outputs, defines what kinds of streams the component gurantees to emitt. Thus, these abstract stream descriptions can be interpreted as so called state-less, assume/guarantee (A/G), real-time interfaces [6, 7, 12, 13]. With such an interface one formally defines (a) what a component expects from its environment, denoted as input assumption and (b) what a component guarantees w. r. t. its outputs, denoted as output guarantee. But, contrary to existing work, this paper not only derives such state-less A/G interfaces for state-based component implementations such as Timed Automata (TA) [2], it also includes properties to be met invariantly by any component implementing this interface. These enriched interfaces allows one to assert properties to be met by the overall system design, rather than asserting this for a given implementation or an abstract model thereof. Examples of such interface-derived measures are the overal delay expirenced by event triggers when travelling thorough the system, or the buffer spaces required by a component. With this strategy for computing dedicated key performance meassures of overall system design from interface deefinitions, we support incremental, i. e. component-wise evolution of system designs. This is for the following reason: provided that the interface-based system description is consistent, the interface-derived properties of the overall system are invariant w. r. t. composition and substitution of components, as long as each TA-based component implementation is conformant with its interface and the interfaces are conformant. In a nutshell, this paper establishs the required consistency and conformance criteria and develops the machinery for carrying out the checks in an automatic fashion. Thereby it targets a rigoros compositional design approach and improves the scalability of state-based analysis methods, which can be an opportunity for integrating formal methods in industrial design flows. For achieving this the paper is organized as follows:

Sec. 3 presents a scheme for testing the conformance of interfaces and TA-based component implementations. It introduces also a scheme for deriving interfaces form TA-based component implementations.

Sec. 4 develops (a) a formal criterion for deciding whether an interface-based system description is consistent, (b) it shows how this criterion can be used for checking conformance of interfaces and component implementations and (c) it is proven that this consistency is sufficient for guaranteeing invariance of interface-defined properties of the overall system design w. r. t. composition and substitution of a component; provided that the components are conformant ith their interfaces and the interfaces are compatible.

Sec. 5 presents an empirical evaluation of the framework. This is done (a) for demonstrating the efficiency of the proposed procedures, (b) for demonstrating that the interfacing of state-less and TA-based components allows for higher accuracy w. r. t. a systems performance metrics when compared to the pure analytic approach as developed in [12, 5] and (c) for demonstrating its scalability w. r. t. standard state-based analysis methodologies.

1.2 Related Work

The authors of [6] established the foundation of interface theory, but explicitly excluded timed behavior from their elaboration. The theory on timed (state-based) interfaces was provided in [7]. The work presented in this paper elaborates on state-less A/G real-time interfaces as developed in [12] in the context of a Real-time Calculus-based [11] performance methodology. The proposed approach employs the TA-based pattern presented in [10], which allows to implement functions of the Real-time Calculus (RTC) [11] as sets of cooperating TA. It is this pattern which allows us here to develop a scheme for computing state-less assume/guarantee (A/G) interfaces for component models which are defined as TA. We will not only exploit a similar notion for interface descriptions but will also employ the same "inclusion" criterion as [12, 5] for deciding wether interfaces are compatible and can therefore safely be composed.. However, this paper also provides a proof that the "inclusion" property suffice for deducing invariance of an consistent interfaced-based system description w. r. t. composition and substitution of interfaces, component resp.. In particular, this includes TA-based component implementations, where the authors of [12, 13, 5] only considered the state-less setting on the basis of RTC.

2. BACKGROUND THEORY

2.1 Streams and their abstract representation

(A) Timed (Event) Traces and Streams. A timed event is a pair (t, τ) where τ is some event label or type and $t \in \mathbb{R}_{\geq 0}$ some non-negative time stamp. A timed trace $tr := (t_1, \tau); (t_2, \tau); \ldots$ is a sequence of timed τ-events ordered by increasing time stamps, s. t. $t_i \leq t_{i+1}$ for $i \in \mathbb{N}$ and $\tau \in Act$ with Act as a set of event labels or event types.
Given a trace tr one may apply a filter-operation which for a given event type $e \in Act$ removes all pairs (t_i, τ) from tr if $\tau \neq e$ holds. A filtered trace addressed by tr_e may also be denoted as (event) stream. A set of traces is denoted $Traces$ and $Traces_e$ refers to the set of streams retrieved from $Traces$ by applying filtering operation $filter$ w. r. t. event type e and for all $tr \in Traces$.
This paper mainly considers the event labels $inEvent$ and $outEvent$, where $tr_{inEvent}$ and $tr_{outEvent}$ refers to the stream extracted from $Traces$. However, as the event type is often clear from the context, we can safely omitt it.

(B) Event Arrival Curves. Let tr be a stream, which is a trace filtered w. r. t. to some event type. A stream can be abstractly characterized by a pair $(\alpha_{in}^{low}, \alpha_{in}^{up})$, which is a pair of so called arrival curves [11, 13]. These arrival curves bound the number of events of the respective type, seen on tr for any interval $\Delta \in [0, \infty)$. Let $R_{tr}(t)$ denote the number of events that arrived on the stream tr in the time interval $[0, t)$, upper and lower arrival curve satisfy the following equation

$$\alpha_{in}^{low}(t-s) \leq R_{tr}(t) - R_{tr}(s) \leq \alpha_{in}^{up}(t-s) \qquad (1)$$

for all $0 \leq s \leq t$.

Remark: As each event from a stream may trigger behaviour within a component, arrival curves provide abstract lower and upper bounding functions α_{in}^{low} and α_{in}^{up} w. r. t. the ressource demand experienced within time interval $\Delta := t - s$. Furthermore, one may note that for a given pair α_{in}^{low} and α_{in}^{up} there might be a (possibly infinite) set of streams of computation stimuli, namely all streams the counting function R_{tr} of which satisfies Eq. 1. In the following we will use the following denotations w. r. t. event arrival curves:

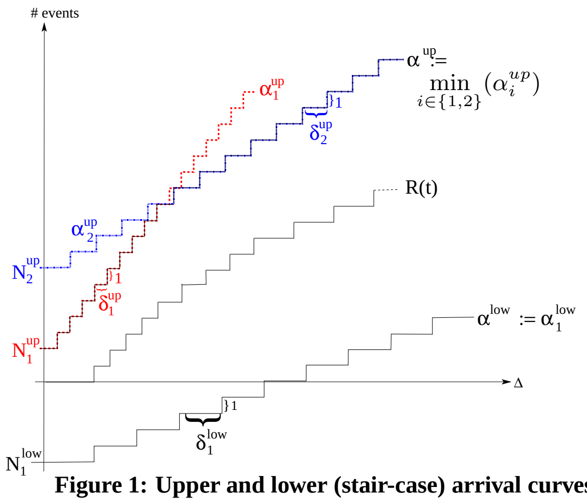


Figure 1: Upper and lower (stair-case) arrival curves

- For simplicity a pair of upper and lower arrival curves will often be addressed as arrival curve α_{in} which in fact is the pair $(\alpha_{in}^{low}, \alpha_{in}^{up})$.
- A stream for whose cumulative counting function the above equation holds is denoted as being bound by α_{in}.
- As each stream refers to a specific event type, so does it bounding arrival curve, this we will employ latter for mapping curves to component ports.
- Curve $\alpha_\epsilon := (\infty, 0)$ addresses the trivial arrival curve which contains all possible event streams, ranging from the empty stream up to a stream with infinite bursts.
- Single curves $\alpha_{\{i,j\}}$ are comparable if α_i *op* α_i holds for $\Delta \in [0, \infty)$ and with *op* $\in \{\leq, \geq\}$. Otherwise the curves are denoted as incomparable.
- In the discussion to follow we require to compare arrival curves, where these refer always to the same type of event. For making this explicit we say that the arrival curves are *event-compatible*.

DEFINITION 1 *(Curve inclusion)*: Let $\alpha_1 := (\alpha_1^{up}, \alpha_1^{low})$ and $\alpha_2 := (\alpha_2^{up}, \alpha_2^{low})$ be two compatible arrival curves. If $\alpha_1^u(\Delta) \leq \alpha_2^u(\Delta)$ and $\alpha_1^l(\Delta) \geq \alpha_2^l(\Delta)$ holds for all $\Delta \in [0, \infty)$ one says α_1 is bounded by or included in α_2 and writes $\alpha_1 \subseteq \alpha_2$. In case neither $\alpha_1 \subseteq \alpha_2$ nor $\alpha_2 \subseteq \alpha_1$ can be established the two curves are denoted as incomparable.

Arrival curves are a generic way for characterizing standard real-time traffic models ranging from strictly periodic event arrivals over PJD-streams, i. e. traces where event arrivals are periodic with jitter and a minimum distance, up to sporadic event arrivals.

(C) Example. For exemplification one may refer to Fig. 1. Parameter N_1^{up} of the upper curve α_{in}^{up} models the (burst) capacity which is the number of events which can arrive at the same instant of time in any stream bounded from above by α_{in}^{up}. Hence δ_1^{up} of the upper curve defines the minimum distance of two events once a burst had occurred. In this realm the step width δ^{low} of the lower curve α_{in}^{low} models the maximum distance between two events in any stream bounded from below by α_{in}^{low}.

2.2 Timed Automata

Timed automata (TA) extended with variables are finite, cooperating state machines equipped with clocks and variables. Instead of states one commonly speaks of locations and instead of transitions one uses the term edges, which is useful for separating the syntactic from the semantic level. The operational semantics of TA can be characterized informally as follows: Enabled edges emanating from the currently active locations can be executed one by one which yields possibly a new set of active locations. While being in a location the values of clocks increase as (global) time progresses, where all clocks of the TA increase at the same speed. As main feature clocks can either be inspected or reset, where inspection takes place when checking the validity of location invariants or when checking the enabledness of an edge. Clock resets are executed once an enabled edge is executed. Variables are treated in a very similar manner, i. e. they can be inspected or manipulated, where manipulations take place on edge executions. When referring to the value of clock x and variable c at global clock time t the notation $x(t)$ and $c(t)$ is employed.

A state s of a cooperating network of TA $\mathcal{T}$ can be characterized by a set of active locations, which are the locations the overall system currently resides in, as well as the valuation of all of its clocks and variables. In principle this yields a potentially infinite state transition system, where transitions either refer to the execution of edges or the change in time. As clocks only evaluate to values from finitely many different intervals, induced by the clock constant employed in $ClockCons(C)$ and given that variables only take values from a finite domain, the original transition system of infinite size can be mapped to a finite quotient system $\mathcal{TS}^{\mathcal{T}}$. Each path as contained within $\mathcal{TS}^{\mathcal{T}}$ constitutes a trace, where the set of traces generated by a TA $\mathcal{T}$ is denoted $Traces^{\mathcal{T}}$. If one applies now a filtering operation w. r. t. events of type e one ends up with a set of streams which we denoted $Traces_e^{\mathcal{T}}$. One may note that for the discussion to come, it is irrelevant if paths and traces are infinite as we are only considering reachability properties of TA which can be verified in finitely many steps, namely by visiting at most all states of a TA's finite quotient transition system.

This paper exploits Uppaal's [3, 4] concept of timed safety automata and its cooperation mechanisms;. For clearness we briefly re-capitulate some important aspects:

(A) Location invariants. An edge leading from an active location a to a successor location b can only be executed if it is enabled, i. e. once it guards evaluate to true. However, this does not sufficient. Most importantly this includes validity of the invariants associated with location b, once the edge has been executed. Hence for the execution of an edge also the invariant of the down-streamed location must hold, where as location invariants clock and variable constraints can be employed.

(B) Cooperation via shared variables. A system model may feature a set of cooperating TA. As these TA may inspect or manipulate global variables, the individual TA may mutually influence their behaviors. When jointly executing synchronizing edges the execution order of the related statements is non-deterministic. Hence its is important to solely employ commutative manipulations on synchronizing edges and global variables, e. g. . increment of a global variable.

(C) Cooperation via synchronization. Uppaal makes use of channels and signals allowing to implement different rendez-vous, synchronization mechanisms resp.: TA may jointly execute their enabled edges if the involved edges carry the same (channel) identifier followed by a question or exclamation mark. The TA providing an edge labeled with a exclamation mark is commonly denoted sender, whereas the TA providing the edge with the question mark is denoted sender. Sending or receiving edges in isolation are blocked, i. e. these edges can not be executed on their own they always require the availability of an enabled counterpart. As the number of senders and receivers may vary, different rules apply:

- *Binary synchronization.* If there are n senders and k receivers the state space exploration mechanism picks non-deterministically sender/receiver pair and jointly execute their edges. As state space exploration is exhaustive this has to be done for all possible sender/receiver pairs.
- *Arbitrary, n-ary synchronization.* The sending of a broadcasting signal is non-blocking, i. e. a broadcasting sender TA executes its sending edge and between 0 and n receivers execute one of their receiving edge at the very same time. It is important to note, that each receiver with at least one enabled receiving edge has to execute the latter.
- *Full, 1:n synchronization.* Usage of global variables and broadcast channels allow one to enforce a joint execution of one sending and n receiving edges which is achieved as follows: each receiver increments a global variable. The sender requires that after executing the sending edge the valuation of this variable equals some constant, here n. Guarding the validity of this constraint is done by a location invariant assigned to the sending edge's target location.

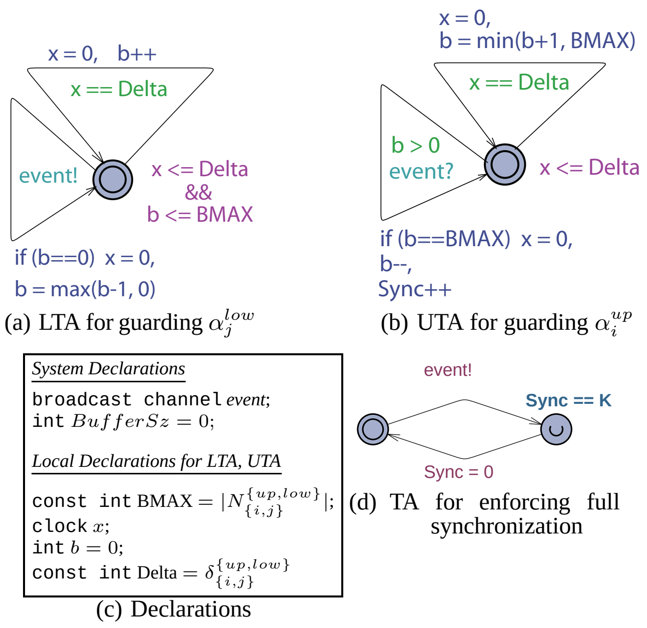
2.3 Timed Verification

Determining if for a network of TA $\mathcal{M}$ a dedicated property Φ holds is denoted as timed verification. The properties to be verified can either be associated with states, as so called state or safety properties or with traces, which are a possibly infinite sequences of states. In the reminder of this paper we concentrate on state properties only. This is justified for the following reasons: (a) this work solely elaborates on interface properties which refer to key metrics of embedded system, e. g. buffer sizes or event delays. These quantities can directly be associated with clocks or variables of a TA, s. t. each state can be labeled with an atomic proposition, the value of which depends on the fact whether the respective clock or variable satisfy a user-defined constraint or not. (b) Behavior w. r. t. some event stream is guarded by so called observer TA. As each observer is equipped with a dedicated failure location, where it transits to this location once the requested timed behavior is violated, an appropriate labeling of states is achieved, and a reachability query on the absence of the respective location identifier asserts the validity of this behavioral property.
For clarity we define a satisfaction relation on state properties and system states as follows:

DEFINITION 2 *(Satisfaction relation (models and state properties))*: Let $\mathcal{M}$ be a TA and let Φ be a state property. The binary satisfaction relation $\models$ is defined as follows:

$$\mathcal{M} \models \Phi \Leftrightarrow \text{each state } s \text{ of } \mathcal{TS}^{\mathcal{M}} \text{possesses property } \Phi$$

Hence the assertion of the validity of a safety property is straightforward: one simply needs to annotate the locations within the TA-based component model with a dedicated label, e. g. `violation` or track the quantity to be measured by a dedicated clock or variable. The atomic proposition is true once a state carries the respective label or it fulfills a queried constraint w. r. t. some clock or variable. As this work exploits the timed model checker Uppaal which uses a fractal of timed CTL the validity of a state property Φ w. r. t. some network of TA can be queried by the statement `A[]` Φ, which stands for 'always invariantly' Φ. It is briefly mentioned how the



(a) LTA for guarding α_j^{low}

(b) UTA for guarding α_i^{up}

System Declarations

```
broadcast channel event;
int BufferSz = 0;
```

Local Declarations for LTA, UTA

```
const int BMAX = |N^{up,low}_{i,j}|;
clock x;
int b = 0;
const int Delta = δ^{up,low}_{i,j}
```

(c) Declarations

(d) TA for enforcing full synchronization

Figure 2: TA-based implementation of RTC-based curves

verification of bounds on buffer sizes and delays works:

- *Maximum buffer size.* For asserting that a TA-based component model $\mathcal{M}$ solely requires a buffer space of a dedicated size one simply needs to assert the validity of an appropriate reachability query. E. g. in case of the timed model checker Uppaal one verifies that $\mathcal{M} \models$ [A[]b $\leq$ K]holds, which gives that for all states of component model $\mathcal{M}$ the (global) variable b is not larger than constant K.
- *Maximum delay.* Verifying the validity of a maximum on a delay is more evolved, it requires the usage of a so called observer TA. This TA which we adapted from [8] non-deterministically measures the delay from an event's generation to the event's output by a clock x considering also that several new events may arrive before an old event has left the component. By checking if $\mathcal{M} \models$ [A[](pause imply x $\leq$ D)] holds, with *pause* as dedicated location, x as clock valuation of clock x and D as clock constant, this assert than that any event traveling through $\mathcal{M}$ will experienced a delay of at most D time units. In the same manner it is straight-forward to assert the validity of a lower bound on event delays.

The above queries and if necessary a set of appropriate observer TA can used in a binary search for finding the maximum or minimum values for K and D. On termination of the search the property is asserted to hold for all states of a TA. Thus we speak in the following about invariants to be asserted, rather than properties to be verified.

2.4 Embedding of TA-based component models into RTC-driven performance analysis

In the following we briefly re-capitulate some aspects as presented in [10]. This is necessary as we exploit these patterns when computing the interface descriptions.

(A) Approximating RTC-conformant curves. This paper deals with complete amounts of events only, i. e. the number of events to be processed are integer values. Such RTC-conformant stair-case curves can be defined by sets of stair-case functions composed via nested minimum and maximum operations. However, for the sake of clearness the nesting of minimum and maximum operations is restricted, s. t. it is assumed that upper arrival curves are obtained as minimum and lower arrival curves as maximum on a set of stair-case curves:

$$\begin{aligned} \alpha_{up}^{\{in,out\}}(\Delta) &:= \min_I(\alpha_i(\Delta)) \text{ and} \\ \alpha_{low}^{\{in,out\}}(\Delta) &:= \max_J(0, \alpha_j(\Delta)) \text{ with} \\ \alpha_{\{i,j\}}(\Delta) &:= N_{\{i,j\}} + \left\lfloor \frac{\Delta}{\delta_{\{i,j\}}} \right\rfloor \end{aligned} \tag{2}$$

The fact that an upper RTC-curve is sub-additive and given that upper curves are assumed to be constructible by a single minimum operation (not nested min/max computations) yields that $\alpha_{up}^{\{in,out\}}$ has increasing step widths, i. e. the distances between two jump points grows with Δ. Analogously lower staircase curves $\alpha_{low}^{\{in,out\}}$ possess decreasing staircase sizes, i. e. the distances between two jump points shrinks for larger values of Δ. This feature will be denoted in the following as pseudo-convexness of upper and pseudo-concaveness of lower arrival curves. It reduces the complexity of embedding TA into RTC-driven performance analysis considerably.

For exemplification one may refer to Fig. 1: the upper arrival curve α^{up} is a pseudo-convex RTC curve, constituted by the minimum of two stair-case curves. The lower curve consist of the maximum of the constant 0-function and some staircase function α_1^{low}. The TA-based implementation of such upper and lower input curves will be discussed next.

(B) TA-based modeling of input curves. This is implemented by a set of cooperating TA, where the emitted event signals serve as input stimuli to a down-streamed TA-based component model. For covering pseudo-convex/concave input curves the event emitting input generator can be implemented as follows: (i) Each staircase curve of Eq. 2 (line 3) is implemented by its own event emitting TA, where in case of a upper curve α_i we speak of UTA i and in case of a lower curve α_j of LTA j. Their generic implementation within the timed model checker Uppaal is shown in Fig. 2 (a) - (c). (ii) **Full synchronization** of all UTA on event emission implements the above mentioned minimum computation on the set of upper event arrival curves (Eq. 2, line 1). (iii) Maximum building on lower staircase curves (cf. Eq. 2, line 2) is realized by enforcing event emission whenever one of the involved LTA has reached its local threshold N_j. Within Uppaal this behavior can be implemented by making use of a broadcast channel and the event scheduling TA depicted in Fig. **??** (d). A set of LTA and UTA cooperating via the event scheduler implements an input generator $\mathcal{G}$ which produces streams w. r. t. a dedicated event type, e. g. of type e. For a generator which implements arrival curve $\alpha := (\alpha^{up}, \alpha^{low})$ we use the notation $\mathcal{G}(\alpha)$ or if it is necessary the more precise notation $\mathcal{G}(\alpha^{up}, \alpha^{low})$. It is important to note that $\mathcal{G}(\alpha)$ allows to produce all streams the cumulative bounding functions of which are bounded in the sense of Eq. 1 (the proof is provided in [10]). As each event e can be used for stimulating a down-streamed, user-defined TA-based component model $\mathcal{M}$ we will speak of stimulated component models in the following. For the composite which is a crossproduct of the TA, we employ the notation $\mathcal{G}\|\mathcal{M}$.

For exemplification one may refer once again to Fig. 1 in combination with Fig. 2. For translating the upper curve α^{up} one need 2 UTA instantiated with $BMAX := N_{\{1,2\}}^{up}$ and $Delta := \delta_{\{1,2\}}^{up}$. For the event generator (Fig. 2 (d)) constant K is set to 2. For implementing the lower curve only a single LTA is needed. As it can not block event emission, it does not need to be considered when setting K to the required value, s. t. full synchronization is achieved.

(C) TA-based detection of output curves. Joint execution of a stimulated component model $\mathcal{G}\|\mathcal{M}$ and a set of dedicated observer TA for verifying some reachability query allows one to extract an upper and lower curve bounding all event streams emitable by $\mathcal{G}\|\mathcal{M}$. With observer we address a TA which guarantees the invulnerability of a fixed parameter value describing burst sizes, widths of stairs or minimum/maximum distance of events. Verifying an appropriate reachability query on a composite consisting of a stimulated component model and a set of observer TA ($\mathcal{G}\|\mathcal{M}\|\mathcal{O}$) allows one to extract parameters N_i and δ_i in a binary search. These parameters can than be employed for constructing a pseudo-convex, concave bounding output curve α_{up}^{out} and α_{low}^{out}.

3. RELATING INTERFACES AND COMPONENTS

3.1 Foundations

In this paper we only consider stateless assume/guarantee (A/G) interfaces, referred to as interfaces in the following. Such an interface defines the assumed input for which a component produces a guaranteed output. Hence an A/G interface restricts the environment to those inputs the respective component produces a guaranteed outputs for. Such a restricted environment is denoted as *"helpful"* environment. An interface (or a component) which has a non-empty, helpful environment is denoted as well-formed. In this work characteristics of inputs and outputs will be defined by RTC-based (event) arrival curves. However, contrary to the work of [12, 5] on RTC-based interfaces, it appears quite natural to extend interface definitions with other real-time constraints, invariants resp., such as bounds on buffer sizes and/or bounds on delays experienced by events processed by the resp. component. These invariants which need to be fulfilled by any component implementing the interface under consideration may refer to a invariant to hold for the overall system design.

For the sake of clearness we first concentrate on an 1-bounded scenario, which is the case of components (interfaces) possessing at most one input and one output port (bound).

DEFINITION 3 *((1-bounded) Component)*: A component $\mathcal{C}$ is a triple $(In, \mathcal{M}, Out)$, with

- In as dedicated input port for consuming event-triggers of a dedicated type.
- $\mathcal{M}$ is a component model which consumes and emits the event triggers.
- Out is a dedicated output port for emitting event-triggers of a dedicated type.

As standard time model checker allow one to use events of dedicated types the differentiation of component and TA-based component models seems to be somehow artificial. In the following, we therefore speak of TA-based component model $\mathcal{M}$, also sloppily denoted as component $\mathcal{M}$, instead of referring correctly to its encapsulating component $\mathcal{C}$.

We define an interface as follows

DEFINITION 4 *(Stateless (1-bounded) A/G Interfaces)*: An assume/guarantee interface $\mathcal{I}$ is a triple $(\Phi, \alpha_{in}^{\mathcal{I}}, \alpha_{out}^{\mathcal{I}})$, with

- Φ as set of dedicated component invariants, e. g. bounds on buffer sizes or bounds on the delays
- $\alpha_{in}^{\mathcal{I}}$ is a RTC curve bounding the cumulative counting function of any stream of event-triggers which we also denote as input bound.
- $\alpha_{out}^{\mathcal{I}}$ is a RTC curve bounding the cumulative counting function of any stream of event-triggers which we also denote as output bound.

One may note that in case of unused input/output bounds one need to employ the trivial bound α_ϵ for inputs and the constant 0-function for outputs. This is justified, as the interface makes no restriction w. r. t. the "helpful" environment and does not contain any guarantees w. r. t. the output.

A component $\mathcal{M}$ and an interface $\mathcal{I}$ are event-compatible, the bounds are mapped to their respective port, each port is assumed to refer to events of a certain type. As we are dealing with one-bounded interfaces and components this is straight-forward. It allows us to relate interface definitions and components as follows:

DEFINITION 5 *(Implementation relation)*: Let $\mathcal{G}$ be an input generator for restricting the input (streaming) behavior of the environment. A TA-based component $\mathcal{M}$ implements an interface $\mathcal{I} := (\Phi, \alpha_{in}^{\mathcal{I}}, \alpha_{out}^{\mathcal{I}})$ if the following conditions apply:
(a) $\mathcal{M}$ and $\mathcal{I}$ are event-compatibleand
(b) for the dedicated output event o and
$\forall tr \in Traces_{\mathsf{o}}^{\mathcal{G}(\alpha_{in}^{\mathcal{I}})\|\mathcal{M}}$:
tr is bounded by $\alpha_{out}^{\mathcal{I}}$ and
(c) $\forall\ s$ contained in $\mathcal{TS}^{\mathcal{G}(\alpha_{in}^{\mathcal{I}})\|\mathcal{M}} : s \models \Phi$
If $\mathcal{M}$ implements $\mathcal{I}$ we also use the notation $\mathcal{M} \lhd \mathcal{I}$ or speak of **conformance** of component $\mathcal{M}$ and interface $\mathcal{I}$.

In a nutshell, a component of an event-compatible interface/component pair guarantees that for all in-going event streams bounded by $\alpha_{in}^{\mathcal{I}}$, the component satisfies a dedicated state property Φ and it solely emits event streams bounded by $\alpha_{out}^{\mathcal{I}}$. This setting gives rise to the following two questions:

1. Does a given TA-based component $\mathcal{M}$ implement a (event-compatible) interface $\mathcal{I}$?
2. What is a component's $\mathcal{M}$ interface $\mathcal{I}$ w. r. t. a set of dedicated invariants Φ?

This question will be dealt with next (Sec. 3.2 and 3.3).

3.2 Conformance testing

Answering the question if a component $\mathcal{M}$ implements an interface $\mathcal{I} := (\Phi, \alpha_{in}^{\mathcal{I}}, \alpha_{out}^{\mathcal{I}})$ can be decided by verifying if $\mathcal{G}(\alpha_{in})\| \mathcal{M}\| \mathcal{O}(\alpha_{out}^{\mathcal{I}}) \models \Phi$ holds.

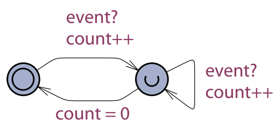
For keeping the complexity of the TA-based verification at a moderate level it is assumed that any interface-defined input or bound is conservatively approximated by a staircase function of pseudo-convex/concave shape (cf. Sec. 2.4).

3.2.1 Encoding the input bound

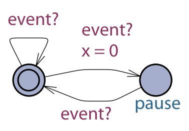
Analogously to the approach illustrated in Sec. 2.4 $\mathcal{G}(\alpha_{in})$ is implemented by a set of fully synchronizing UTA and LTA.

3.2.2 Encoding the output bound

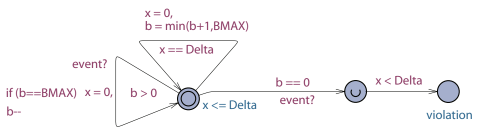
Instead of deciding if the stimulated component $\mathcal{G}(\alpha_{in})\|\mathcal{M}$ respects the interface-defined output bound $\alpha_{out}^{\mathcal{I}}$, we verify the invulnerability of a set of staircase curves, whose minimum/maximum



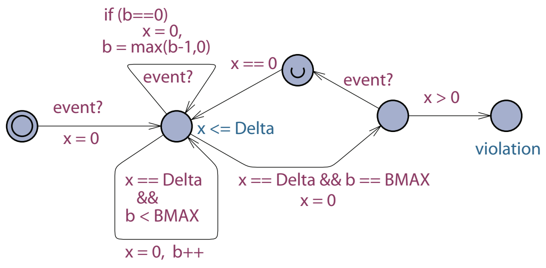
(a) Maximum burst



(b) Maximum distance



(c) Upper bound



(d) Lower bound

Figure 3: Observer TA for guarding interface-defined properties and output bounds

implement $\alpha^{\mathcal{I}}_{out}$. To do so the following TA and the respective queries can be employed:

(A) Guarding the i'th upper segment. The $i'th$ segment α^u_i of an upper output curve can be guarded by employing the output guarding TA presented in Fig. 3(c). This TA witnesses the violation or invulnerability of a staircase curve with parameters $N^u_i := BMAX$ and $\delta^u_i := Delta$. It does so by moving into the location `violation`, once the stimulated component produces too many events, i. e. once α^u_i is violated. Consequently testing the exclusive reachability of states which are not labeled with the atomic proposition `violation`, state by the query `A[] (not violation)`, asserts that α^u_i is a safe upper bound in the time-interval domain for all event streams emitable by the composite $\mathcal{G}(\alpha^{\mathcal{I}}_{in})\|\mathcal{M}\|\mathcal{O}(\alpha^u_i)$.

(B) Guarding the i'th lower segment. For guarding the $i'th$ staircase segment α^l_i one may employ the guarding TA depicted in Fig. 3(d). In analogy to any upper curve segment the TA, as well as the respective reachability query allows one to assert that a staircase curve α^l_i is a safe lower bound in the time-interval domain of any event stream producible by the composite $\mathcal{G}(\alpha^{\mathcal{I}}_{in})\|\mathcal{M}\|\mathcal{O}(\alpha^l_i)$.

Given the above guarding TA and a set of queries, a timed model checker can assert the invulnerability of the bound imposing staircase function, where the observer TA synchronize on event emission.

3.3 Computing an A/G interface for a TA-based component descriptions

Given a fixed component property to be fulfilled invariantly, the computation of interfaces allows one to either fix the input or the output bound.

3.3.1 Searching for the output bound

To do so one fixes the input bound to $\alpha^{\mathcal{I}}_{in}$ and computes the output bound α via a set of observer TA in a binary search and by verifying the TA $\mathcal{G}(\alpha^{\mathcal{I}}_{in})\|\mathcal{M}\|\mathcal{O}(\alpha)$. In a nutshell this is basically the procedure of [10] for interfacing RTC and TA-based component models.

3.3.2 Searching for the input bound

Analogously to the above procedure we guard the output bound $\alpha^{\mathcal{I}}_{out}$ by a set of guarding TA. What is left, is the deduction of an input bound $\alpha^{\mathcal{I}}_{in}$ via a binary search with differently instantiated input generators. The basic scheme for computing a conservative bound for a component's input streams, i. e. the detection of $\alpha^{\mathcal{I}}_{in}$, can be organized in two major parts.

(A) Bounding the long-term behavior of inputs. Algo. 1 computes the smallest value for δ in a binary search fashion which is the "steepest" long term rate s. t. $\mathcal{G}(\alpha^{up}_{in}, \alpha^{low}_{in})\|\mathcal{M}\|\mathcal{O}(\alpha^{\mathcal{I}}_{out}) \models \Phi$ holds. As input it requires some safe bounds $(\delta_{up}, \delta_{low})$, which are the smallest and largest values assigned to any δ_{crr} (cf. Sec. 3.3.3). s. t. $\mathcal{G}(\alpha^{low}_{in}, \alpha^{low}_{in})\|Model\|\mathcal{O}(\alpha^{\mathcal{I}}_{out}) \models \mathcal{I}$ holds. The statement of

Algorithm 1 Binary Search for RTC-interface Computation (upper bound)

```
Require: safe values for $\delta_{low}, \delta_{up}$.
 1: integer $\delta_{crr} := \delta_{low}, N := 0$
 2: while $|\delta_{up} - \delta_{low}| > \epsilon$ do
 3:     if Verifyta($\mathcal{G}[\texttt{BMAX:=N,Delta:=}\delta_{crr}]\|\mathcal{M}\|\mathcal{O}(\alpha^{\mathcal{I}}_{out}), \Phi$) = true
        then
 4:         $\delta_{low} := \delta_{crr}$
 5:         $\delta_{crr} :=$ makeNewValue($\delta_{low}, \delta_{up}$)
 6:     else
 7:         $\delta_{up} := \delta_{crr}$
 8:         $\delta_{crr} :=$ makeNewValue($\delta_{low}, \delta_{up}$)
 9:     end if
10: end while
```

line 2 executes the command line version of Uppaal, with a component $\mathcal{M}$ and the respective input generator $\mathcal{G}$. This generator solely consists of one UTA (cf. Fig. 2) and is instantiated with $BMAX := N$ and $Delta := \delta_{crr}$. Function `makeNewValue` as employed in line 5 and 8 computes a new value for δ_{crr}, e. g. the mean of δ_{low} and δ_{up}. At termination Algo. 1 delivers a d_{LT} which is the "steepest" long term rate. With δ_{crr} fixed to d_{LT} one is now enabled to find the largest value B_{LT} in a binary search s. t. for $\alpha^{up}_{in}(\Delta) := B_{LT} + \lfloor \frac{\Delta}{d_{LT}} \rfloor$ the stimulated and guarded model $\mathcal{M}$ satisfies Φ, e. g. $\mathcal{G}(\alpha^{up}_{in}, 0)\|\mathcal{M}\|\mathcal{O}(\alpha^{\mathcal{I}}_{out}) \models \Phi$ holds.
One may note that in case δ_{low} is not a valid value for d_{LT} s. t. $\mathcal{G}(\lfloor \frac{\Delta}{\delta_{low}} \rfloor, 0)\| \mathcal{M}\|\mathcal{O}(\alpha^{\mathcal{I}}_{out}) \models \Phi$ holds we above scheme does not work; we will come back to this in Sec. 3.3.3.

In a similar way it is straight-forward to derive now a lower curve α^{low}_{in}, s. t. $\mathcal{G}(\alpha^{up}_{in}, \alpha^{low}_{in}) \times \mathcal{M} \models \Phi$ holds.

At termination one obtains a conservative approximation for an input curve $\alpha_{in} := (\alpha^{up}_{in}, \alpha^{low}_{in})$ bounding *any* input stream s. t. the interface-defined invariant Φ and the output bound $\alpha^{\mathcal{I}}_{out}$ holds. As next we briefly indicate how to refine the basic scheme, s. t. not only long-term behaviors are approximated in an adequate manner, but also the short-term streaming behaviors are characterized less pessimistic.

(B) Bounding the short-term behavior of inputs. For conciseness the paper concentrates once again on the tightening the upper input bound α^{up}_{in}. The proposed scheme intends to model α^{up}_{in} as minimum of two staircase curves, i. e. it operates with a generator which consists of two UTA implementing arrival curves α_1 and α_2 ($\mathcal{G}[BMAX_1 := N_1, DELTA_1 := \delta_1, BMAX_2 := N_2, DELTA_2 := \delta_2]$). Consequently this setting gives a degree of freedom w. r. t. the (burst-)parameters N_1 and N_2 and w. r. t. δ_1;

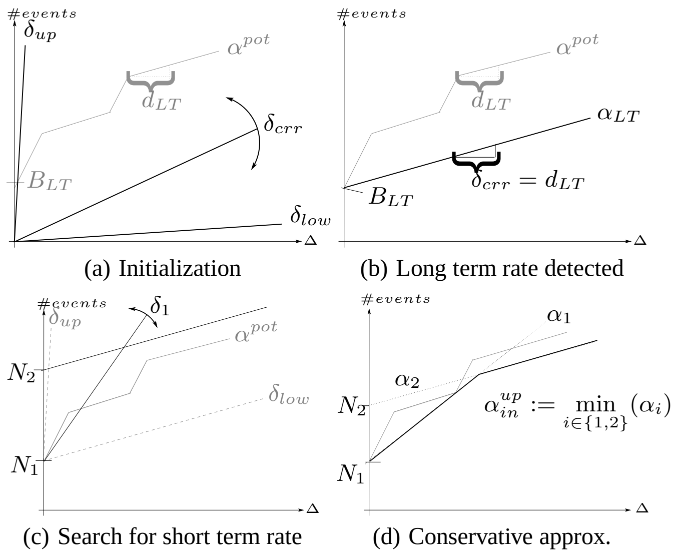


Figure 4: Computing an input bound

parameter δ_2 is already known, it is the long term rate d_{LT}. When choosing appropriate values for free parameters the following relation must hold $B_{LT} \leq N_1 \leq N_2$ and $0 < \delta_1 \leq d_{LT}$; this is because we model α_{in}^{up} as minimum of two staircase curves (cf. Eq. 2). According to this a possible initialization could than be $N_1 := B_{LT}$, $N_2 := k \cdot B_{LT}$ and with a δ_2 chosen as small as possible. It also appears reasonable to start with a binary search for finding an adequate value for δ_1, where for $\delta_1 = d_{LT}$ the search is restarted, but with smaller values for N_2 and N_1. At termination such a scheme would deliver a pseudo-concave staircase curve α_{in}^{up} bounding the inputs of model $\mathcal{M}$, where in a worst-case scenario this could once again yield curve $\alpha_{in}^{up}(\Delta) := B_{LT} + \lfloor\frac{\Delta}{d_{LT}}\rfloor$. It is interesting to note that for $\alpha_1 := N_1\lfloor\frac{\Delta}{\delta_1}\rfloor$ the respective UTA produces much more events on the long run if compared to the UTA implementing α_2. On the other hand this latter UTA produces much more events on the short run as the UTA implementing α_1. It is the full synchronization of both of them yielding that the overall input bound is implemented as the minimum of both curves.

(C) Example. Fig. 4 illustrates the basic functionality of the illustrated procedure, given that the input bound α_{in} is given by the unknown curve α^{pot} with its step widths bounded by d_{LT}. For illustration purpose we ignore the fact that in reality we are dealing with staircase functions. At first one searches now for the steepest long term rate which is depicted in Fig. 4(a), where δ_{crr} is the current slope to be tested in the binary search. At termination Algo. 1 delivers some value for δ_{crr} which allows us now to search for the largest value for N, which is input (burst-)parameter B_{LT} (Fig. 4(b)). With the long-term slope d_{LT} and the input (burst-)parameter B_{LT} one proceeds with the routine for finding a tighter bound, especially w. r. t. to short term behaviors, which is depicted in Fig. 4(b). As shown there the initial choice for N_2 was to large, hence the search for finding an adequate δ_1 stops unsuccessfully, namely once $\delta_1 = d_{LT}$. With a second run, executed with an adequate choice for N_2 the finding of δ_1 terminates as soon as $\mathcal{G}\|\mathcal{M}\|\mathcal{O} \models \Phi$ holds, where $\mathcal{G}$ is instantiated with the tuple $[BMAX_1 := N_1, DELTA_1 := \delta_1, BMAX_2 := N_2, DELTA_2 := \delta_2]$.

3.3.3 *Obstacles for the procedure*

There are certain conditions to be fulfilled by the component s. t. the presented scheme, which is based on a binary search works correctly. The obstacles which interfere with the scheme's applicability will be discussed now.

(A) Well formedness of components. It might appear that for a given component $\mathcal{M}$ there is no α_{in} s. t. $\mathcal{G}(\alpha_{in})\|\mathcal{M}\|\mathcal{O}(\alpha_{out}^{\mathcal{I}}) \models \Phi$ holds. For ruling this out, we adapt the concept of well-formedness of components as follows:

DEFINITION 6 *(Well-formed components)*: A component is denoted as well-formed iff there exists at least on RTC curve α s. t. $\mathcal{G}(\alpha)\|\mathcal{M}\|\mathcal{O}(\alpha_{out}^{\mathcal{I}}) \models \Phi$ holds.

(B) Monotonic regions. For the binary search to work it is required that Φ and $\alpha_{out}^{\mathcal{I}}$ holds for all traces produced by $\mathcal{G}(\alpha_{in}^{up}, \alpha_{in}^{low})$, where we denote this set of traces as monotonic region. Tow monotonic regions are disjoint, if they do not have any input stream in common.

(C) Non-uniqueness of input bounds. A component may have different incomparable input curves α' and α'' which both restrict the environment in such a way that Φ and $\alpha_{out}^{\mathcal{I}}$ hold. With their input generators $\mathcal{G}' := \mathcal{G}(\alpha')$ and $\mathcal{G}'' := \mathcal{G}(\alpha'')$ we have than $Traces^{\mathcal{G}'\|\mathcal{M}\|\mathcal{O}} \neq Traces^{\mathcal{G}''\|\mathcal{M}\|\mathcal{O}}$. This implies that the here proposed procedure for detecting a possible bounding input curve α_{in} for a given interface invariant Φ and a interface-defined output bound $\alpha_{out}^{\mathcal{I}}$ is a heuristic. As solution to this one may think to employ combinations of the obtainable input curves, similar to the pseudo-convex/concave approximation illustrated in Sec. 2.4. However, this is not necessarily possible, as the obtained curve allows behaviors which may violate the interface-defined invariant Φ or output bound $\alpha_{out}^{\mathcal{I}}$, or the obtained combination could be to pessimistic. We will come back to this issue, when we speak about the conformance of components and interfaces, defined by curve inclusion instead of employing the above mentioned conformance test (Sec. 3.2).

(D) Valid starting values for the binary search. The presented framework only works if the component $\mathcal{M}$ is well-formed. In case of disjoint monotonic regions it also delivers only one of them. Furthermore for dealing with non-uniqueness of input bounds different strategies may apply. In the illustrated scheme it is our aim to derive the input curve with the steepest long term rate, and the the highest burst, but other strategies may also be applicable. We also omitted a search for valid starting values for N and δ_{low} by simply setting N to 0 and δ to some value from $[\delta_{low}, \delta_{up}]$, where we also implicitly assumed that $\alpha'(\Delta) := \lfloor\frac{\Delta}{\delta_{low}}\rfloor$ is a valid upper bound and 0 a valid lower bound, i. e. $\mathcal{G}(\alpha', 0)\|\mathcal{M}\|\mathcal{O} \models \Phi$ must hold. However, other valid starting values for N and δ may apply which yields different results when bounding inputs: one may start a search with a generator instantiated with $BMAX := N$, $Delta := \infty$[1]. Once a valid value N_{safe} is known one executes a series of verification runs with $BMAX := N_{safe}$ and $Delta := delta$, which at termination delivers a safe starting value for δ. With $\alpha(\Delta) := N_{safe} + \lfloor\frac{\Delta}{\delta_{safe}}\rfloor$ this gives a safe curve to be refined in subsequently executed search schemes for obtaining tighter upper and lower bounds. But it is clear, that this once again only works for well-formed components.

[1] $Delta := \infty$ refers to the fact that this generator only releases burst of size N and remains inactive right after

3.4 Extending the framework for coping with multiple input/output ports

So far we only considered the case that an interface and the respective component possesses only a single input/output port. For coping with multiple ports, each dedicated to its own event type one simply needs to extend component and interface definition to the case of sets of input and output ports, set of input and output bounds resp.. As before components and interfaces are event-compatible if there is a mapping of each interface-defined bound to its own input port, where once again α_ϵ is assigned to unmatched input ports and the constant 0-function to all unmatched output ports.
In such a setting the scheme for asserting the conformance of a model $\mathcal{M}$ and its interface definition $\mathcal{I}$ needs not to be altered. One solely needs to employ sets of generators and guarding TA when testing $\mathcal{G}\|\mathcal{M}\|\mathcal{O} \models \Phi$, where $\mathcal{G}$ and $\mathcal{O}$ refers to sets of (networks) of TA. However in case of computing a component's interface form its TA-based implementation, i. e. the input/output bounds for the different ports the basic scheme needs to be adapted. As different event types may interfere with each other, when processed by the TA-based component model $\mathcal{M}$, one needs to fix the bounds of the other ports, when computing an input or output bound for a dedicated port which allows one to execute a binary search for finding the bounds w. r. t. the current event type.

4. INTERFACE-DRIVEN SYSTEM DESIGN

A major benefit of this work should be its support of compositionality w. r. t. system design. Hence the interconnection of two components is required to be safe w. r. t. some dedicated property if the components interfaces are conformant. Furthermore a system model is required to be invariant w. r. t. component's substitution, once it is known that the involved interfaces are conformant. Contrary to [6] we do not establish a formal notion of interface and component algebra here. This paper directly concentrates on the relevant, concrete aspects for establishing a methodology which allows incremental, i. e. component-wise evolution of system designs. In a nutshell, it is curve inclusion of interface-defined bounds which allows to establish such a feature for a system designs.

4.1 Foundations

At first it is shown that inclusion of input bounds yields invariance w. r. t. some system property Φ and some output bound. This basic feature is than exploited in the concourse of this section.

THEOREM 1: *Input inclusion implies invariance w. r. t. property* Φ

Let α and γ be some event-compatible input bounds, i. e. they refer to the same event type, let α' be a bound w. r. t. to some output event and let Φ be some (interface-defined) property. With $\mathcal{G}(\gamma)\|\mathcal{M}\|\mathcal{O}(\alpha') \models \Phi$ we have

$$\alpha \subseteq \gamma \Rightarrow \mathcal{G}(\alpha)\|\mathcal{M}\|\mathcal{O}(\alpha') \models \Phi \qquad \circ$$

The above theorem is correct if $Traces^{\mathcal{G}(\alpha)} \subseteq Traces^{\mathcal{G}(\gamma)}$ holds which will be shown by contradiction.

PROOF. Let α and γ be the arrival curves implemented by generators $\mathcal{G}_\alpha, \mathcal{G}_\gamma$. Generator $\mathcal{G}(\alpha)$ is capable of producing all streams bound by α and generator $\mathcal{G}(\gamma)$ is capable of producing all streams bound by γ [10]. Let $tr \in Traces^{\mathcal{G}_\alpha}$ and $tr \notin Traces^{\mathcal{G}_\gamma}$ with $(t, e) \in tr$ as the first timed event which separates the membership of tr from the set $Traces^{\mathcal{G}_\gamma}$. Consequently generator $\mathcal{G}_\gamma$ is not capable of producing an event at time t, but is generator $\mathcal{G}_\alpha$. With the number of produced events $R_{tr}(0, t)$ being bounded by α and α bounded by γ the cumulative event counting function $R_{tr}(0, t)$ is also being bound by γ. Thus a blocking of $\mathcal{G}_\gamma$ at time t, as it must have occurred, otherwise $\mathcal{G}_\gamma$ could emit an event, is not possible. Hence such an event (e, t) does not exists and therefore such a tr is not possible. Consequently $Traces^{\mathcal{G}(\alpha)} \subseteq Traces^{\mathcal{G}(\gamma)}$ holds for each curve α where $\alpha \subseteq \gamma$ holds. □

4.2 Consistent system designs and properties

With the scheme illustrated in the previous section it is possible to either guarantee conformance between an interface and its TA-based component implementation or to compute an interface for a component. Hence it is justified to assume that the following description can be established for a system design under consideration:

DEFINITION 7 *(Interface-based system description)*: A (RTC-conformant) interface-based system description Sys is a tuple $(\mathbb{I}, \mathbb{C})$, where $\mathbb{I}$ is a set of interface definitions and $\mathbb{C} \subseteq \mathbb{I} \times \mathbb{I}$ is a (directed) input/output (I/O) connection relation for the interface definitions.

For simplicity it is assumed that there are only 1:1 connections among the interfaces, the extension to the more general case of $n : m$ I/O relations is handled in Sec. 4.3.

DEFINITION 8 *(Safe I/0 interface-connectability)*: An interface $\mathbf{A}$ is safely I/O connectable with an interface $\mathbf{B}$ iff for their event-compatible arrival curves inclusion holds, i. e. $\alpha_{out}^{\mathbf{A}} \subseteq \alpha_{in}^{\mathbf{B}}$ holds. We also write $\mathbf{A} \subseteq_{IO} \mathbf{B}$.

A system description Sys is denoted as (interface) consistent iff for all pairs $(\mathbf{A}, \mathbf{B}) \in \mathbb{C}$ the relation $\mathbf{A} \subseteq_{IO} \mathbf{B}$ w. r. t. the respective event type holds. The above theorem implies the following features:

- two event-compatible interfaces $\mathbf{A}$ and $\mathbf{B}$ can be safely interconnected, if curve inclusion holds for the respective input/output bounds.
- Two event-compatible components A and B can be safely interconnected, if their conformant interfaces $\mathbf{A}$ and $\mathbf{B}$ can be safely interconnected.
- An interface $\mathbf{A}$ can be safely substituted by an interface $\mathbf{B}$ if for the interface-defined bounds curve inclusion holds.
- A component A can be safely substituted by an abstracted component B if the conformant interfaces $\mathbf{A}$ and $\mathbf{B}$ are compatible (not substitutable!).

With safely we refer to the fact that the above operations will not interfere with some system-wide property as established *from the interface definitions*. In the following the above features will be discussed in greater detail. Safely I/O connectable interfaces can be interconnected without interfering with the consistency of the overall system description. Such a composition yields a composite interface which we define as follows:

DEFINITION 9 *(Interconnection of interfaces)*: Let $\mathcal{I} := \mathbf{A}\|\mathbf{B}$ be the safe interconnection of the two safely I/O connectable interfaces $\mathbf{A}$ and $\mathbf{B}$. The composite $\mathcal{I}$ is than defined as follows:

$$\mathcal{I} := (\Phi := \Phi_a \wedge \Phi_b, \alpha_{in}^{\mathcal{I}} := \alpha_{in}^{\mathbf{A}}, \alpha_{out}^{\mathcal{I}} := \alpha_{out}^{\mathbf{B}})$$

One may note that $\mathbb{C}$ is a set of ordered pairs, hence $(\mathbf{A}, \mathbf{B}) \not\equiv (\mathbf{B}, \mathbf{A})$. According to the above theorem we can put an interfaces $\mathbf{A}$ (or component) in line with an interface $\mathbf{B}$ as long as the assumed input of $\mathbf{B}$ is included in the guaranteed output as provided by $\mathbf{B}$. This we can exploit now for safely putting components together.

COROLLARY 1: *Safe connectability of interfaces implies invariance w. r. t. composition of conformant components*

Let Sys be a consistent interface-based system description. For any pair of safely I/O connectable interfaces ($\mathbf{A}$, $\mathbf{B}$) and their conformant components $\mathcal{M}_a$ and $\mathcal{M}_b$, we have that curve inclusion of the interfaces implies that the composite $\mathcal{M}_a \| \mathcal{M}_b$ implements the composite interface $\mathcal{I} := \mathbf{A} \| \mathbf{B}$. Formally: for $\mathcal{M}_a \lhd \mathbf{A}$ and $\mathcal{M}_b \lhd \mathbf{B}$

$$\mathbf{A} \subseteq_{IO} \mathbf{B} \implies \mathcal{M}_a \| \mathcal{M}_b \lhd \mathbf{A} \| \mathbf{B} \qquad \circ$$

The above corollary follows from the above theorem and the fact that $\alpha_{out}^{\mathbf{A}} \subseteq \alpha_{in}^{\mathbf{B}}$ as $\mathbf{A}$ and $\mathbf{B}$ are defined to be safely I/O connectable.

As $\mathcal{G}(\alpha_{in}^{\mathbf{A}}) \| \mathcal{M}_a \| \mathcal{M}_b \| \mathcal{O}(\alpha_{out}^{\mathbf{B}}) \models \Phi_{\mathbf{A}} \wedge \Phi_{\mathbf{B}}$ must hold it is also clear that the interconnection of model $\mathcal{M}_a$ and $\mathcal{M}_b$ does not interfere with the properties of the overall system. As next we elaborate on component substitution.

DEFINITION 10 *(I/O interface compatibility)*: An interfaces $\mathbf{A}$ is I/O compatibility to an interface $\mathbf{B}$ iff they both state the same component invariant and for their event-compatible input and output curves inclusion holds, i. e.

$$\Phi_{\mathbf{A}} = \Phi_{\mathbf{B}} \text{ and } \alpha_{in}^{\mathbf{B}} \subseteq \alpha_{in}^{\mathbf{A}} \text{ and } \alpha_{out}^{\mathbf{A}} \subseteq \alpha_{out}^{\mathbf{B}}.$$

In case of strict I/O compatibility we use the notation $\mathbf{A} \sim_{IO} \mathbf{B}$

This definition allows us to establish invariance of a consistent system design w. r. t. substitution of interfaces as long as they are IO-compatible.

COROLLARY 2: *I/0 interface-compatibility implies invariance w. r. t. interface replacement*

Let Sys be a consistent interface-based system description. and let $\mathbf{A} \subseteq_{IO} \mathbf{B}$ hold. Each replacement of $\mathbf{B}$ by $\mathbf{A}$ will not change the consistency of Sys. $\circ$

Replacing an interface with a definition which is more tolerant w. r. t. its inputs but more restrictive w. r. t. its outputs does not change the behaviors of the interconnected interfaces, as curve inclusion is transitive and therefore the above definition applies. This together with corollary (1) gives here the invariance of interface-based system descriptions w. r. t. consistency. Finally we are now able to point out the most important feature w. r. t. compositional evolution of system designs:

COROLLARY 3: *Invariance of consistent interface-defined system descriptions w. r. t. component substitution*

Let Sys be a consistent interface-based system description. Component $\mathcal{M}$ can be substituted by any model $\mathcal{M}'$ without interfering with the invariants of the overall system if $\mathcal{I}' \sim_{IO} \mathcal{I}$ holds, given $\mathcal{M} \lhd \mathcal{I}$ and $\mathcal{M}' \lhd \mathcal{I}'$. $\circ$

The above corollary establishes a notion of equivalence of component implementations w. r. t. a property of the overall system design, where one may substituted components as long as the substitute is less restrictive w. r. t. its input assumption and more restrictive w. r. t. its outputs guarantees, if compared to the component to be substituted or the interface thereof. This is quite intuitive, as it states that the replacing component is prepared for accepting inputs bounded at least by $\alpha_{in}^{\mathcal{I}}$ and guarantees to emit streams at most bounded by $\alpha_{out}^{\mathcal{I}}$. This contravariance of inputs and outputs was already pointed out in [6],Here it pops out of curve inclusion, allowing us to establish a framework for the incremental and compositional evolution of system designs when using TA-based component implementations and RTC-based A/G interfaces.

COROLLARY 4: *Conformance of interfaces and components*

Let $\mathcal{M}$ be a component which implements an interface $\mathbf{A}$ ($\mathcal{M} \lhd \mathbf{A}$) and let $\mathcal{I}$ be some interface which is event-compatible to $\mathbf{A}$ For deciding if $\mathcal{M} \lhd \mathcal{I}$ holds, the following conditions are sufficient:

$$\Phi_{\mathbf{A}} \subseteq \Phi_{\mathcal{I}} \text{ and } \alpha_{in}^{\mathbf{A}} \subseteq \alpha_{in}^{\mathcal{I}} \text{ and } \alpha_{out}^{\mathbf{A}} \subseteq \alpha_{out}^{\mathcal{I}}. \qquad \circ$$

This can be verified for comparable curves on the basis of the RTC-based MPA toolbox [1], but doing so for pseudo-convex/concave input generators appears to be much more complex, especially if the computed input bounds are not comparable. This question might be the objective of future research. Nevertheless, the above corollary allows us to make an interesting observation: from corollary 3 we know that if $\mathbf{A} \sim_{IO} \mathbf{B}$, $\mathcal{M}_a \lhd \mathbf{A}$ and $\mathcal{M}_b \lhd \mathbf{B}$ holds, one may substitute $\mathcal{M}_a$ by $\mathcal{M}_b$. However, this does not imply that $\mathcal{M}_a \lhd \mathbf{B}$, as $\alpha_{out}^{\mathbf{A}} \subsetneq \alpha_{out}^{\mathbf{B}}$ may hold (cf. corollary 4). Hence the above setting allows us to substitute components with each other, even though they not necessarily implement the same interface.

4.3 Components with multiple I/O elements

For keeping the discussion as straight-forward as possible we only develop a set of criteria which allow to safely interconnect components with multiple ports of neighboring components bound to a single input port. At first this requires to extend component and interface definition to sets of input/output elements. Hence a component $\mathcal{C}$ is now a triple, where In is a set of input ports, each referring to a dedicated type of event, and where Out is a set of output ports, each emitting event-triggers of a dedicated type. Analogously we extend interfaces to set of input/output bounds, each referring to a specific event type. Consequently, a component $\mathcal{C}$ implements an interface $\mathcal{I}$ if Def. 5, corrollary 4 resp., applies for all output ports, resp. their event types and for all input bounds contained in $\mathcal{I}$; the input generators and guarding TA are executed in parallel. Overall this allows us to equip each port p with its interface-defined bound α_p, where α_ϵ is assigned to unmatched input ports and the constant 0-function to all unmatched output ports. This is once again justified, as the resp. interface makes no restriction w. r. t. the helpful environment and does not contain any guarantees w. r. t. the output. Let $\mathbb{C}$ be in this setting the interconnection-relation which puts input and output ports of components together, previously we only put interfaces together. This allows us to construct a set of output ports mapped to a dedicated input port p as follows:

$$In_p := \{q | (q, p) \in \mathbb{C}\}$$

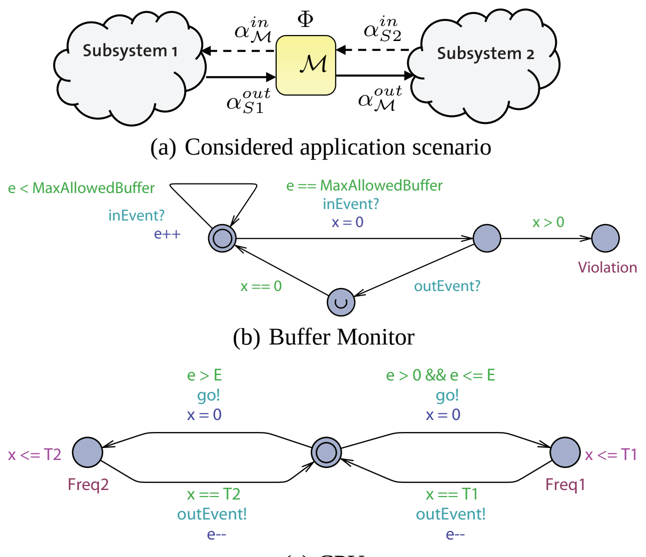
Given these sets we can exploit curve inclusion for safely interconnecting interfaces.

DEFINITION 11 *(I/0 interface-connectability in case of multiple inputs)*: Let $Out_{\mathbf{A}}$ be a set of output ports associated with interface $\mathbf{A}$, and let $In_{\mathbf{A}}$ be a set of input ports associated with interface $\mathbf{B}$. An interface $\mathbf{A}$ can safely be connected to interface $\mathbf{B}$ if the following conditions apply:

$$\sum_{q \in In_p} \alpha_q \subseteq \alpha_p$$

where α_r is the interface-defined input bound associated with port r.

A system is defined as consistent if all of its interfaces are I/0 interface-connectable. This allows one to establish safe interconnection of interfaces and I/O interface compatibility as done in the previous section, but now for interfaces with multiple inputs. As this allows also to check conformance of components and inter-



(a) Considered application scenario

(b) Buffer Monitor

(c) CPU

Figure 5: TA-based component model $\mathcal{M}$

faces, invariance of system description w. r. t. component composition and substitution can be shown.

5. CASE STUDY

In this section we show how the interface computation described above can be applied in practice. We perform four different experiments that are based on a simple application scenario. The first experiment illustrates the non-uniqueness of the interface computation. The second one compares our TA-based approach with a pure RTC-based method for deriving input interfaces. The third experiment demonstrates how the concept of interfaces simplifies the substitution or the refinement of components in a distributed system. The last experiment illustrates how the complexity of the analysis for large systems can be dramatically reduced by means of interface technologies.

(A) Application scenario. We consider the scenario depicted in Fig. 5(a) as basis for our experiments. The system consists of a state-based component $\mathcal{M}$ with an invariant Φ that is embedded in a larger system. Subsystems 1 and 2 can be arbitrary complex systems with multiple state-based or state-less components. For component $\mathcal{M}$ these subsystems are abstracted by means of a output bound α^{out}_{S1} guaranteed by the preceeding component and an assumed input bound α^{in}_{S2} by the succeeding component. For simplicity we solely considere upper input/output bounds, i. e. the lower bounds are simply set to the constant 0-function.

Component $\mathcal{M}$ models a CPU that executes a single task with an execution demand of 10^6 cycles. The CPU implements a load-dependent frequency adaptation. In particular, we assume that it operates at 166MHz if there are less than 4 events in its input buffer, and at 500MHz otherwise. Note that, for the sake of simplicity, we assume that the CPU frequency cannot be changed during the processing of an event. That is, the new CPU frequency is chosen only at the beginning of an event processing (depending on the current buffer fill level) and this frequency is kept constant until the next event processing starts. We assume that the input buffer of the CPU is limited to a maximum of 5 events. The invariant Φ of $\mathcal{M}$ asserts that the input buffer of the CPU does not overflow.

(B) Non-uniqueness of input bounds. In this first experiment we ignore Subsystems 1 and 2, and look only at component $\mathcal{M}$. The goal of the experiment is to derive an arrival curve $\alpha^{in}_{\mathcal{M}}$ that characterizes the maximum input that $\mathcal{M}$ can handle without violating invariant Φ. For finding $\alpha^{in}_{\mathcal{M}}$ we first model the behavior of $\mathcal{M}$ by means of the the two TA shown in Fig. 7. These automata use the broadcast signals inEvent and outEvent to distinguish between ingoing events coming from Subsystem 1 and outgoing events sent to Subsystem 2. The fill level of the input buffer is modelled by means of a counter variable e. The TA of Fig. 5(c) represents the load-dependent behavior of the CPU. The two locations *Freq1* and *Freq2* represent the processing of events at low and high frequency, resp., with corresponding processing times ETslow $= 6ms$ and ETfast $= 2ms$. With uregnt signal hurry we enforce greedy event processing. The TA of Fig. 5(b) is used to monitor the fill level of the input buffer, i.e., to detect violations of Φ. At this point we apply the heuristic of Sec. 3.3 to derive the largest tolerable input bound α^{in}_1 for $\mathcal{M}$. For the sake of simplicity, in this experiment we limit the search to a curve with one staircase segment and do not make any assumptions about the output bound to be guranteed by component $\mathcal{M}$ or assumed by Subsystem 2, i. e. $\alpha^{out}_{\mathcal{M}}$ and α^{in}_{S2} are the constant 0-function. The result of the search is the arrival curve $\alpha^{\mathcal{M}}_{1,a1}$ characterized by the two parameters $N = 1$, $\delta = 3$ and shown in Fig. 6(a). The interpretation of the curve is that if an input event arrives at most every $3ms$, then the buffer of the CPU will not overflow. This solution is, however, not unique. Some simple trial-and-error tests with different input curves reveal that also the curve $\alpha^{in}_{\mathcal{M},a2}$ shown in Fig. 6(a) is a valid maximum upper bound for the input such that Φ is guarded. $\alpha^{in}_{\mathcal{M},a2}$ tells us that the CPU can also tolerate a burst of 5 simultaneous input events followed by a stream of periodic input events with period $6ms$. The experiment nice illustrates that commonly there might be incomparable solutions for a component's input bound α^{in}. In this context it is intresting to note the following: let the output bound α^{out}_{S1} of Subsystem 1 be a periodic event stream with jitter, specified by the parameters $p^{out}_1 = 7ms$, $j^{out}_1 = 21ms$ (cf. Fig. 6(a)). Curve $\alpha^{in}_{\mathcal{M},a1}$ does not allow to decide, if an interconnection of $\mathcal{M}$ and the Subsystem 1 is safe. The reason is that $\alpha^{in}_{\mathcal{M},a1}$ and α^{out}_{S1} are not comparable and s. t. safeness of the interconnection is not guaranteed, As Def. 8 and Corollary 1 do not apply. On the other hand, $\alpha^{in}_{\mathcal{M},a2}$ shows that the connection is indeed safe, as $\alpha^{in}_{\mathcal{M},a2}\ supseteq\alpha^{out}_{S2}$. However, even in cases where one may only extract input bounds $\alpha^{in}_{\mathcal{M}}$ that are not comparable with the output guarantee α^{out} of a preceeding component, this is not problematic, as one can always fall back to the conformance test introduced in Sec. 3.2. It is importanyt to note that in case of a negative conformance check, the violation of input assumptions (and output gurantees), i. e. the violation of Def. 8 (10) may propagate in the system, enforcing to re-assert the consistency of an interface-defined system description and if necessary re-verifying the conformance of components and interfaces.

(C) Comparing RTC with TA-based interface computation. The goal of the second experiment is to compare the maximum tolerable input for $\mathcal{M}$ computed with the method described in this paper with the bound obtained by means of a pure RTC-based approach [12, 5]. In this experiment we consider not only the buffer constraint Φ, but also an output bound $\alpha^{out}_{\mathcal{M}} := \alpha^{in}_{S2}$ for $\mathcal{M}$ imposed by Subsystem 2. We assume that α^{in}_{S2} is a periodic stream with jitter, specified by the parameters $p^{in}_2 = 5ms$, $j^{in}_2 = 10ms$, $d^{in}_2 = 2ms$, which corresponds to the minimum of two staircase curves with parameters $N_a = 1$, $\delta_a = 2$, $N_b = 3$, $\delta_b = 5$. For guarding the invulnerability of $\alpha^{out}_{\mathcal{M}}$ we additionally employed two guarding TA (cf. Fig. 3(c)). We then run the heuristic of Sec. 3.3, where this time we choose to search for a solution consisting of two staircase segments in order to improve the accuracy of the input bound $\alpha^{in}_{\mathcal{M}}$ to be computed.

In the pure RTC-based analysis of $\mathcal{M}$, we cannot capture its load-dependent behavior. Therefore, we use a conservative approximation to model the component, namely that the CPU always runs at the slow frequency of 166MHz which we represent by means of an appropriate service curve $\beta_{\mathcal{M}}$. For the computation of $\alpha^{in}_{\mathcal{M}}$ we adopt the methodology presented in [5]. In particular, we use the expression

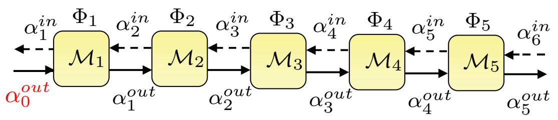
$$\alpha^{in}_{\mathcal{M}}(\Delta) = \min\{\inf_{0\leq\lambda\leq\Delta}\{\lfloor\beta_{\mathcal{M}}(\Delta-\lambda)\rfloor+\alpha^{in}_{S2}(\lambda)\},\ \beta_{\mathcal{M}}(\Delta)+b_{max}\}$$

to compute the maximum tolerable input for $\mathcal{M}$, where b_{max} denotes the maximum allowed fill level for the input buffer.

The results of the two analysis approaches are shown in Fig. 6(b). As can be seen in the figure, the RTC-based analysis computes a considerably more conservative result, i.e., allows only a smaller maximum input load for $\mathcal{M}$. This can be explained by the rough CPU model adopted in the RTC analysis which completely ignores the load-dependent behavior of the CPU.

(D) Substitution of components. In this experiment we demonstrate how the concept of interfaces simplifies the substitution of components in a distributed system. Assume that in the above scenario the interfaces of Subsystems 1 and 2 are defined by means of two periodic streams with jitter, specified by the following parameters: $p^{out}_1 = 5ms$, $j^{out}_1 = 5ms$, $d^{out}_1 = 3ms$, and $p^{in}_2 = 5ms$, $p^{in}_2 = 15ms$, $d^{in}_2 = 2ms$. Consider the component $\mathcal{M}$ with the above described load-dependent behavior and invariant Φ. By means of the same procedure as adopted in the second experiment we can derive the maximum tolerable input bound $\alpha^{in}_{\mathcal{M}}$ s. t. interconnection of $\mathcal{M}$ and Subsystem 2 is safe. The resulting curve is shown in Fig. 6(c) and demonstrates that $\mathcal{M}$ is safely interconnectable not only with Subsystems 1, but also with Subsystem 2, as $\alpha^{out}_{\mathcal{M}} \subseteq \alpha^{in}_{S2}$ holds. This latter fact was established by using the respective guarding TA for asserting the invulnerability of bound α^{in}_{S2} when computing $\alpha^{in}_{\mathcal{M}}$. The goal of this third experiment is to determine whether $\mathcal{M}$ can be replaced by a simpler component that fulfills the same invariant Φ and is compatible to Subsystems 1 and 2. In particular, we try to replace $\mathcal{M}$ by two different components $\mathcal{M}'$ and $\mathcal{M}''$. $\mathcal{M}'$ models a CPU running at a constant speed of 166Mhz, whereas represents a CPU running constantly at 200Mhz. By repeating the analysis procedure for these two components we obtain the maximum input bounds $\alpha^{in}_{\mathcal{M}'}$ and $\alpha^{in}_{1,\mathcal{M}''}$ shown in Fig. 6(c). As can be seen, $\alpha^{in}_{\mathcal{M}'}$ is not comparable to α^{out}_{S1}, hence we cannot guarantee that the system works if we use component $\mathcal{M}'$. In fact, according to the bound α^{out}_{S1}, Subsystem 1 can produce an input event for $\mathcal{M}'$ every $5ms$, whereas $\mathcal{M}'$ processes incoming events at a maximum rate of one every $6ms$. This means that on the long-term the input buffer of $\mathcal{M}'$ can overflow which corresponds to a violation of invariant Φ. On the other hand, we have that $\alpha^{in}_{\mathcal{M}''} \subseteq \alpha^{out}_{S1}$, which means that we can guarantee that the system still works correctly if we replace $\mathcal{M}$ by $\mathcal{M}''$. Note that we can safely substitute $\mathcal{M}$ by $\mathcal{M}''$ without the need of reiterating the analysis of Subsystem 2. This represents the major advantage of the described interface-based approach: We can locally verify the feasibility of changes in a complex distributed system and decide on the level of arrival curves if the changes made are safe w. r. t. the properties of overall system.

(E) Scalability. In this experiment we demonstrate how the computation of explicit interfaces for single components helps to considerably reduce the verification effort for large state-based systems. Consider a system consisting of 5 CPUs that process an event stream in a sequential manner. Fig. 7(a) shows an abstract representation of the system. Assume that each CPU in the chain implements a load-dependent frequency adaptation, according to the parameters summed up in Table 7(b). In particular, each CPU will execute at f_{low} if there are less then threshold events in its input buffer, and at f_{high} otherwise, where again we exclude frequency changes during the processing of an event. For each component $\mathcal{M}_i$ we introduce an invariant Φ_i, which says that CPU $|Model_i$ has at most 5 events in its input buffer. The goal of the analysis is to determine whether for an input bound α^{out}_0 (specified by the parameters $p^{out}_1 = 5ms$, $j^{out}_1 = 20ms$, $d^{out}_1 = 2ms$) each invariant Φ_i ($i \in \{1, \cdots, 5\}$) holds. We perform the analysis in two different ways in order to compare the computational effort. In the first analysis approach we ignore the modularity of the system an build a single TA model for the entire component chain. We couple this model with an event generator $\mathcal{G}(\alpha^{out}_0)$, i. e. it produces all streams bounded by α^{out}_0. With this genertor the executed all interconneted component models at once and check the individual invariants. In the second approach we analyse the system in a modular manner considering one CPU model $\mathcal{M}_i$ at a time. In particular, we go through the components starting from $\mathcal{M}_5$, and at each step compute the maximum tolerable input α^{in}_i of the corresponding component. In this way we can propagate the requirements of all the components to the system input, where they are subsumed by the input bound α^{in}_1. Note that at each step we have to consider only one single component, as the remaining part of the chain is abstracted by an appropriate interface. At the end we just need to compare α^{in}_1 with α^{out}_0 in order to verify whether a connection of this subsystem a up-streamed system emitting streams bounded by α^{out}_0 is safe. In both scenarios the analysis reveals that not all buffer constraints can be met for the given input α^{out}_0. In this experiment we want, however, to focus on the verification effort of the two different approaches. While the holistic analysis of the system requires a verification time of more than one hour, the modular analysis is carried out in a total time of less than one minute. Even if in the modular approach one has to consider a potential degradation of the analysis accuracy due to the conservative approximation of event streams, the experiment clearly shows the major advantage of interface-based modular analysis methods, in particular, when it comes to the use of (state-based) formal methods.



(a) Chain of CPUs

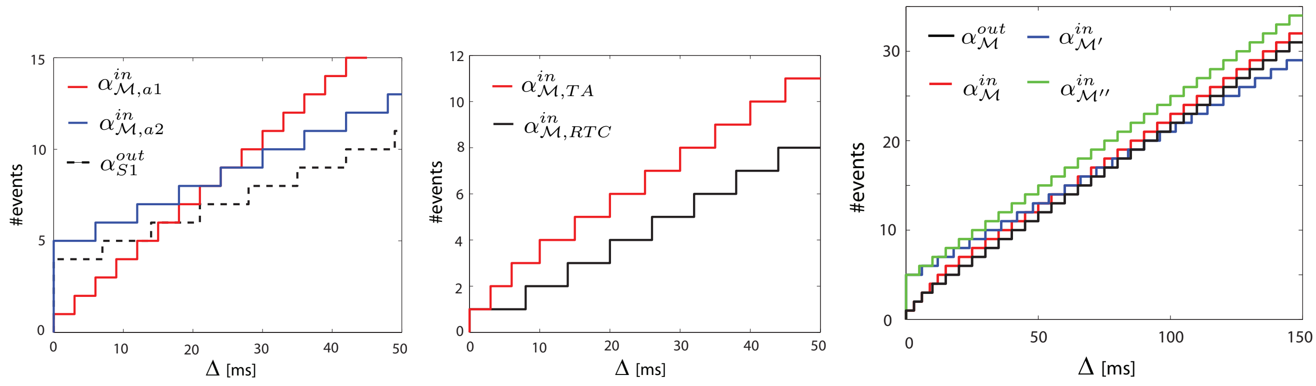
	CPU$_1$	CPU$_2$	CPU$_3$	CPU$_4$	CPU$_5$
f_{low} [MHz]	166	166	166	166	166
f_{high} [MHz]	1000	500	333	1000	500
threshold [events]	2	2	2	2	2

(b) Parameters for the CPU chain

Figure 7: CPU chain for investigting scalability

6. CONCLUSION

This paper develops a procedure for computing analytic input/output bounds for event streams consumed/emitted by TA-based component implementations of a system design, allowing to define stateless assume/guarantee (A/G) real-time interfaces for each component. However, the proposed procedure is not limited to TA, instead one could have used any other (state-based) real-time formalism, which allows to verify validity of invariants Φ w. r. t. a given RTC-based input and output bound. As we integrated these invari-



(a) Incomparable input bounds for $\mathcal{M}$ (b) TA and RTC-driven analysis (c) Compatibility checks

Figure 6: Input bounds computed for th different scenarios

ants directly into the interface definitions, we allow a computation of key performance metrics of the overall system design from the interfaces, rather than the component-based system model. This turns out to be a interesting feature, as it establishes incremental, i. e. component-wise evolution of system designs for the proposed framework. This is for the following reason: given a consistent interface-based system description the interface-derived properties are invariant w. r. t. composition and substitution of components, as long as each component is conformant to its interface and the interfaces are compatible. Overall this paper develops the required consistency, compatibility and conformance criteria and develops the machinery for carrying out the checks in an automatic fashion, where the applicable of th procedure could be shown by several case studies.

7. REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

[5] S. Chakraborty, Y. Liu, N. Stoimenov, L. Thiele, and E. Wandeler. Interface-based rate analysis of embedded systems. In *7th IEEE International Real-Time Systems Symposium (RTSS 06)*, pages 25–34, Rio de Janeiro, Brasil, 2006.

[6] L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In T. A. Henzinger and C. M. Kirsch, editors, *EMSOFT'01: Proc. of the 1'st International Workshop on Embedded Software*, volume 2211 of *LNCS*, pages 148–165. Springer, 2001.

[7] L. de Alfaro, T. A. Henzinger, and M. I. A. Stoelinga. Timed interfaces. In A. Sangiovanni-Vincentelli and J. Sifakis, editors, *EMSOFT 02: Proc. of 2nd Intl. Workshop on Embedded Software*, LNCS, pages 108–122. Springer, 2002.

[8] 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.

[9] 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.

[10] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: A hybrid method for analyzing embedded real-time systems. In *8th ACM & IEEE International conference on Embedded software, EMSOFT 2009*, pages 107–116, Grenoble, France, 2009. ACM.

[11] 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.

[12] E. Wandeler and L. Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 80–89, New York, NY, USA, 2005. ACM.

[13] 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.