

Component-based system design: analytic real-time interfaces for state-based component implementations

Kai Lampka · Simon Perathoner · Lothar Thiele

Published online: 9 August 2012
© Springer-Verlag 2012

Abstract Compositionality can be a helpful paradigm for coping with the complexity of large embedded systems with real-time constraints. This article exploits state-less assume/guarantee real-time interfaces extended by component properties, both to be satisfied invariantly by any component implementing the respective interface. This supports compositional evaluation of system designs, as overall properties of the latter can be derived from the component interfaces, rather than verifying them for the complete system model at analysis time. Moreover, our design strategy based on extended analytic real-time interfaces features an incremental, and component-wise development of (heterogeneous) system designs. This is, because the interface-derived properties of an overall system design are proven to be invariant under composition and substitution of interfaces and components, as long as the interface definitions are consistent, the (potentially heterogeneous) components implement their interfaces, and a dedicated inclusion criterion holds. This article develops the machinery for deriving analytic interface descriptions from Timed Automata-based component models. It develops the required consistency and conformance tests for deciding the aforementioned invariance of interfaces and components with respect to composition and substitution. This way we hope to advocate a strictly compositional design approach and improve the scalability of state-based analysis methods which we investigate in several case studies.

Keywords Real-time systems · System-level design technique · Interface theory · Real-time calculus · Timed automata

1 Introduction

1.1 Motivation

Verifying system behaviour by state-based modelling and model-checking techniques can be very beneficial. This is, because system behaviours can be precisely captured by the operational semantics of the modelling methods such as real-time UML state charts, Timed Petri-Nets, Timed Automata (TA), etc. However, due to the notorious state space explosion problem, i.e., the state-transition systems derived from such (high-level) models can be exponential in size with respect to their original specification, state-based system analysis can be extremely expensive in term of memory and computation time. Moreover, interaction of components of the commonly modular structured system models prohibits the analysis of the latter in a compositional way.

For limiting component's mutual interference and maintaining compositionality, we abstractly model interaction of components by streams of uniform, discrete activity-triggers. Such event streams can be characterized by so-called (event) arrival curves [15] or event models [10]. In this article, we exploit arrival curves for abstractly capturing the input/output behaviour of state-based component models, namely by interpreting them as so-called state-less, assume/guarantee (A/G), real-time interfaces. With such an interface we formally define (a) what kind of input traffic a component expects from its (up-streamed) environment, denoted as input assumption and (b) what kind of stream the component guarantees to emit to the (down-streamed) environment. Besides event curves, we also include the definition of properties

K. Lampka (✉)
Department of Information Technology, Uppsala University,
Uppsala, Sweden
e-mail: kilampka@gmail.com

S. Perathoner · L. Thiele
Computer Engineering and Communication Networks Lab,
ETH Zurich, Zurich, Switzerland

to be met invariantly by any component implementing the respective interfaces which allows one to compute dedicated key metrics of an overall system design from its interface description, rather than dealing with the system model all at once. With this strategy, we hope to support component-wise and incremental development of system designs. As long as the interface definitions are consistent, the (potentially heterogeneous) components implement their interfaces, and a dedicated inclusion criterion holds, the interface-derived properties of an overall system design are proven to be invariant under composition and substitution of interfaces and components.

This article establishes the required consistency and conformance criteria and develops the machinery for carrying out the checks in an automatic fashion. As level of component implementations we consider Timed Automata. However, it can be noted that other state-based methods which allow for the modelling of real-time constraints could also be employed. As we carry over properties established at the interface-level to the level of the component implementations we not only support a rigorous compositional design methodology and allow for a component-wise evolution of system designs, but also may improve the scalability of state-based system analysis.

1.2 Related work

de Alfaro and Henzinger [7] established the foundation of interface theory, but explicitly excluded timed behaviour from their elaboration. The theory on Timed Automata-based interfaces was provided in [8]. Weiss and Alur [17] exploit the theory of automata over infinite words to define interfaces between control designs and software implementations. This allows composition of interfaces by language-theoretic operations, where the authors exploit these features for real-time schedulability testing.

This article elaborates on analytic, i.e., state-less, assume/guarantee (A/G), real-time interfaces [5]. Like Chakraborty et al. [5], we exploit the streaming modelling of real-time calculus (RTC) [15] for bounding a component's input/output streaming behaviour. However, this article differs from previous work by offering the following innovations:

1. Unlike Chakraborty et al. [5], this article operates in a hybrid setting: it introduces the machinery for testing and deriving analytic A/G interfaces from state-based component models by exploiting ideas of [12]. Instead, one can also make use of the ideas presented in [1], which features the embedding of synchronous data-flow models into an RTC-driven performance analysis.
2. This work explicitly enriches interfaces with invariants to hold for the respective components. This allows to exclusively derive properties of the overall system from the interface-defined invariants.

3. Like Chakraborty et al. [5], this article establishes also the (contra-variant) "curve inclusion" as condition for efficiently deciding, whether the interface-derived properties of the overall system are guaranteed to hold under composition and refinement of the component models. But contrary to Chakraborty et al. [5], we accomplish this for TA-based component implementations. In turn this features heterogeneous modelling as curve inclusion is a sufficient condition now for RTC, as well as for TA-based component models.

With the above innovations, we implement a correctness-by-construction scheme for designing and implementing concurrent components communicating via ports. Thereby we feature heterogeneous, as well as incremental, component-wise evolution of such systems. This might help to cope with their increasing complexity.

1.3 Organization

Section 2 discusses the relevant background theories and presents relevant techniques established in previous works.

Section 3 presents a scheme for testing the conformance of interfaces and TA-based component implementations. Moreover, it introduces also a scheme for deriving interfaces from TA-based component implementations.

Section 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 components.

Section 5 presents an empirical evaluation of the framework. This is done (a) for demonstrating the applicability of the proposed algorithms, (b) for demonstrating that the interfacing of state-less and TA-based components allows for higher accuracy w.r.t. a system's performance metrics in comparison to a pure RTC-based interface methodology as established in [5,16], (c) for demonstrating the benefits of the usage of interfaces when evaluating alternate component implementations and (d) for exemplifying the scalability w.r.t. standard state-based analysis methodologies.

2 Background theory

2.1 Streams and their abstract representation

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

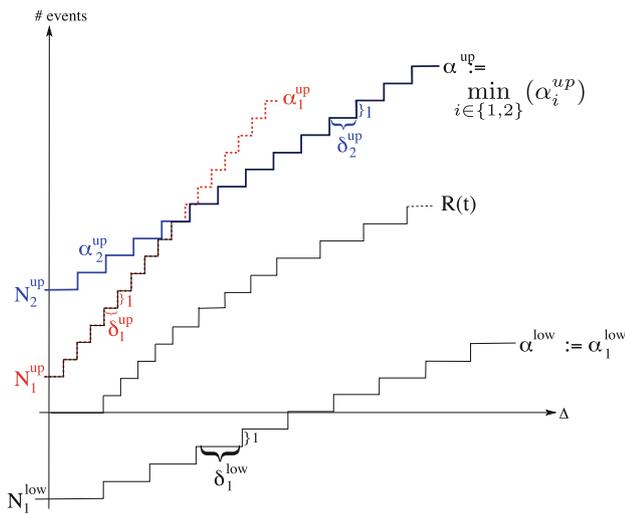


Fig. 1 Upper and lower (stair-case) arrival curves

trace $tr := (t_1, \tau); (t_2, \rho); \dots$ 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 $\tau, \rho \in Act$ 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 set of traces is referred to by $Traces$, and $Traces_e$ denotes the set of traces retrieved by applying a filtering operation w.r.t. event type e .

A stream commonly refers to a (possibly infinite set) of filtered traces. It can be abstractly characterized by the tuple $\alpha := (\alpha^{up}, \alpha^{low})$ which is a pair of an upper and a lower arrival curve [15]. An arrival curve α bounds a stream, the number of events seen on it, respectively, as follows:

$$\alpha^{low}(t - s) \leq R(t) - R(s) \leq \alpha^{up}(t - s) \tag{1}$$

for all $0 \leq s \leq t$ and where R is a cumulative event counting function. Overall, 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.

For exemplification one may refer to Fig. 1. Parameter N_1^{up} of the upper curve α^{up} models the (burst) capacity which is the number of events which can arrive at the same instant of time in any trace bounded from above by α^{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^{low} models the maximum distance between two events in the stream bounded from below by α^{low} .

In the following, we use the following concepts:

- 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.

- A stream, trace, port or an arrival curve always refer to a specific type of event, where we silently assume the presence of a correct mapping w.r.t. the event types. For making this explicit, we generically speak then of *event-compatible* ports, traces, streams or arrival curves.
- Two event-compatible arrival curves α_i^k and α_j^k are comparable iff $\alpha_i^k(\Delta) \leq \alpha_j^k(\Delta)$ or $\alpha_i^k(\Delta) \geq \alpha_j^k(\Delta)$ for $\Delta \in [0, \infty)$ and $k \in \{low, up\}$ holds. Otherwise the curves are denoted as incomparable.
- If for two arrival curves α_i and α_j it holds that $\alpha_i^{up}(\Delta) \leq \alpha_j^{up}(\Delta)$ and $\alpha_i^{low}(\Delta) \geq \alpha_j^{low}(\Delta)$ with $\Delta \in [0, \infty)$, α_i is said to be included in α_j and we write $\alpha_i \subseteq \alpha_j$.

2.2 Real-time analysis with timed automata

2.2.1 Timed automata

Timed automata [2] are cooperating state machines extended with clocks (and variables). Instead of states one commonly speaks of locations and instead of transitions one uses the term edges. The operational (or execution) semantics of TA can be characterized informally as follows: Enabled edges emanating from the currently active locations can be traversed which yields possibly a new set of active locations. While being in a location the values of clocks increase at the same speed, as (global) time progresses. 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 validity of an edge’s clock guards for deciding its traversability. In contrast, clock resets are executed upon the traversal of an edge. Variables are treated in a very similar manner, i.e., they can be inspected or manipulated, where manipulations take place upon the traversal of edges. 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.

For referring to this operational semantics of a TA, we will speak in the following of enabled edges and edge executions which refers to the fact that an edge’s guard evaluates to true and it is traversed.

As clocks only evaluate to values from finitely many different intervals, the originally infinite transition system of a TA can be represented by a finite quotient system, which we denote \mathcal{TS}^T . Hence validity of properties associated with states are decidable for TA [2]. Each path contained in \mathcal{TS}^T constitutes a set of traces, where the set of all traces defined by \mathcal{TS}^T is denoted by $Traces^T$. If one applies now a filtering operation w.r.t. event type e one ends up with a set of filtered traces, a stream, respectively, referred to by the notation $Traces_e^T$.

This paper employs the timed model checker Uppaal for computing real-time interfaces for TA-based component models. With Uppaal one may compose a system model from

a set of interacting TA which in the following, we sloppily denote as network of TA or simply TA. For clearness, we briefly re-capitulate some important rules of interaction, composition rules, respectively, as well as other relevant concepts; for further details see, e.g., [3,4].

Location invariants Constraints on variables or clocks can be assigned to locations, denominated as location or state invariants; the constraints assigned to the currently active locations have to hold as long as the TA resides in them.

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 behaviours. When jointly executing synchronizing edges the execution order of the related statements is non-deterministic. Hence it is important to solely employ commutative manipulations on synchronizing edges and global variables, e.g., increment of a global variable.

Cooperation via synchronization Uppaal makes use of channels, each identified by its own event label. With these channels different rendezvous, synchronization mechanisms, respectively, can be implemented. In the following, we highlight full synchronization of 1 sending TA and n receiving TA.

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 potential receiver of a broadcast event increments a global variable. The location invariant of the sender requires that after executing the sending edge the valuation of this variable equals some pre-defined constant, here n . Hence, either all of the respective edges are executed or none.

2.2.2 Verifying key metrics of systems

Determining if for a 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 in case of liveness properties with (execution) traces. In this paper we concentrate on the verification of state properties for the following reason: this work is concerned with interface properties which refer to key (performance) metrics of embedded (real-time) systems, e.g., buffer sizes or event latencies. These quantities can directly be derived from clock readings or values of variables and by checking the reachability of the respective states. Moreover, bounds on event latencies can be guarded by some (observer) TA which transits into a dedicated violation state once the measuring clock violates the requested time bound.

For convenience we define a satisfaction relation on state properties and system states:

Definition 1 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:

$$\mathcal{M} \models \Phi \Leftrightarrow \text{each states of } \mathcal{T}\mathcal{S}^{\mathcal{M}} \text{ possesses property } \Phi$$

Asserting the invalidity of (bad) state properties is straightforward: one simply annotates the locations of the TA-based models with dedicated labels, e.g., `violation` or the respective clock, or variable constraints and check $\mathcal{T}\mathcal{S}^{\mathcal{M}}$ for the absence of any respective state. With Uppaal this absence can be queried by the statement `A[] not violation`, which stands for ‘always invariantly’ Φ . On a technical level the verification of bounds on buffer sizes and event latencies, delays, respectively, works as follows:

Maximum buffer size In Uppaal $\mathcal{M} \models [A[]b \leq \kappa]$ queries if the buffer fill-level tracked by local variable b is (invariantly) smaller than some constant K .

Maximum delay Verifying the validity of a delay bound is more evolved as it requires the usage of a respective observer TA. This TA which we adapted from [9] non-deterministically measures the delay from an event’s generation to the event’s output.

By querying if

$$\mathcal{M} \models [A[](\text{pause imply } x \leq D)]$$

holds, with *pause* as dedicated location, x as clock and D as clock constant, this assert that that any event travelling 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.

In general, the finding of the maximum and minimum values for K and D can be achieved by executing a single exhaustive state space traversal, organized with the system model and a set of observer TA. However, the usage of a generic model checker like Uppaal requires the usage of the above queries validated for a system model and a set of observer TA in a binary search scheme for finding the maximum or minimum values of K and D . In any case, at termination the respective property is asserted to hold for all states of a system model. In the following, we therefore speak about invariants to be asserted, rather than properties to be verified.

2.3 Coupling TA and RTC-driven system analysis

We briefly present the pattern developed in [11] for coupling RTC and TA-based component descriptions.

2.3.1 Approximating RTC-conformant curves

This article deals with streams of discrete event triggers. The arrival curves of such streams can be modelled by sets of staircase functions composed via nested minimum and maximum operations. However, for the sake of simplicity it is assumed that upper arrival curves are obtained by a single minimum and lower arrival curves by a single maximum operation:

$$\alpha_{\{in,out\}}^{up}(\Delta) := \min_{i \in I} \left(N_{\{i,j\}} + \left\lfloor \frac{\Delta}{\delta_{\{i,j\}}} \right\rfloor \right) \text{ and}$$

$$\alpha_{\{in,out\}}^{low}(\Delta) := \max_{j \in J} \left(0, N_{\{i,j\}} + \left\lfloor \frac{\Delta}{\delta_{\{i,j\}}} \right\rfloor \right) \quad (2)$$

where I, J are the index set, curve parameters $N_{\{i,j\}}$ and $\delta_{\{i,j\}}$ are taken from the rational, with $\delta_{\{i,j\}}$ as divider of $N_{\{i,j\}}$, and Δ as the variable, here modelling the length of the time intervals.

The fact that an upper curve is sub-additive and given that upper curves are assumed to be constructible by a single minimum operation yields that $\alpha_{\{in,out\}}^{up}$ has increasing step widths δ_i , i.e. the distances between two jump points grows with variable Δ . Analogously, lower staircase curves $\alpha_{\{in,out\}}^{low}$ possess decreasing staircase sizes δ_j , i.e. the distances between two jump points shrinks for larger values of Δ . This feature will be denoted in the following as pseudo-concavity of upper and pseudo-convexity of lower arrival curves. It reduces the complexity of embedding TA into RTC-driven performance analysis considerably but may introduce some pessimism into the analysis. A detailed discussion on this subject can be found in [13].

Example Figure 1 depicts a pseudo-concave upper arrival curve α^{up} composed by the minimum of two staircase 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.

2.3.2 TA-based modelling of input curves

For implementing an event generator those timed traces w.r.t. a dedicate input event are bounded by a pseudo-concave/convex arrival curve α we use a network of interacting TA. The composite \mathcal{G} , i.e., the network of TA, as implemented in Uppaal works as follows:

1. Each staircase curve employed in the minimum and maximum operation (cf. Eq. 2) is implemented by its own event emitting TA. In case of an upper curve α_i , we speak of UTA i and in case of a lower curve α_j of LTA j . Their implementation in Uppaal is presented in Fig. 2a–c.
2. The event emitter, scheduler, respectively, is given in Fig. 2d. This TA *fully synchronizes* the UTA and LTA

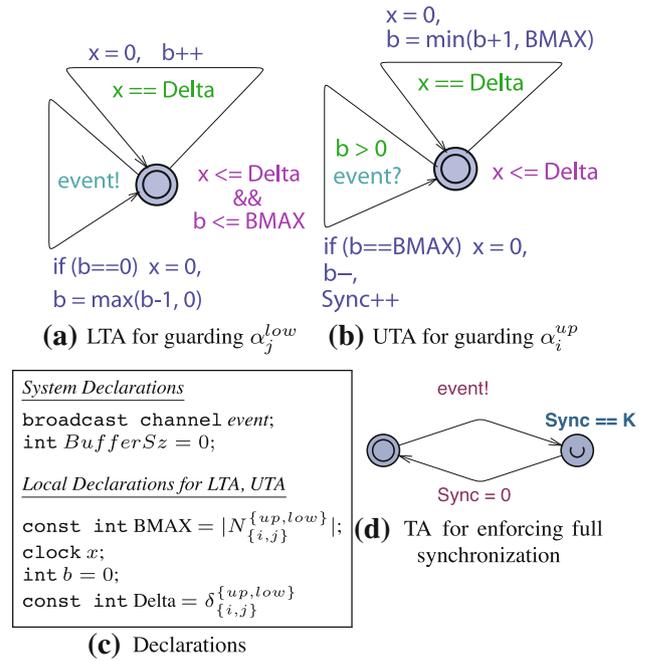


Fig. 2 TA-based implementation of RTC-based curves

on event emission. Event emission can only take place if and only if all UTA are able to execute the respective edge; the one guarded by the statement $b > 0$ of Fig. 2b. This mechanism implements the minimum computation of Eq. 2 (line 1).

3. 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 .
4. A set of LTA and UTA together with an event scheduler implement an input generator \mathcal{G} which is capable of producing all traces w.r.t. a dedicated event type, e.g., of type e and a bounding curve α . In the following, we use the notation $\mathcal{G}(\alpha)$, or $\mathcal{G}(\alpha^{up}, \alpha^{low})$ for emphasizing this. If it is clear from the context, we only use the symbol \mathcal{G} when referring to the input event generator. It is important to note that $\mathcal{G}(\alpha)$ allows to produce all traces the cumulative bounding functions of which are bounded in the sense of Eq. 1, the proof was provided in [11].

As each event e produced by \mathcal{G} may trigger subsequent behaviour in a down-streamed, user-defined TA-based component model \mathcal{M} we also speak of stimulated component models. For the composite, constituted by combining the TA \mathcal{G} and \mathcal{M} , we employ the notation $\mathcal{G} \parallel \mathcal{M}$. In this context, it is important to note that \mathcal{G} and \mathcal{M} are coupled via a shared variable which indicates the fill level of the input buffer of a component, rather than synchronizing the models via the joint execution of the event-generating edge. This is crucial as this

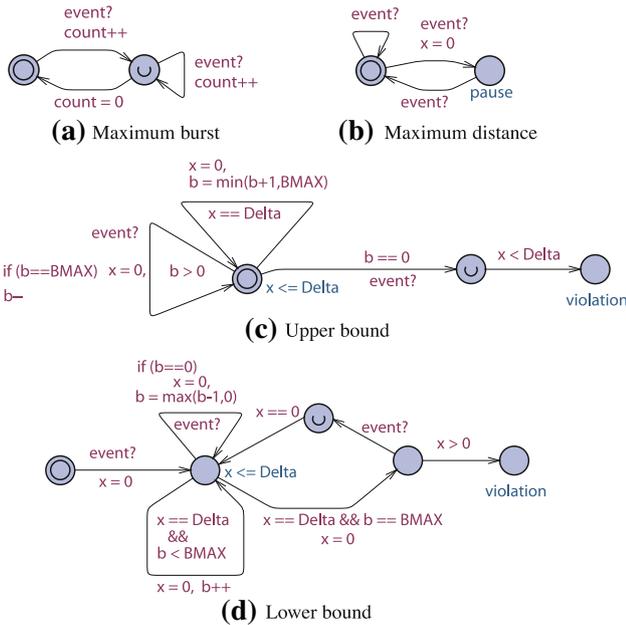


Fig. 3 TA for asserting invariances and output bounds

way one avoids the undesired blocking of event emission by the user-defined component model \mathcal{M} .

Example One may refer once again to Fig. 1 in combination with Fig. 2. For translating the upper curve α^{up} one needs 2 UTA instantiated with $BMAX := N_{\{1,2\}}^{up}$ and $\Delta := \delta_{\{1,2\}}^{up}$. For the event generator (Fig. 2d) constant K is set to 2. For implementing the lower curve only a single LTA is needed. As the LTA cannot block event emission, they do not need to be considered when enforcing full synchronization; as one may recall full synchronization implements the minimum computation of Eq. 2.

2.3.3 TA-based detection of output curves

Joint execution of a stimulated component model $\mathcal{G} \parallel \mathcal{M}$ and a set of dedicated observer TA \mathcal{O} allows to deduce a set of parameters in a binary search. These parameters can be used for constructing a pseudo-concave/convex output curve which bounds the stream emitted by $\mathcal{G}(\alpha) \parallel \mathcal{M}$ w.r.t. input curve α . We will come back to this in greater detail in the next section. For an example, one may already refer to Fig. 3.

3 Interfacing components

3.1 Foundations

This paper considers stateless assume/guarantee interfaces, referred to as interfaces in the following. Such interfaces define the assumed inputs for which components produce 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.

We limit ourself to components with a single input and output port. This is useful as in this article we employ a binary search for constructing the interface-related input/output bounds of a component. A brief discussion on how to handle settings with multi-ports follows in Sect. 3.3.3.

Definition 2 (1-bounded) Component.

A component \mathcal{C} is a triple (In, \mathcal{M}, Out) , with

- In is a dedicated input port for consuming event-triggers of a dedicated type.
- Out is a dedicated output port for emitting event-triggers of a dedicated type.
- \mathcal{M} is a component model, e.g., a single TA or a network of interacting TA. The component model consumes and emits the event triggers fetched, respectively, delivered, from the respective port.

In the same style we define an interface as follows

Definition 3 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 properties invariantly fulfilled by any component implementing the interface, where for simplicity we simply speak of an interface-defined invariant to be asserted.
- $\alpha_{in}^{\mathcal{I}}$ is an arrival curve which bounds any input trace accepted by a component in the sense of Eq. 1. In the following, this curve will be denoted as input bound.
- $\alpha_{out}^{\mathcal{I}}$ is an arrival curve which bounds any trace emitted by the component in the sense of Eq. 1. This curve is denote as output bound in the following.

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{C} and an interface \mathcal{I} are event-compatible, if the input/output bounds and input/output ports matches w.r.t. their event types, where we silently assume the presence of correct mappings only. Hence the concrete event type associated with a port is irrelevant. For this reason and because standard time model checker allow one to use events of dedicated types the differentiation between a port-based component \mathcal{C} and its TA-based model \mathcal{M} seems to be somehow

exaggerated. In the following, we therefore sloppily speak of the TA-based component model \mathcal{M} , rather than referring correctly to its encapsulating component \mathcal{C} . In total, we relate interface definitions and components now as follows:

Definition 4 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-compatible
- (b) for output event \circ and $tr \in \text{Traces}_{\circ}^{\mathcal{G}(\alpha_{in}^{\mathcal{I}})\|\mathcal{M}}$: tr is bounded by $\alpha_{out}^{\mathcal{I}}$ w.r.t. Eq. 1
- (c) for any s contained in $TS^{\mathcal{G}(\alpha_{in}^{\mathcal{I}})\|\mathcal{M}}$: $s \models \Phi$

If \mathcal{M} implements \mathcal{I} we also use the notation $\mathcal{M} \triangleleft \mathcal{I}$ or speak of *conformance* of component \mathcal{M} and interface \mathcal{I} .

The above setting gives rise to the following questions:

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

These questions will be addressed in Sects. 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 $\mathcal{G}(\alpha_{in}^{\mathcal{I}})\|\mathcal{M}\|\mathcal{O}(\alpha_{out}^{\mathcal{I}}) \models \Phi$ holds. The observer TA of $\mathcal{O}(\alpha_{out}^{\mathcal{I}})$ are used for guarding the invulnerability of output bound $\alpha_{out}^{\mathcal{I}}$ (cf. Fig. 3).

For keeping the complexity of the TA-based verification at a moderate level it is assumed that any interface-defined input or output bound is conservatively approximated by a staircase function of pseudo-concave/convex shape (cf. Sect. 2.3).

3.2.1 Encoding the input bound

Analogously to the approach illustrated in Sect. 2.3 $\mathcal{G}(\alpha_{in})$ is implemented by a set of 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 implement $\alpha_{out}^{\mathcal{I}}$. To do so the following TA and the respective queries can be employed:

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

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

Given sets of the above guarding TA and a set of queries a timed model checker can assert the invulnerability of an upper and lower pseudo-concave/convex staircase function.

3.3 Computing interfaces for TA-based models

Given an invariant to be met by a component, the binary search for computing a component's interfaces requires to either fix the input or the output bound.

3.3.1 Searching for the output bound

Here one fixes the input bound by setting it to $\alpha_{in}^{\mathcal{I}}$ and computes the output bound α via a set of observer TA employed in a binary search which verifies if $\mathcal{G}(\alpha_{in}^{\mathcal{I}})\|\mathcal{M}\|\mathcal{O}(\alpha) \models \Phi$ holds. This is basically the procedure of [11] for coupling RTC- and TA-based component models.

3.3.2 Searching for the input bound

Analogously to the procedure of Sect. 3.2.2 one employs a set of guarding TA for asserting the invulnerability of an interface-defined output bound $\alpha_{out}^{\mathcal{I}}$. What is left, is the deduction of an input bound $\alpha_{in}^{\mathcal{I}}$ via a binary search with differently instantiated input generators. The basic scheme for computing a conservative bound on a component's acceptable input stream, i.e., the detection of $\alpha_{in}^{\mathcal{I}}$, can be organized in two major parts.

Bounding the long-term behavior of inputs Algorithm of Fig. 4 computes the smallest value for δ in a binary search fashion which is the "steepest" long term rate s.t.

Require: safe values for $\delta_{low}, \delta_{up}$.

```

1: integer  $\delta_{crr} := \delta_{low}$ 
2: while  $|\delta_{up} - \delta_{low}| > \epsilon$  do
3:   if  $\text{VerifyTA}(\mathcal{G}[\text{BMAX}:=0, \text{Delta}:=\delta_{crr}] \parallel \mathcal{M} \parallel \mathcal{O}(\alpha_{out}^I), \Phi) = \text{true}$ 
     then
4:      $\delta_{low} := \delta_{crr}$ 
5:      $\delta_{crr} := \text{makeNewValue}(\delta_{low}, \delta_{up})$ 
6:   else
7:      $\delta_{up} := \delta_{crr}$ 
8:      $\delta_{crr} := \text{makeNewValue}(\delta_{low}, \delta_{up})$ 
9:   end if
10: end while

```

Fig. 4 Binary search for upper bound on RTC-interface

$\mathcal{G}(\alpha_{in}^{up}) \parallel \mathcal{M} \parallel \mathcal{O}(\alpha_{out}^I) \models \Phi$ 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. Sect. 3.3.3). In line 3 the algorithm executes the command line version of Uppaal, with a component \mathcal{M} and the respective input generator \mathcal{G} . This input generator solely consists of one UTA (cf. Fig. 2) and is instantiated with $\text{BMAX} := 0$ and $\text{Delta} := \delta_{crr}$. Function `makeNewValue` as employed in lines 5 and 8 computes a new value for δ_{crr} , e.g., the mean of δ_{low} and δ_{up} . At termination the search algorithm delivers a value 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 $\alpha_{in}^{up}(\Delta) := B_{LT} + \lfloor \frac{\Delta}{d_{LT}} \rfloor$ the stimulated and guarded model \mathcal{M} satisfies Φ , i.e., $\mathcal{G}(\alpha_{in}^{up}) \parallel \mathcal{M} \parallel \mathcal{O}(\alpha_{out}^I) \models \Phi$ holds.

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

At termination, one obtains a conservative approximation for an input curve $\alpha_{in} := (\alpha_{in}^{up}, \alpha_{in}^{low})$ bounding *any* input stream s.t. the interface-defined invariant Φ and the output bound α_{out}^I hold. As next we briefly indicate how to refine the basic scheme, s.t. the short-term behaviour of the input stream can be characterized less pessimistic.

Bounding the short-term behaviour of inputs For conciseness, the paper concentrates on the tightening of an upper input bound α_{in}^{up} . The proposed scheme intends to model α_{in}^{up} 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 ($\mathcal{G}[\text{BMAX}_1 := N_1, \text{DELTA}_1 := \delta_1, \text{BMAX}_2 := N_2, \text{DELTA}_2 := \delta_2]$). 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 ; parameter δ_2 is already known, it is the long term rate d_{LT} . When choosing appropriate values for the 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 δ_1 chosen as small as possible, where inadequate parameter settings can be

noticed by the case that the search routine reaches a stage, where $\delta_1 = d_{LT}$ holds. In such situations the search has to be restarted, but with a smaller value for N_2 . At termination, the above scheme delivers 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.

Example Figure 5 illustrates the basic functionality of the illustrated procedure in case of approx. an unknown input bound α^{pot} with two staircase segments; for illustration purpose Fig. 5 ignores that in our setting we are actually dealing with staircase functions. At first one searches now for the steepest long term rate which is depicted in Fig. 5a, where δ_{crr} is the current slope to be tested in the binary search. At termination, Fig. 4 delivers a value d_{LT} assigned to δ_{crr} . With δ_{crr} fixed we search now for the largest value for N , which gives us the input (burst-)parameter B_{LT} (Fig. 5b). With the long-term slope d_{LT} and the input (burst-)parameter B_{LT} one proceeds with the routine for finding a tighter bound on the short term streaming behaviour (Fig. 5c, d). As shown in Fig. 5c, the initial choice for N_2 was too large, hence the search for finding an adequate δ_1 stops unsuccessfully, namely once $\delta_1 = d_{LT}$. With a second run, executed with a smaller value for parameter N_2 the finding of δ_1 terminates as soon as $\mathcal{G}(\alpha_{in}^I) \parallel \mathcal{M} \parallel \mathcal{O}(\alpha_{out}^I) \models \Phi$ holds, where $\mathcal{G}(\alpha_{in}^I)$ is the set of input event generating TA instantiated with the tuple $[\text{BMAX}_1 := N_1, \text{DELTA}_1 := \delta_1, \text{BMAX}_2 := N_2, \text{DELTA}_2 := \delta_2]$.

3.3.3 Remarks

There are certain side conditions for the scheme to work properly. We briefly discuss them now.

Unknown input and output bounds If neither input nor output bound are fixed the binary search as proposed above is not feasible. This is because, the search always constructs a bound w.r.t. the fixed bound. However, in actual system designs the communication structure of the system is known. Moreover, one commonly knows at design time what the up-streamed environment guarantees to deliver or, alternatively, what the downstream environment assumes as input. These bounds can be used as starting values allowing one to compute the interface-related bounds along the communication paths

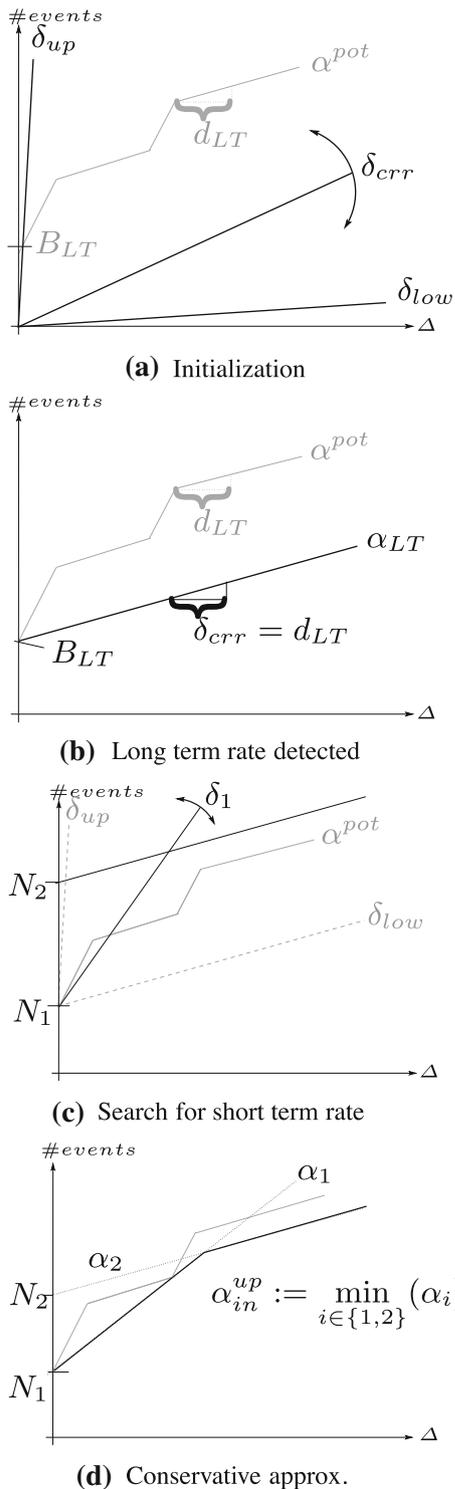


Fig. 5 Computing an input bound

and either forwardly or backwardly propagate the respective bounds.

As an alternative to this, one could also employ variants of the technique of [6, 14]. These approaches rely on Parametric TA and employ a timed model checker together

with a SMT solver to obtain regions for parameters of the system under study. In the interface setting considered in this paper, this would allow one to obtain regions for the parameters of the input and output bounding staircase functions employed for modeling the interface-related input and output bound.

Well-formedness of component models It might appear that for a given component \mathcal{M} there is no α s.t. $\mathcal{G}(\alpha) \parallel \mathcal{M} \parallel \mathcal{O}(\alpha_{out}^T) \models \Phi$ holds. For ruling this out we define that a component is well-formed iff there exists at least one RTC curve α s.t. $\mathcal{G}(\alpha) \parallel \mathcal{M} \parallel \mathcal{O}(\alpha_{out}^T) \models \Phi$ holds.

Feasible regions For the binary search to work, it is required that Φ and α_{out}^T hold for all traces produced by $\mathcal{G}(\gamma^{up}, \gamma^{low})$ or vice versa acceptable to $\mathcal{O}(\gamma^{up}, \gamma^{low})$. The set of traces bounded by γ^{up} and γ^{low} is denoted in the following as feasible set. Two feasible sets are disjoint, if they do not have any trace in common. In the following, we solely consider regular component models. These are models where the feasible input/output sets obtained by successively enlarge the respective bounds are non-disjoint.

Non-uniqueness of input bounds A well-formed component may have different incomparable input, output bounds α' and α'' respectively, s.t. Φ and the remaining other bound hold (see Sect. 5B for an example on the non-uniqueness of the input bound). As solution to this one may think of composing the individual bounds into a single bound, similar to the pseudo-concave/convex approximation illustrated in Sect. 2.3 (cf., Fig. 7a). However, this is not necessarily possible, as the obtained curve may allow behaviors which violate the interface-defined invariant Φ or the remaining bound. Moreover, such a composed bound could be overly pessimistic and thereby destroying invariance of the system properties when putting the overall system together, namely in case the constructed bound violates the invariance criteria developed in Sect. 4.

Initialization The presented framework only works properly if the component \mathcal{M} is well-formed and there are no disjoint feasible regions for the curves. Furthermore for dealing with non-uniqueness of input bounds different strategies may apply. The presented scheme constructs the input curve with the steepest long term rate, and a maximal burst size, but, other strategies may also be applied. Moreover, so far we silently assumed that $N := 0$ and $\delta \in [\delta_{low}, \delta_{up}]$ with δ_{low} and δ_{up} as valid bounds on the long-term rate. However, other starting values for N and δ may yield different results when constructing non-unique input bounds. As an alternative one could also start with a generator which implements curve $\alpha(\Delta) := N$. Once the maximum for parameter N is found, one would execute a search for detecting the respective rate δ . At termination of this search one would

obtain curve $\alpha(\Delta) := N_{\text{safe}} + \lfloor \frac{\Delta}{\delta_{\text{safe}}} \rfloor$. This curve could serve as starting point for a subsequent search which could target the tightening of this initial upper and lower bound.

Components with multiple ports In the basic setting we only consider components with a single input or output port. For coping with multiple ports, each dedicated to its own event type, one needs to extend the above component and interface definitions accordingly. As before components and interfaces are event-compatible if there is a mapping of each interface-defined bound to its own input port. Moreover, curve α_e 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 $\mathcal{G} \parallel \mathcal{M} \parallel \mathcal{O} \models \Phi$, where \mathcal{G} and \mathcal{O} refers to sets of (networks) of TA. However, when computing a component's interface from 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, the bounds obtained by executing a binary search scheme are not unique. For solving this problem, one could consider search techniques based on SMT-solvers [14] which are capable of delivering regions of bounds.

Handling of models where the above conditions are not met and solutions to the above obstacles are left to future work. For conciseness, we restricted the discussion to a very basic setting.

4 Interface-driven system design

A major benefit of this work should be its support of composition-wise evolution of system designs. Hence interface-derived properties of the overall system need to be invariant under interconnection and substitution of interfaces or components. This will be developed in the following. Contrary to [7], we do not establish a formal notion of interface and component algebra here. This paper concentrates on the relevant, concrete aspects for establishing the above mentioned invariance.

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 then exploited in the course of this section.

Theorem 1 *Input inclusion implies invariance w.r.t. a 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. some output event and let Φ be some (interface-defined) property. With $\mathcal{G}(\gamma) \parallel \mathcal{M} \parallel \mathcal{O}(\alpha') \models \Phi$ we have*

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

The above theorem is correct if $\text{Traces } \mathcal{G}(\alpha) \subseteq \text{Traces } \mathcal{G}(\gamma)$ 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}_α is capable of producing all traces bound by α and generator \mathcal{G}_γ is capable of producing all traces bound by γ (the proof can be found in [11]). Let $\text{tr} \in \text{Traces } \mathcal{G}_\alpha$ and $\text{tr} \notin \text{Traces } \mathcal{G}_\gamma$ with $(t, e) \in \text{tr}$ as the first timed event which separates the membership of tr from the set $\text{Traces } \mathcal{G}_\gamma$. Consequently generator \mathcal{G}_γ is not capable of producing an event at time t , but so is generator \mathcal{G}_α . With the number of produced events $R_{\text{tr}}(0, t)$ being bounded by α and α bounded by γ the cumulative event counting function $R_{\text{tr}}(0, t)$ must also bound by γ . Thus a blocking of \mathcal{G}_γ at time t , as it must have occurred, otherwise \mathcal{G}_γ could emit an event, is not possible. Hence, such an event (e, t) does not exist and therefore such a tr is not possible. Consequently $\text{Traces } \mathcal{G}(\alpha) \subseteq \text{Traces } \mathcal{G}(\gamma)$ holds for each curve α where $\alpha \subseteq \gamma$ holds. \square

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 5 Interface-based system description.

A (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, i.e., \mathbb{C} is a set of ordered pairs such that $(\mathbf{A}, \mathbf{B}) \neq (\mathbf{B}, \mathbf{A})$.

As before it is assumed that there are only 1:1 connections among the interfaces.

In the following, we use the attribute *safe*. With this we refer to the fact that the respective operation (composition or substitution) does not interfere with a system-wide property as derived from the interface-based system description.

Definition 6 Safe I/O interface-connectability.

The event-compatible interfaces \mathbf{A} and \mathbf{B} can safely be I/O connected iff for the event-compatible arrival curves inclusion holds, i.e., $\alpha_{\text{out}}^{\mathbf{A}} \subseteq \alpha_{\text{in}}^{\mathbf{B}}$, where we write $\mathbf{A} \subseteq_{\text{I/O}} \mathbf{B}$.

With the above definition we define system’s consistency as follows:

Definition 7 Consistent interface-based system descriptions. A system description Sys is denoted as (interface) consistent iff for all pairs $(\mathbf{A}, \mathbf{B}) \in \mathbb{C}$ the relation $\mathbf{A} \subseteq_{I/O} \mathbf{B}$ w.r.t. the respective event type holds.

The above theorem and the above definitions yield:

- Safe I/O connectable interfaces can safely put together, i.e., without interfering with the system’s properties.
- Two event-compatible components can be safely interconnected, if their conformant interfaces are safely I/O connectable.

This will be developed formally now.

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 8 Safe interconnection of interfaces.

Let $\mathcal{I} := \mathbf{A} \parallel \mathbf{B}$ be the safe interconnection of two safely I/O connectable interfaces \mathbf{A} and \mathbf{B} . The composite \mathcal{I} is defined as

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

According to the above theorem, we can put an interface \mathbf{A} in line with an interface \mathbf{B} as long as the assumed output bound of \mathbf{B} includes the guaranteed output as provided by \mathbf{A} . 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 $\mathcal{M}_a \parallel \mathcal{M}_b$ implements the composite interface $\mathcal{I} := \mathbf{A} \parallel \mathbf{B}$, i.e.,

$$\mathcal{M}_a \triangleleft \mathbf{A} \wedge \mathcal{M}_b \triangleleft \mathbf{B} \wedge \mathbf{A} \subseteq_{I/O} \mathbf{B} \implies \mathcal{M}_a \parallel \mathcal{M}_b \triangleleft \mathbf{A} \parallel \mathbf{B}$$

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 $\mathcal{G}(\alpha_{in}^{\mathbf{A}}) \parallel \mathcal{M}_a \parallel \mathcal{M}_b \parallel \mathcal{O}(\alpha_{out}^{\mathbf{B}}) \models \Phi_{\mathbf{A}} \wedge \Phi_{\mathbf{B}}$ 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 the safeness of interface and component substitutions.

Definition 9 I/O interface compatibility. Two event-compatible interfaces \mathbf{A} and \mathbf{B} are I/O compatible iff the following condition applies:

$$\Phi_{\mathbf{A}} \supseteq \Phi_{\mathbf{B}} \wedge \alpha_{in}^{\mathbf{A}} \subseteq \alpha_{in}^{\mathbf{B}} \wedge \alpha_{out}^{\mathbf{B}} \subseteq \alpha_{out}^{\mathbf{A}}.$$

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

In the above definition, $\Phi_{\mathbf{A}} \supseteq \Phi_{\mathbf{B}}$ means that \mathbf{B} is more strict w.r.t. the interface defined component properties, e.g., it uses less buffer space or features reduced end-to-end delays of events travelling through the component.

The above setting yields an ordered relation as $\mathbf{B} \sim_{I/O} \mathbf{A}$ does not imply $\mathbf{A} \sim_{I/O} \mathbf{B}$. Nevertheless, the above setting yields the invariance of a consistent system design w.r.t. substitution of interfaces, where interface \mathbf{B} may replace interface \mathbf{A} but not vice versa.

Corollary 2 I/O interface-compatibility implies invariance w.r.t. interface substitution. Let Sys be a consistent interface-based system description and let $\mathbf{B} \sim_{I/O} \mathbf{A}$ hold. Each substitution of \mathbf{A} by \mathbf{B} does not change the consistency of Sys .

Replacing an interface with a definition which is more tolerant w.r.t. its input but more restrictive w.r.t. its output does not change the behaviours of 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. Overall this yields that one may substitute an interface as long as the substitute is less restrictive w.r.t. its input assumption and more restrictive w.r.t. its outputs guarantees. This is quite intuitive, since it states that the substituting interface is prepared for accepting traces bounded at least by $\alpha_{in}^{\mathbf{A}}$ and guarantees to emit traces at most bounded by $\alpha_{out}^{\mathbf{B}}$. This contra-variance of input and output was already pointed out in [7], however here it pops out from curve inclusion. With substitution of interfaces we are now in the position to develop 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}_a can be substituted by any model \mathcal{M}_b without interfering with the invariants of the overall system if $\mathbf{B} \sim_{I/O} \mathbf{A}$ holds, given $\mathcal{M}_a \triangleleft \mathbf{A}$ and $\mathcal{M}_b \triangleleft \mathbf{B}$.

The above Corollary establishes a notion of equivalence of component implementations w.r.t. interface-defined properties of the overall system design. Analogously to interfaces one may substitute components as long as the substitute is less restrictive w.r.t. its input and more restrictive w.r.t. its output guarantee. In the last step we like to point out, that the substituting component, not necessarily implements the same interface.

Corollary 4 Conformance of interfaces and components. Let $\mathcal{M} \triangleleft \mathbf{A}$ and let \mathbf{B} be some interface which is safe I/O-compatible to \mathbf{A} . For deciding if $\mathcal{M} \triangleleft \mathbf{B}$, the following conditions are sufficient:

$$\Phi_{\mathbf{B}} \subseteq \Phi_{\mathbf{A}} \wedge \alpha_{in}^{\mathbf{A}} \subseteq \alpha_{in}^{\mathbf{B}} \wedge \alpha_{out}^{\mathbf{B}} \subseteq \alpha_{out}^{\mathbf{A}}.$$

$\Phi_{\mathbf{B}} \subseteq \Phi_{\mathbf{A}}$ is interpreted as before.

The above conditions can be verified for comparable and event-compatible components and interfaces on the basis of the algorithm presented earlier, as it deliver event bounds for components; arrival curves can be compared within the RTC methodology.¹

Corollaries 3 and 4 allow us to make an interesting observation: if $\mathbf{B} \sim_{I/O} \mathbf{A}$, $\mathcal{M}_a \triangleleft \mathbf{A}$ and $\mathcal{M}_b \triangleleft \mathbf{B}$ holds, one may substitute component implementation \mathcal{M}_a by component implementation \mathcal{M}_b . This substitution will not lead to a violation of a system’s interface-derived overall properties, even though component \mathcal{M}_b may not implement interface \mathbf{A} , e.g., $\alpha_{out}^{\mathbf{B}} \subset \alpha_{out}^{\mathbf{A}}$ may hold. Hence the developed framework allows one to substitute a component, where the substitute may not implement the original interface.

4.3 Components with multiple I/O elements

At first this requires to extend component and interface definitions to sets of input/output elements. Hence an interface is now a triple, where the individual input/output elements are set of bounds. Analogously, we extend components to set of ports, each referring to a specific event type. Consequently, a component implements an interface if Definition 5, Corollary 4, respectively, applies for all output ports, respectively, their event types and for all input bounds contained in the interface. Overall this allows us to assign a bound to each port p , where α_e is assigned to unmatched input ports and the constant 0-function to all unmatched output ports. Relation \mathbb{C} enable us to construct a set \mathcal{B}_i of input bounds for each input port. This way one may establish safe interconnection and safe I/O compatibility of components, namely by asserting that the sum of the elements of \mathcal{B}_i is bounded by the interface-defined bound assigned to the respective port. As before such a criterion allows to establish the properties as developed in the 1:1 bounded case.

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 can

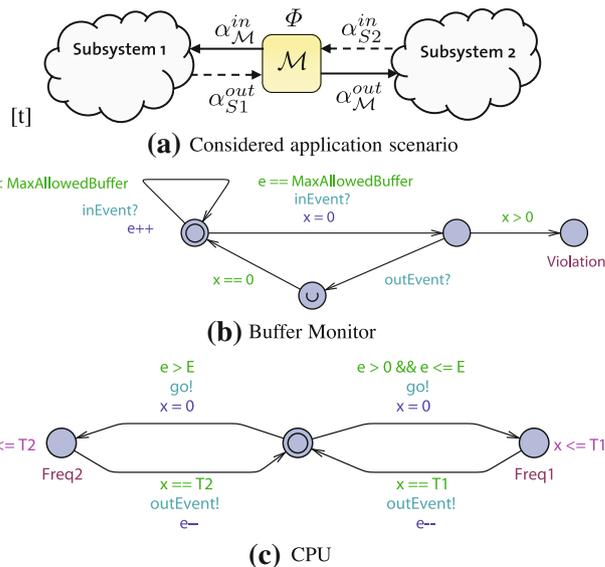


Fig. 6 TA-based component model \mathcal{M}

be dramatically reduced for large systems by means of interface technologies.

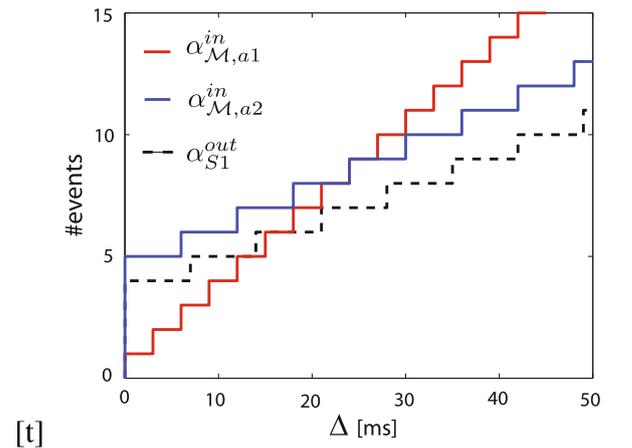
Application scenario We consider the scenario depicted in Fig. 6a as basis for our experiments. The system consists of a state-based component \mathcal{M} with an invariant Φ that is embedded in a larger system. Subsystems 1 and 2 can be arbitrary complex systems with multiple state-based or stateless components. For component \mathcal{M} these subsystems are abstracted by means of an output bound α_{S1}^{out} guaranteed by the up-streamed component and an assumed input bound α_{S2}^{in} by the down-streamed component. For simplicity, we solely consider upper input/output bounds, i.e., the lower bounds are set to the constant 0-function.

Component \mathcal{M} models a CPU that executes a single event-triggered 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 166 MHz if there are less than four events in its input buffer, and at 500 MHz 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 a task execution (depending on the current buffer fill level) and this frequency is kept constant until the next task execution 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 must not overflow.

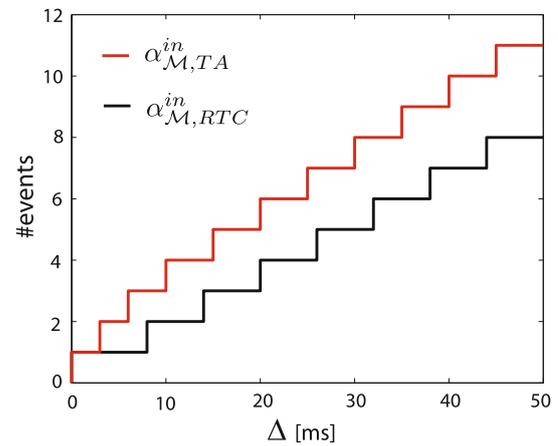
Non-uniqueness of input bounds In this first experiment we ignore Subsystems 1 and 2, and look only at component \mathcal{M} . The goal of the experiment is to derive an arrival curve

¹ A direct comparison of pseudo-concave/convex input generators and staircase curves remains unclear, where this question might be the objective of future research.

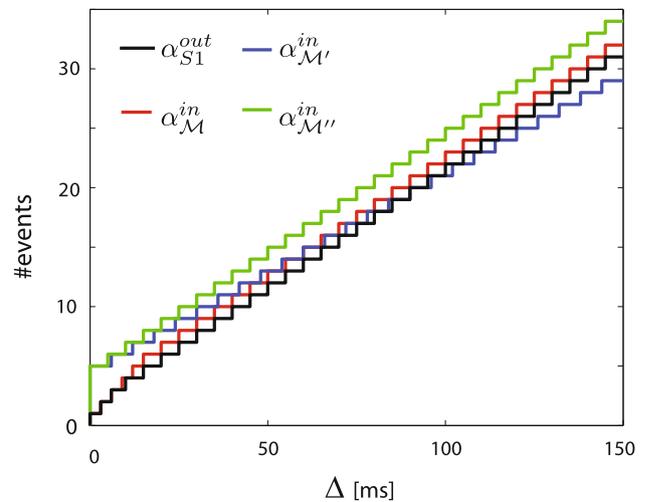
$\alpha_{\mathcal{M}}^{\text{in}}$ that characterizes the maximum input that \mathcal{M} can handle without violating invariant Φ . For finding $\alpha_{\mathcal{M}}^{\text{in}}$ we first model the behavior of \mathcal{M} by means of the two TA shown in Fig. 8. These automata use the broadcast signals *inEvent* and *outEvent* to distinguish between in-going 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. 6c represents the load-dependent behavior of the CPU. The two locations *Freq1* and *Freq2* represent the processing of events at low and high frequency, respectively, with corresponding processing times $T1 = 6$ ms and $T2 = 2$ ms. With the urgent signal *go* we enforce greedy event processing. The TA of Fig. 6b is used to monitor the fill level of the input buffer, i.e., to detect violations of Φ . We assume that in the case of a simultaneous occurrence of an input and an output event the input buffer of the CPU does not increase and consider this in the TA of Fig. 6b. At this point, we apply the heuristic of Sect. 3.3 to derive the maximum input bound α_1^{in} 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 guaranteed by component \mathcal{M} , i.e., we set $\alpha_{S2}^{\text{in}} := \alpha_\epsilon$. The result of the search is the arrival curve $\alpha_{1,a1}^{\mathcal{M}}$ characterized by the two parameters $N = 1, \delta = 3$ and shown in Fig. 7a. The interpretation of $\alpha_{1,a1}^{\mathcal{M}}$ is that if an input event arrives at most every 3 ms, 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 $\alpha_{\mathcal{M},a2}^{\text{in}}$ shown in Fig. 7a is a valid maximum upper bound for the input s.t. Φ is guarded. $\alpha_{\mathcal{M},a2}^{\text{in}}$ tells us that the CPU can also tolerate a burst of 5 simultaneous input events followed by periodic inputs with period 6 ms. The experiment nicely illustrates that commonly there might be incomparable solutions for the input of a component. In this context it is interesting to note that $\alpha_{\mathcal{M},a1}^{\text{in}}$ and $\alpha_{\mathcal{M},a2}^{\text{in}}$ might lead to different design decisions. For instance, let the output bound α_{S1}^{out} of Subsystem 1 be a periodic event stream with jitter, specified by the parameters $p_1^{\text{out}} = 7$ ms, $j_1^{\text{out}} = 21$ ms (cf. Fig. 7a). Then, curve $\alpha_{\mathcal{M},a1}^{\text{in}}$ does not allow to decide, whether connecting \mathcal{M} and Subsystem 1 is safe. The reason is that $\alpha_{\mathcal{M},a1}^{\text{in}}$ and α_{S1}^{out} are not comparable and hence Definition 8 and Corollary 1 do not apply. On the other hand, $\alpha_{\mathcal{M},a2}^{\text{in}}$ shows that the connection is indeed safe, as $\alpha_{\mathcal{M},a2}^{\text{in}} \supseteq \alpha_{S1}^{\text{out}}$. However, even in cases where one may only extract input bounds $\alpha_{\mathcal{M}}^{\text{in}}$ that are not comparable with the output guarantee α^{out} of the up-streamed component, this is not problematic, as one can always fall back to the conformance test introduced in Sect. 3.2. It is important to note that in case of a negative conformance check, the violation of input assumptions (and output guarantees), i.e., the violation of Definition 8 and 9, may propagate in the system. This enforces to re-assert the



(a) Incomparable input bounds for \mathcal{M}



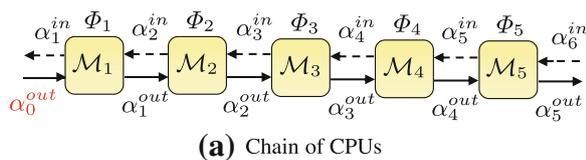
(b) TA and RTC-driven analysis



(c) Compatibility checks

Fig. 7 Input bounds computed for th different scenarios

consistency of the interface-defined system description and, if necessary, to re-verify the conformance of components and interfaces.



(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

Fig. 8 CPU chain for investigating scalability

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 [5, 16]. In this experiment, we consider not only the buffer constraint Φ , but also a bound α_{S2}^{in} for the output of \mathcal{M} imposed by Subsystem 2. We assume that α_{S2}^{in} is a periodic stream with jitter, specified by the parameters $p_2^{in} = 5$ ms, $j_2^{in} = 10$ ms, $d_2^{in} = 2$ ms, 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 α_{S2}^{in} we additionally employ two guarding TA (cf. Fig. 3c). We then run the heuristic of Sect. 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 representation of $\alpha_{\mathcal{M}}^{in}$.

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 166 MHz, which we represent by means of an appropriate service curve $\beta_{\mathcal{M}}$. For the computation of $\alpha_{\mathcal{M}}^{in}$ we adopt the methodology presented in [5].

The results of the two analysis approaches are shown in Fig. 7b. 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.

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: $p_1^{out} = 5$ ms, $j_1^{out} = 5$ ms, $d_1^{out} = 3$ ms, and $p_2^{in} = 5$ ms, $p_2^{in} = 15$ ms, $d_2^{in} = 2$ ms. Consider the component \mathcal{M} with the above described load-

dependent behavior and invariant Φ . By means of the same procedure as adopted in (C), we can derive the maximum tolerable input $\alpha_{\mathcal{M}}^{in}$ s.t. the interconnection of \mathcal{M} and Subsystem 2 is safe. The resulting curve is shown in Fig. 7c and demonstrates that \mathcal{M} is safely interconnectable with Subsystems 1, and also with Subsystem 2, as $\alpha_{\mathcal{M}}^{out} \subseteq \alpha_{S2}^{in}$ holds. This latter fact was established using the respective guarding TA for asserting the invulnerability of the bound α_{S2}^{in} when computing $\alpha_{\mathcal{M}}^{in}$. 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 166 MHz, whereas \mathcal{M}'' represents a CPU running constantly at 200 MHz. By repeating the analysis procedure for these two components we obtain the maximum input bounds $\alpha_{\mathcal{M}'}^{in}$ and $\alpha_{\mathcal{M}''}^{in}$ shown in Fig. 7c. As can be seen, $\alpha_{\mathcal{M}'}^{in}$ is not comparable to α_{S1}^{out} , hence we cannot guarantee that the system works if we use component \mathcal{M}' . In fact, according to the bound α_{S1}^{out} , Subsystem 1 can produce an input event for \mathcal{M}' every 5 ms, whereas \mathcal{M}' processes incoming events at a maximum rate of one every 6 ms. This means that on the long-term the input buffer of \mathcal{M}' can overflow which corresponds to a violation of invariant Φ . It is also interesting to compare the initial parts of the curves $\alpha_{\mathcal{M}}^{in}$ and $\alpha_{\mathcal{M}'}^{in}$. They show that the system can tolerate a short-term burst of events at the output of Subsystem 1 if the slow CPU \mathcal{M}' is used, whereas this is not the case with the faster adaptive CPU \mathcal{M} . The explanation for this apparently implausible fact is simple: In the presence of an input burst the adaptive CPU \mathcal{M} switches to the fast mode and overloads Subsystem 2, whereas this is not the case with the slow CPU \mathcal{M}' . For the second alternative component \mathcal{M}'' , we have that $\alpha_{\mathcal{M}''}^{in} \subseteq \alpha_{S1}^{out}$ which guarantees 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 the overall system.

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. Figure 8a 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 Fig. 8b. In particular, each CPU will execute at f_{low} if there are less than *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 states that the corresponding CPU must not have more than five events in its input buffer. The goal of the analysis is to determine whether for an input bound α_0^{out} (specified by the parameters $p_0^{\text{out}} = 5$ ms, $j_0^{\text{out}} = 20$ ms, $d_0^{\text{out}} = 2$ ms) each invariant Φ_i ($i \in \{1, \dots, 5\}$) holds. We perform the analysis in two different ways and compare the computational effort. In the first analysis approach, we ignore the modularity of the system and build a single TA model for the entire component chain. We couple this model with an event generator $\mathcal{G}(\alpha_0^{\text{out}})$ which produces all streams bounded by α_0^{out} . With this generator we execute all interconnected component models at once and check the individual invariants. In the second approach, we analyze 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_i^{in} 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_1^{in} . 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_1^{in} with α_0^{out} in order to verify whether a connection of this subsystem with an up-streamed system that emits streams bounded by α_0^{out} is safe. In both scenarios the analysis reveals that not all buffer constraints can be met for the given input α_0^{out} . 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, especially when state-based formal methods are used.

6 Conclusion and future work

This article developed a procedure for computing arrival curves which bound the event stream consumed/emitted by a TA-based component implementation as contained in a design of an embedded real-time system. This features the definition of state-less assume/guarantee real-time interfaces for TA-based component implementations. 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 input and output bounding arrival curve. As we integrated these

invariants 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 an interesting feature, as it establishes incremental, i.e., component-wise evolution of system designs for the proposed framework. This is because, for a given 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. In this article we developed the required criteria for deciding the consistency, compatibility and conformance of interfaces and components.

The interface computation which is essential for deriving consistency, compability and conformance and as presented in this article relies on a number of assumptions, e.g., well-formed components, or monotonic trace regions. An interesting option for future work could be to relax these assumptions, for instance by applying techniques for the determination of schedulability regions, such as introduced in [6]. At the time being, it is not yet clear how to meaningfully translate such schedulability regions into arrival curves. Moreover, the extension to the setting of multiple ports per component appears as everything, but not straight-forward.

Acknowledgments This work is funded by the European Union project CERTAINTY under grant number 288175.

References

1. Altisen, K., Moy, M.: ac2lus: bringing SMT-solving and abstract interpretation techniques to real-time calculus through the synchronous language Lustre. In: 22nd Euromicro Conference on Real-Time Systems (ECRTS), Brussels, Belgium (2010)
2. Alur, R., Dill, D.L.: Automata for modeling real-time systems. In: Paterson, M. (ed.) Proceedings of ICALP 1990. LNCS, vol. 443, p. 335. Springer, Berlin (1990)
3. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems. LNCS, vol. 3185, pp. 200–236. Springer, Berlin (2004)
4. Bengtsson, J., Yi, W.: Timed automata: semantics, algorithms and tools. In: Lectures on Concurrency and Petri Nets. LNCS, vol. 3098, pp. 87–124. Springer, Berlin (2004)
5. Chakraborty, S., Liu, Y., Stoimenov, N., Thiele, L., Wandeler, E.: Interface-based rate analysis of embedded systems. In: RTSS 2006, pp. 25–34 (2006)
6. Cimatti, A., Palopoli, L., Ramadian, Y.: Symbolic computation of schedulability regions using parametric timed automata. In: RTSS 2008, pp. 80–89 (2008)
7. de Alfaro, L., Henzinger, T.A.: Interface theories for component-based design. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 148–165. Springer, Berlin (2001)
8. de Alfaro, L., Henzinger, T.A., Stoelinga, M.I.A.: Timed interfaces. In: Sangiovanni-Vincentelli, A., Sifakis, J. (eds.) EMSOFT 2002. LNCS, pp. 108–122. Springer, Berlin (2002)
9. Hendriks, M., Verhoef, M.: Timed automata based analysis of embedded system architectures. In: Proceedings of the 20th

- International Parallel and Distributed Processing Symposium (IPDPS 2006). IEEE, New York (2006)
10. Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., Ernst, R.: System level performance analysis—the SymTA/S approach. *IEEE Proc. Comput. Digit. Tech.* **152**(2), 148–166 (2005)
 11. Lampka, K., Perathoner, S., Thiele, L.: Analytic real-time analysis and timed automata: a hybrid method for analyzing embedded real-time systems. In: *EMSOFT 2009*, pp. 107–116. ACM/IEEE, New York (2009)
 12. Lampka, K., Perathoner, S., Thiele, L.: Analytic real-time analysis and timed automata: a hybrid methodology for the performance analysis of embedded real-time systems. *Des Autom. Embed. Syst.* **14**(3), 193–227 (2010)
 13. Perathoner, S.: Modular performance analysis of embedded real-time systems: improving modeling scope and accuracy. PhD thesis. ETH Nbr: 19648. Pub Nbr: 127. Swiss Federal Institute of Technology Zurich, ETH Zurich (2011)
 14. Simalatsar, A., Ramadian, Y., Lampka, K., Perathoner, S., Passerone, R., Thiele, L.: Enabling parametric feasibility analysis in real-time calculus driven performance evaluation. In: *Proceedings of the 2011 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES 2011*. ACM, New York (2011)
 15. Thiele, L., Chakraborty, S., Naedele, M.: Real-time calculus for scheduling hard real-time systems. In: *Proceedings of the International Symposium on Circuits and Systems*, vol. 4, pp. 101–104 (2000)
 16. Wandeler, E., Thiele, L.: Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In: *EMSOFT 2005*, pp. 80–89. ACM/IEEE, New York (2005)
 17. Weiss, G., Alur, R.: Automata based interfaces for control and scheduling. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) *Hybrid Systems: Computation and Control. Lecture Notes in Computer Science*, vol. 4416, pp. 601–613. Springer, Berlin (2007)