On the Computation and Properties of Real-time Interfaces for State-based Component Models

[Extended Abstract]

Kai Lampka Lothar Thiele

Computer Engineering and Communication Networks Lab, ETH Zurich, Switzerland
{lampka,thiele}@tik.ee.ethz.ch

ABSTRACT

Event arrival curves [13] are an abstract way of characterizing event streams or arrival patterns as used in the context of real-time analysis of embedded systems. One may employ these abstract stream descriptions as part of analytic, assume/guarantee (A/G), real-time interfaces and thereby define the traffic a component is willing to accept, and what it guarantees to emit. This extended abstract presents the machinery for deriving analytic A/G real-time interfaces based on arrival curves from Timed Automata (TA) [2] based component models. Moreover, we develop the criteria for which composition, as well as refinement, of components is safe, i. e.,it does not interfere with the interface-derived properties of the overall system. This way we strictly advocate the component-wise evolution of system designs.

1. INTRODUCTION

1.1 Motivation and Contribution

Today's real-time systems are embedded deeply into the physical environment and are highly integrated, thereby constituting complex systems. These systems have often to deliver safety-critical services, such that correctness of system-level designs is of uttermost importance. For coping with the complexity inherent to modern real-time systems, incremental, i.e., component-wise evolution of system designs, as well as integrated analysis frameworks appear to be an adequate mean. This extended abstract presents the machinery for deriving analytic real-time interfaces based on arrival curves of Real-Time Calculus (RTC) [13] from Timed Automata (TA) [2] based component implementations. Moreover, we develop the essential properties which guarantee that interface derived properties of an overall system are invariant w. r. t. composition of interfaces or components, as well as w. r. t. refinements (substitutions) of components. The usage of interfaces not only features a component-wise evolution of system designs, furthermore, it allows one to handle analytic models, i. e.,models of the RTC, and operational models such as TA in a single framework, e. g. the Matlab-based MPA toolbox [1].

1.2 Related Work

The authors of [8] established the foundation of interface theory, but explicitly excluded timed behaviour from their elaboration. The theory on timed (state-based) interfaces was provided in [9]. The work presented in this paper elaborates on analytic, i. e.,state-less A/G real-time interfaces as established on the foundations of Real-time Calculus (RTC) [13] and presented in [14, 6]. In fact we develop the same contra-variant “curve inclusion” for deciding whether interfaces and components can safely be composed or refined. However, this paper provides a proof that the “inclusion” property also suffices in the case of TA-based component implementations. The authors of [14] solely considered state-less RTC-based component models, adequate for a non-heterogeneous RTC-based system analysis. We also exploit the pattern presented in [10, 11] for coupling TA and RTC-based model descriptions.

2. BACKGROUND THEORY

2.1 Timed automata (TA)

In this extended abstract the level of component models is concerned with TA [2], where we employ the concepts as implemented in the timed model checker Uppaal. Uppaal supports a compositional style of modelling and features Timed Safety Automata extended with variables [4, 5]; we refer to this concept when speaking of Timed automata (TA) in the following. In the following we will use the notation $\mathcal{M} \models \Phi$ for stating that TA $\mathcal{M}$ fulfils some property Φ, where Φ is a property expressed by some extension of a temporal logic, e. g. Timed CTL (TCTL). In the scope of this paper, it is sufficient to assume that Φ refers to some time-bounded reachability property, e. g. a bound on the size of a variable or a bound on a clock measuring the consumption and emission of a pair of events, signals respectively. Such safety properties are sufficient for verifying key performance measures of real-time systems such as buffer sizes, buffer underflows, and event-processing delays. Moreover, it is well known that for TA the verification of such (timed) safety properties is decidable, e. g. see the textbook [3] for details.

2.2 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 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 traces is addressed by tr_e, where a set of possibly infinitely many of such traces constitutes a(n) (event) stream. A set of traces is denoted $Traces$ and $Traces_e$ refers to the set of traces retrieved by applying the respective filtering operation w. r. t. event type e and for all $tr \in Traces$. However, most of the time the event types are clear from the context or irrelevant, hence we do not always explicitly point them out. We also generically speak of (event) traces, (generic) input/output events, and use event labels such as e or o.

Event streams are a well accepted object of real-time system analysis. Commonly they are characterized by standard real-time traffic

models ranging from strictly periodic event occurrences over PJD-streams, i. e.,traces where event occurrences are periodic with jitter and a minimum distance, up to sporadic event occurrences.
The Real-time Calculus [13] provides an alternative characterization of event streams, namely by pairs of upper and lower arrival curves, where we use the notation $\alpha := (\alpha^{low}, \alpha^{up})$. An arrival curve α bounds the number of events seen on any trace of the respective stream as follows:

$$\alpha^{low}(t-s) \leq R(t) - R(s) \leq \alpha^{up}(t-s) \text{ for all } 0 \leq s \leq t.$$

In the above equation R is the function which counts event occurrences (or arrivals) up to time t, i. e.,cumulative (event) counting function. As each event arrival may trigger behaviour within a down-streamed component, arrival curves provide abstract lower and upper bounding functions α^{low} and α^{up} for the resource demand experienced by the component within time interval $\Delta := t - s$. In the following we will use the following denotations w. r. t. arrival curves:

- A trace tr (or stream) whose cumulative counting function R_{tr} is bounded in the above sense is denoted as being bound by α.
- A stream or an arrival curve always refer to a specific type of event. In this paper we assume the presence of a correct mapping of streams and arrival curves w. r. t. their event types, as well as for the mapping of input and output ports where the resp. events are consumed or emitted. For making this explicit we generically speak then of *event-compatible* streams, arrival curves, components or interfaces.

Let $\alpha_i := (\alpha_i^{up}, \alpha_i^{low})$ and $\alpha_j := (\alpha_j^{up}, \alpha_j^{low})$ be two event-compatible arrival curves.

DEFINITION 1 (Curve inclusion): If $\alpha_i^{up}(\Delta) \leq \alpha_j^{up}(\Delta)$ and $\alpha_i^{low}(\Delta) \geq \alpha_j^{low}(\Delta)$ holds for all $\Delta \in [0, \infty)$ one says α_i is bounded by or included in α_j and writes $\alpha_i \subseteq \alpha_j$. In case neither $\alpha_i \subseteq \alpha_j$ nor $\alpha_j \subseteq \alpha_i$ can be established the two curves are denoted as incomparable. □

Example. For exemplification one may refer to Fig. 1. Parameter N_1^{up} of the upper curve α_1^{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_1^{up}. Hence δ_1^{up} defines the minimum distance of two events once a burst had occurred. In this realm the step width δ_1^{low} of the lower curve α_1^{low} models the maximum distance between two events in the stream bounded by α_1^{low}.

2.3 Coupling of TA and RTC

We briefly recapitulate the major principles as developed in [10, 11]. This is necessary as we exploit these patterns when computing the interface descriptions.

2.3.1 *Approximating RTC-conformant curves*

We consider 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:

$$\begin{aligned} \alpha^{up}(\Delta) &:= \min_{i \in I}(N_i^{up} + \left\lfloor \frac{\Delta}{\delta_i^{up}} \right\rfloor) \quad \text{and} \\ \alpha^{low}(\Delta) &:= \max_{j \in J}(0, N_j^{low} + \left\lfloor \frac{\Delta}{\delta_j^{low}} \right\rfloor) \end{aligned} \tag{1}$$

The fact that an upper RTC-curve is subadditive and given that upper curves are assumed to be constructible by a single minimum operation yields that α^{up} has increasing step widths, i. e.,the distances between two jump points grows with Δ. Analogously lower staircase curves α^{low} possess decreasing staircase sizes, i. e.,the distances between two jump points shrinks for larger values of Δ. This feature is 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.

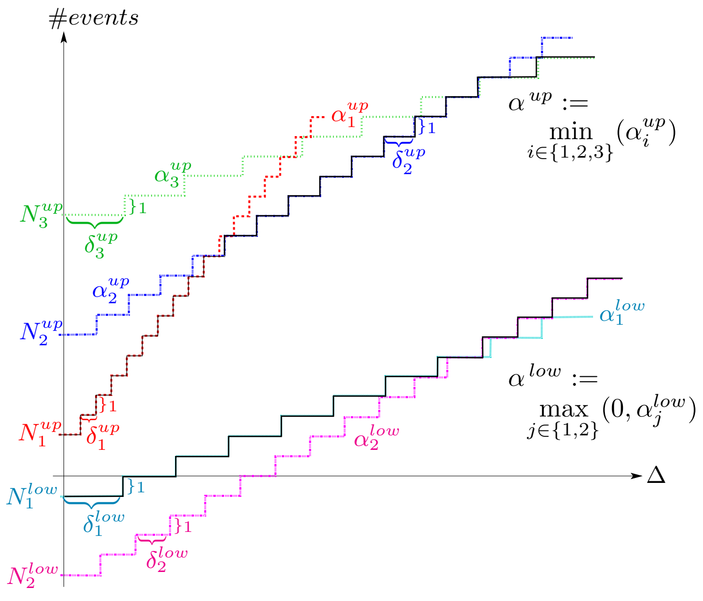


Figure 1: Arrival curves as combination of staircase functions

Example. An example for the construction of a pseudo-concave and convex arrival curve by means of individual staircase curves is presented by Fig. 1: α^{up} is given as the minimum of three staircase curves α_1^{up}, α_2^{up} and α_3^{up}. Lower curve α^{low} is the maximum of the constant 0-function and the two staircase curves α_1^{low} and α_2^{low}.

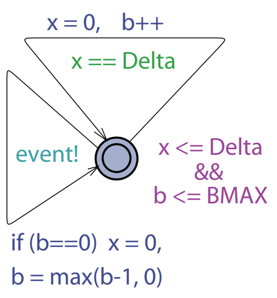
2.3.2 *TA-based modelling of input curves*

Arrival curves of the above kind can be implemented by a set of cooperating TA:

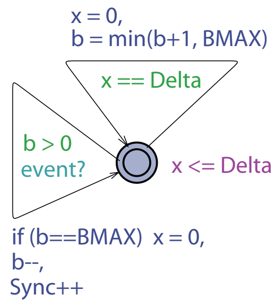
1. Each staircase curve employed in the minimum and maximum operation (cf. Eq. 1) is implemented by its own event emitting TA. In case of a TA guarding upper curve α_i^{up} we speak of UTA i and in case of a TA guarding a lower curve α_j^{low} of LTA j. As main principle each of these TA periodically increase a local counter up to a local threshold. As long as the local counter is greater 0 event emission may take place.
2. Fully synchronizing the UTA on event emission implements the minimum computation of Eq. 1.
3. Maximum building on lower staircase curves (cf. Eq. 1, line 2) is realized by enforcing event emission whenever one of the involved LTA reaches its local threshold.

An example of the pattern as implementable with the Uppaal model checker is presented by Fig. 2.

Sets of LTA and UTA implement an input generator $\mathcal{G}$ emitting traces of a dedicated event type e and w. r. t. some bounding curve α. We use the notation $\mathcal{G}(\alpha)$ or if necessary $\mathcal{G}(\alpha^{up}, \alpha^{low})$.
It is important to note that $\mathcal{G}(\alpha)$ is capable of producing all traces of input events the cumulative bounding functions of which are



(a) LTA for lower curve α_j

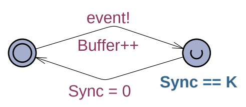


(b) UTA for upper curve α_i

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

Local Declarations for LTA, UTA
const int BMAX = |N_{i,j}|;
clock x;
int b = 0;
const int Delta = δ_{i,j}
```

(c) Declarations



The location marked with **c** is a committed location, i. e.,time does not pass while this location is active and it has to be left on the next edge execution.

(d) Full synchronization

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

bounded by curve α; the proof is provided in [11]. As each event e emitted 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 which is the crossproduct of $\mathcal{G}$ and $\mathcal{M}$ we employ the notation $\mathcal{G}\|\mathcal{M}$. One may note that in case TA $\mathcal{M}$ is not capable of reacting to an emitted event the composite is requested to deadlock. This can be implemented by exploiting Uppaal's concepts of urgent signals and variables [4, 5].

2.3.3 *TA-based detection of output curves*

Executing a stimulated component model together with a set of observer TA $\mathcal{O}$ in a binary search allows one to obtain a set of parameters. These parameters can be used for constructing a pseudo-concave/convex arrival curve which bounds any trace, i. e.,the stream, emittable by $\mathcal{G}(\alpha)\|\mathcal{M}$.

3. ANALYTIC REAL-TIME INTERFACES FOR TA-BASED MODELS

3.1 Definitions

A (single port) component $\mathcal{C}$ is a triple $(In, \mathcal{M}, Out)$, where In is the input port for consuming events of type e, $\mathcal{M}$ is a component model which consumes and emits events of type e, resp, o, and Out is the dedicated output port for emitting events of type o. An analytic (single port) assume guarantee (A/G) interface $\mathcal{I}$ is a triple $(\Phi, \alpha_{in}^{\mathcal{I}}, \alpha_{out}^{\mathcal{I}})$, where

- Φ is a set of dedicated (safety) properties invariantly fulfilled by any component implementing the interface.
- Curve $\alpha_{in}^{\mathcal{I}}$ is the input bound w. r. t. the stream of input events to be processed by component $\mathcal{M}$.
- Curve $\alpha_{out}^{\mathcal{I}}$ is the output bound w. r. t. the stream emitted by component $\mathcal{M}$.

Formally, we relate interface definitions and component models as follows: 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:

1. $\mathcal{M}$ and $\mathcal{I}$ are event-compatible.
2. For the dedicated output event $\circ$ the composite $\mathcal{G}(\alpha_{in}^{\mathcal{I}})\|\mathcal{M}$ only emits traces which are bounded by $\alpha_{out}^{\mathcal{I}}$.
3. For each reachable system state s to be generated by executing the composite $\mathcal{G}(\alpha_{in}^{\mathcal{I}})\|\mathcal{M}$ $s \models \Phi$ holds.

If $\mathcal{M}$ implements $\mathcal{I}$ we write $\mathcal{M} \lhd \mathcal{I}$ and note the **conformance** of component $\mathcal{M}$ and interface $\mathcal{I}$.
Verifying if $\mathcal{M} \lhd \mathcal{I}$ applies is straight-forward: one encodes the input/output curves of the interface by the approach illustrated in Sec. 2.3. Testing the validity of $\mathcal{G}\|\ \mathcal{M}\|\ \mathcal{O} \models \Phi$ allows to decide whether $\mathcal{M} \lhd \mathcal{I}$ holds or not.

Given an interface invariant Φ to hold for a component, one may, alternatively, execute a binary search for either computing the bound on a component's input stream or on its output stream. This will be outlined in greater detail below, where an illustration of the scheme is provided by Fig. 3. The proposed scheme allows one to derive an analytic A/G interface for the TA-based component models contained in the overall system model; for the RTC-based component models one may follow the approach of [14, 6].
Having interface descriptions for all the system's components allows one to establish properties for the system, e. g. required buffer-spaces and experienced waiting times. These properties may hold under component/interface composition and component refinement, where a detailed discussion on this issue follows in Sec. 4.

3.2 Scheme for detecting the input bound

The basic scheme for computing a conservative, pseudo-concave/convex bound for a component's input stream, i. e.,the construction of a pseudo-concave/convex curve $\alpha_{in}^{\mathcal{I}}$ can be organized as binary search. In this search one changes the parameters of a set of UTA which implements an upper curve $\alpha_{in}^{\mathcal{I}}$ and queries a timed model checker, e. g. Uppaal, if $\mathcal{G}(\alpha_{in}^{\mathcal{I}})\|\ \mathcal{M}\|\ \mathcal{O}(\alpha_{out}^{\mathcal{I}}) \models \Phi$ holds. The procedure in finding a curve $\alpha_{in}^{\mathcal{I}}$ can be partitioned as follows:

3.2.1 *Bounding the long-term behaviour*

One may recall that an upper curve α_{in}^{up} is approximated by a set of staircase curves composed via minimum computations. Each of the curves is defined by its burst-capacity N_i and stepwidth δ_i. By means of a timed model checker, e. g. Uppaal, in a binary search one may compute now the smallest stepwidth δ s. t. $\mathcal{G}(\alpha_{in}^{up})\|\ \mathcal{M}\|\ \mathcal{O}(\alpha_{out}^{\mathcal{I}}) \models \Phi$ holds; the (burst-)parameter N is set to 0. At termination the binary search has found the "steepest" long term rate d_{LT} acceptable to the component model. With parameter δ fixed to d_{LT} one may now execute a binary search on (burst-)parameter N. This search delivers the largest value B_{LT} s. t. for $\alpha_{in}^{up}(\Delta) := B_{LT} + \lfloor\frac{\Delta}{d_{LT}}\rfloor$ $\mathcal{G}(\alpha_{in}^{up})\|\ \mathcal{M}\|\ \mathcal{O}(\alpha_{out}^{\mathcal{I}}) \models \Phi$ holds.
In the same way one proceeds for finding a lower curve α_{in}^{low}.

At termination this procedure yields an approximation for an input curve $\alpha_{in} := (\alpha_{in}^{up}, \alpha_{in}^{low})$ bounding *any* trace, resp. the input stream consumable by component model $\mathcal{M}$ and s. t. the interface-defined invariant Φ and the output bound $\alpha_{out}^{\mathcal{I}}$ hold. However, this approximation can be refined, particularly for obtaining a tighter bounding for the short-term streaming behaviour.

3.2.2 *Bounding the short-term behaviour*

For simplicity we focus on the tightening of an upper input bound α_{in}^{up}, as minimum of two staircase curves. In this setting one faces a degree of freedom w. r. t. the parameters of the staircase curves, i. e.,(burst) parameters N_1, N_2 and stepwidth δ_1 are unknown, stepwidth δ_2 is already known, as it is the long term rate d_{LT}. When choosing appropriate values for the free variables 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. According to this a possible initialization could than be $N_1 := B_{LT}$, $N_2 := k \cdot B_{LT}$. What follows next is a binary search for determining parameter δ_1. Inadequate choices are detectable, as the search terminates and $\delta_1 = d_{LT}$ holds. In such cases the search has to be restarted, but at least with a smaller value for N_2. At termination this scheme delivers a pseudo-concave staircase curve α_{in}^{up} bounding the inputs of model $\mathcal{M}$.

3.2.3 *Conservativeness*

One may note that the constructed curve α_{in} is a conservative bound as $\alpha_{in}(s-t) \leq \alpha^{pot}(s-t)$ for all $0 \leq s < t$ holds, with input curve α^{pot} as the actual, but unknown bound on the stream fed into the component.

3.2.4 *Example*

Fig. 3 illustrates the procedure for approx. an unknown input bound α^{pot} with two staircase segments; for illustration purpose Fig. 3 abstracts away that we are actually dealing with staircase functions. At first one searches for the steepest long term rate which is depicted in Fig. 3(a), where δ_{crr} is the current slope to be tested in the binary search. The α_{up} and α_{low} refer to some upper and lower bounding curve those rates ($\delta_{up}, \delta_{low}$) restrict the search space of δ_{crr}. At termination the scheme delivers a value d_{LT}. With δ_{crr} fixed we search now for the largest value for N, which gives us the input (burst) parameter B_{LT} (Fig. 3(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 on the short term streaming behaviour (Fig. 3(c) and 3(d)). As shown in Fig. 3(c) the initial choice for N_2 was to large, hence the search for finding an adequate δ_1 stops unsuccessfully, namely once $\delta_1 = d_{LT}$. In a second run which uses a smaller value for parameter N_2 the finding of δ_1 terminates once $\mathcal{G} \| \mathcal{M} \| \mathcal{O} \models \Phi$ holds.

3.3 Scheme for detecting the output bound

One fixes the input bound by setting it to $\alpha_{in}^{\mathcal{I}}$ and executes a set of observer TA $\mathcal{O}(\alpha)$ together with composite $\mathcal{G}(\alpha_{in}^{\mathcal{I}}) \| \mathcal{M}$ and checks for the validity of interface invariant Φ. Repeating this step in a binary search allows to construct the respective bound for the stream to be emitted by component $\mathcal{M}$.

4. COMPOSITIONAL EVOLUTION OF SYSTEM DESIGNS

The here presented technique aims to support component-wise evolution of systems. Hence the system properties which are derived from the invariants of the interfaces needs to be invariant w. r. t. composition and substitution of interfaces, components respectively. This will be established now, where for simplicity we solely consider the case of single port components and interfaces.

Curve inclusion turns out to be the central property when it comes to component-wise refinement, and component-wise composition, s. t. the respective interface defined invariant is maintained.

THEOREM 1. *Let α and γ be some event compatible input bounds, let α' be some output bound and let Φ be some property. In this setting we have*

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

The above theorem is correct if an input generator $\mathcal{G}(\alpha)$ is capable of producing all traces bound by $\alpha := (\alpha^{up}, \alpha^{low})$ and if $Traces^{\mathcal{G}(\alpha)} \subseteq Traces^{\mathcal{G}(\gamma)}$ holds. The first condition was shown in [11]. Hence we solely elaborate here on trace inclusion, which will be show by contradiction.

PROOF. Let α and γ be the arrival curves implemented by generators $\mathcal{G}_\alpha$, $\mathcal{G}_\gamma$ resp. and let $tr \in Traces^{\mathcal{G}_\alpha}$ and $tr \notin Traces^{\mathcal{G}_\gamma}$. Moreover, let the components be event-compatible.

Let (t, e) be the first timed event of tr which separates its membership from the set $Traces^{\mathcal{G}_\gamma}$. Hence generator $\mathcal{G}_\gamma$ is not capable of producing an event at time t, but so does generator $\mathcal{G}_\alpha$. But with the number of produced events $R_{tr}(0,t)$ being bounded by α we have $R_{tr}(0,t)$ also being bound by γ. Thus a blocking of $\mathcal{G}_\gamma$ as it must have occurred, otherwise $\mathcal{G}_\gamma$ could emit an event, is not possible. Hence such a trace tr can not exists. Consequently $Traces^{\mathcal{G}(\alpha)} \subseteq Traces^{\mathcal{G}(\gamma)}$ holds iff $\alpha \subseteq \gamma$. □

Let A and B be two event-compatible components with their respective interface definitions. The above theorem (informally) implies the following features:

1. Component A and component B can safely be combined, i. e.,put in row, if the event emitter A is more restrictive w. r. t. the bound on the emitted events than the event consumer B. Formally, this means that $\alpha_{out}^A \subseteq \alpha_{in}^B$ has to hold. This directly arises from Th. 1, as for smaller bounds the respective component, here B does not emit more events and respects the interface invariant Φ, as well as the interface-defined output bound.

2. Component B is a safe refinement (or substitute) of component A if it is less restrictive w. r. t. its input bound and more restrictive w. r. t. its output bounds. Formally, $\alpha_{in}^A \subseteq \alpha_{in}^B$ and $\alpha_{out}^B \subseteq \alpha_{out}^A$ has to hold. This follows form the above theorem and from the fact that any down-streamed neighbour of component A maintains its behaviour as substitute B guarantees to produce not more events than component A.

The above properties can be checked on the level of interfaces, rather than dealing with the complete system model each time a component is altered or new signal- or data paths among them are introduced. For asserting the conformance of an interface definition and a (refined) component the machinery introduced in Sec. 3 can be used.

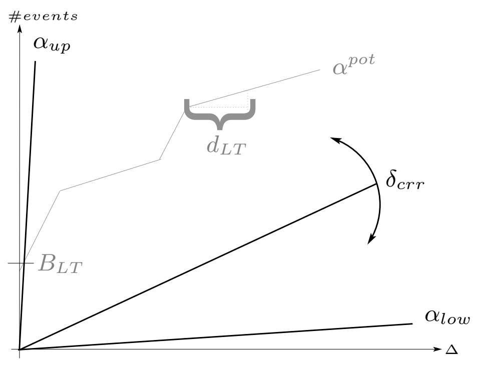
5. CONCLUSION AND FUTURE WORK

5.1 Summary

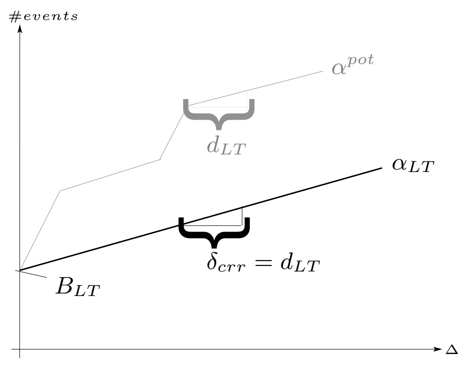
This extended abstract reports on the integration of a heterogeneous modelling landscape in a single framework by advocating the use of analytic assume/guarantee (A/G) real-time interfaces. Component implementations can be hidden as long as their conformance to interface definitions can be formally established. In this abstract we introduced the machinery for deriving analytic assume/guarantee (A/G) real-time interfaces from component implementations, e. g. Timed Automata. Furthermore we isolated the properties of (interface-based) system descriptions which allow for the composition and substitution of components, interface respectively s. t. interface-derived system properties are maintained. This is an interesting feature, as it allows an incremental, i. e.,component-wise evolution of system designs.

5.2 Future work

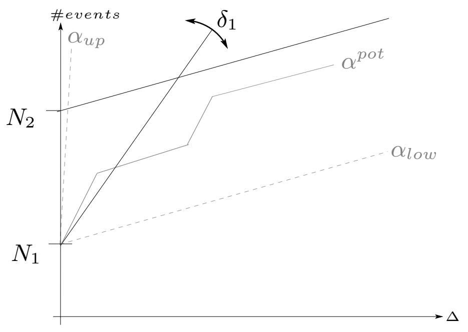
The interface computation presented in this paper relies on a number of assumptions such as well-formed components or existence of a coherent region spanned by the input or output bound. An



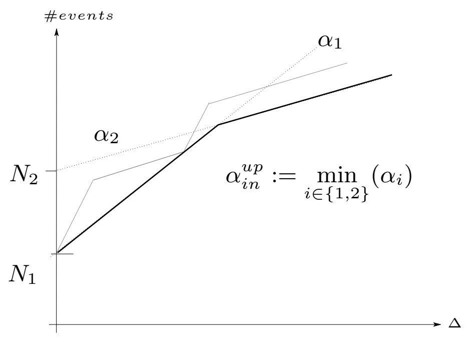
(a) Initialization



(b) Long term rate detected



(c) Search for short term rate



(d) Conservative approx.

Figure 3: Finding an input bound for a TA-based component model

interesting option for future work could be to relax these assumptions, for instance by applying techniques for the determination of schedulability regions, such as derived from parametric Timed Automata [7]. A coupling of RTC and parametric Timed Automata has recently been presented in [12].

Acknowledgement

Funded by the EU project COMBEST, grant number 215543.

6. 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 ICALP 1990*, volume 443 of *LNCS*, pages 322–335. Springer, 1990.

[3] C. Baier and J.-P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[4] 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*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.

[5] 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.

[6] S. Chakraborty, Y. Liu, N. Stoimenov, L. Thiele, and E. Wandeler. Interface-based rate analysis of embedded systems. In *RTSS 2006*, pages 25–34, 2006.

[7] A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *RTSS 2008*, pages 80–89, 2008.

[8] L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In T. A. Henzinger and C. M. Kirsch, editors, *EMSOFT'01*, volume 2211 of *LNCS*, pages 148–165. Springer, 2001.

[9] L. de Alfaro, T. A. Henzinger, and M. I. A. Stoelinga. Timed interfaces. In A. Sangiovanni-Vincentelli and J. Sifakis, editors, *EMSOFT'02*, LNCS, pages 108–122. Springer, 2002.

[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 *EMSOFT'09*, pages 107–116. ACM/IEEE, 2009.

[11] K. Lampka, S. Perathoner, and L. Thiele. Analytic real-time analysis and timed automata: A hybrid methodology for the performance analysis of embedded real-time systems. *Design Automation for Embedded Systems*, 14(3):193–227, 2010.

[12] A. Simalatsar, Y. Ramadian, K. Lampka, S. Perathoner, R. Passerone, and L. Thiele. Enabling parametric feasibility analysis in real-time calculus driven performance evaluation. In *Proceedings of the 2011 Int. Conf. on Compilers, Architecture, and Synthesis for Embedded Systems, CASES 2011*. ACM, 2011.

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

[14] E. Wandeler and L. Thiele. Real-time interfaces for interface-based design of real-time systems with fixed priority scheduling. In *EMSOFT'05*, pages 80–89. ACM/IEEE, 2005.