Enabling Parametric Feasibility Analysis in Real-time Calculus Driven Performance Evaluation

Alena Simalatsar
EPFL, Switzerland
alena.simalatsar@epfl.ch

Yusi Ramadian
University of Trento, Italy
ramadian@disi.unitn.it

Roberto Passerone
University of Trento, Italy
roberto.passerone@unitn.it

Kai Lampka
ETH Zurich, Switzerland
lampka@tik.ee.ethz.ch

Simon Perathoner
ETH Zurich, Switzerland
perathoner@tik.ee.ethz.ch

Lothar Thiele
ETH Zurich, Switzerland
thiele@tik.ee.ethz.ch

ABSTRACT

This paper advocates a rigorously formal and compositional style for obtaining key performance and/or interface metrics of systems with real-time constraints. We propose a hierarchical approach that couples the independent and different by nature frameworks of Modular Performance Analysis with Real-time Calculus (MPA-RTC) and Parametric Feasibility Analysis (PFA). Recent work on Real-time Calculus (RTC) has established an embedding of state-based component models into RTC-driven performance analysis for dealing with more expressive component models. However, with the obtained analysis infrastructure it is possible to analyze components only for a fixed set of parameters, e. g., fixed CPU speeds, fixed buffer sizes etc., such that a big space of parameters remains unstudied. In this paper, we overcome this limitation by integrating the method of parametric feasibility analysis in an RTC-based modeling environment. Using the PFA tool-flow, we are able to find regions for component parameters that maintain feasibility and worst-case properties. As a result, the proposed analysis infrastructure produces a broader range of valid design candidates, and allows the designer to reason about the system robustness.

Categories and Subject Descriptors

[**Embedded systems**]: analysis techniques for embedded system including design space exploration, co-simulation;

General Terms

Design, Performance, and Verification

Keywords

Tool integration, System-level Design, Feasibility areas, Worst-case Analysis

1. INTRODUCTION

System level analysis plays an essential role in the design of hard real-time embedded systems at early stage. Several different methodologies that address the problem of system-level analysis of embedded systems have been presented recently [4, 7, 11, 14, 16, 18, 28, 29]. These methodologies are typically based on some form of abstraction and can often be applied to only a specific or limited set of system architectures or parameter space. Abstraction comes in the form of *analytical* and *executable* models. Analytical models are advantageous in that they generally provide good scalability, particularly if they are compositional as in case of the Real-time Calculus. On the other hand, executable models can often be more accurate as a more general semantic model is inherent to them. Because of the complementary nature of analytical and executable methods, there has been a trend recently in trying to combine techniques from these domains and thereby taking advantage of their respective strengths [27]. In this paper, we present a new approach that allows performance analysis of parameterized systems by combining two standalone methods of Real-Time Calculus [29, 30] and of Parametric Feasibility Analysis [12] and by integrating their respective tool flows. Designers benefit from this integration because a larger portion of the design space can be explored, while still maintaining feasibility and worst-case properties.

The Modular Performance Analysis Toolbox [30], based on the Real-Time Calculus (MPA-RTC) [29], makes use of functions on the time-interval domain to represent both system workload and availability of computation and communication resources. Component interaction is abstractly modeled by sets of functions, instead of signals, tokens or other activity triggers. As this features compositionality also on the level of the formal analysis, MPA-RTC supports a wide and efficient performance evaluation of key metrics of component-based real-time systems. Pessimism contained in the obtained guarantees w. r. t. worst-case system behaviors, e. g., burst-sizes, backlog-sizes and delays, can be avoided as long as the system under analysis matches the model of computation inherent to RTC. However, it may be difficult to adequately model components having state-dependent behaviors, e. g., CPUs whose speed changes with the size of the backlog buffer. For less pessimistic results in such situations, a methodology for embedding Timed Automata (TA) [1] based component models into a MPA-RTC-based system model [20, 21] was developed recently. However, with this approach it is possible to study only fixed values of parameters, e. g., a fixed CPU speed, fixed buffer sizes, and fixed parameters of functions modeling the component/environment interaction. Hence, a big space of feasible parameters values remains unstudied.

On the contrary, Parametric Feasibility Analysis (PFA) performs feasibility analysis for a system associated with a set of parameters for their full range of possible values. These parameters can describe the activation pattern, the real-time constraints and the ex-

CASES'11, October 9–14, 2011, Taipei, Taiwan.

ecution parameters of the concurrent software entities (or tasks) scheduled on one or more shared CPUs. Schedulers and activation patterns are represented with Parametric Timed Automata (PTA). Some of the parameters are known, others are known with a limited precision (e.g., computation times and activation rates). PFA allows one to find the region of the set of parameters that make the system feasible, i.e., it satisfies the real-time constraints expressed by the model.

The main contribution of this paper lays in the coupling of the PFA and the MPA-RTC frameworks which enables one to find the feasibility area for the set of parameters of the MPA-RTC-based component through its TA extension. This leads to optimized implementations both in terms of performance and in terms of their robustness. We organized the paper as follows. In Section 2 we report on related and preliminary work. Section 3 describes the fundamental aspects of the underlying theories. The tool integration is introduced in Section 4. Section 5 presents a case study to demonstrate the effectiveness of the proposed approach and Section 6 concludes the paper.

2. RELATED WORK

The literature on classical real-time feasibility theory is vast [6, 8, 22]. Wang Yi et al. [24] were the first to use a Timed Automaton (TA) to represent a system resource (CPU or communication element) as a scheduler model together with a notion of discrete events that trigger the execution of a real-time tasks executable on this scheduler. In this work they also show that the feasibility problem of this extended model can be transformed into a reachability problem for a standard TA. This approach allows one to check whether a system is feasible for some fixed set of parameters. However, in early system design phases some of the parameters can be unknown or known with a limited precision. Therefore, this idea was extended by using Parametric Timed Automata (PTA) [12] manipulated by Uppaal [5] and NuSMV [10]. As a part of our tool integration we have extended the approach based on PTA to compute the feasibility region of systems characterized by not only periodic tasks, but also periodic tasks with jitter. We refer to this extension as Parametric Feasibility Analysis (PFA).

In another line of work [3], André shows how given a certain known parameter valuation, one can compute the relaxed constraint on the parameter space that will have the same trace. This approach depends on the input, is not maximal and not guaranteed to terminate. In [33], the authors present a method to model check a real time system using parametric timed automata when a constraint over the parameter is *given*. This problem is referred to as the "emptiness" problem for `PTA` and has been proven undecidable, in the general case, in [2]. Interestingly, in [17] a subclass of `PTA` called `L/U PTA` is identified for which the problem is decidable, and a solution strategy based on Parametric Difference Bound Matrices is proposed. The solution to the emptiness problem is actually part of our method in getting the reachability of the error state that we use to solve a more general problem.

The above techniques are well accepted methodologies for analyzing real-time systems, e.g., see [23] for a recent example. However, the finite state/transition system to be derived from some high-level model tends to grow exponentially with the number of clocks and clock constants. Therefore, the detailed analysis of a complex system may be hampered in practice, if not impossible at all. In contrast, purely analytic (or stateless) methods such as provided by the Real-Time Calculus [29, 31], SymTA/S [16] or MAST [15] solely depend on solutions of closed form expressions, yielding a very good scalability with the size of systems to be analyzed. But this advantage has its costs: (a) analytic methods are limited to the computation of a few specific system measures and (b) each method is restricted to a specific model to which the system specification under analysis must be translated, which in general may lead to overly conservative analysis results.

To overcome these shortcomings, recent research has made an attempt at comparing tools that do similar type of analysis [25], as well as elaborated on a combination of purely analytic and state-based performance analysis methods [13]. In one proposed method, Phan et al. [26] elaborate on an embedding of event count automata (ECA) [9] into RTC-driven performance analysis. In this approach, the user must specify, for each location of the ECA, the minimum and maximum number of event arrivals taking place while the ECA resides in that location. As a consequence of the implicit notion of time, the modeling of systems with time dependencies may yield a severe blow up of the computational effort for their state-space oriented analysis. To interface ECA to MPA-RTC framework and vice versa, one needs to derive abstract stream representations, i.e., their bounding RTC-curves. In this approach, this is accomplished through the use of observer ECA. The analysis uses binary search for extracting the maximum and minimum number of events seen in a window size Δ via reachability analysis. A set of point values obtained from the binary search allows one to construct a (pessimistic) RTC-curve. While one may potentially analyze parametric systems in this way, binary search relies on certain monotonicity properties of the model, a limitation which is not present in our approach based on parametric feasibility analysis.

Krcal et al. [19] also address the combination of RTC-based components and timed automata. For including the abstract stream representation used in RTC into TA-based system models one operates on an array of clocks. Each clock is associated with the number of events produced so far, as well as with a minimal and maximal number of events to be generated within the respective time interval length. For deriving an RTC-based stream representations from the combined model, the authors suggest the usage of observer automata and binary search on the maximal and minimal number of events that appear within any time interval of length Δ, which is in fact similar to the idea cited previously [26], and suffers from the same limitations.

In this paper, we exploit the pattern, recently presented in [21], for embedding TA-based model descriptions into RTC-driven performance analysis. This provides us with a translation path between the PTA-based feasibility analysis and the RTC-based performance analysis of the MPA-framework [30, 31], since PTA is an extension of classical TA. Contrary to the existing work, the usage of PTA proposed here allows one to obtain regions of RTC-curve parameters rather than individual values. This of course holds also for the features of the embedded component model, as regions or bounds on execution speeds, buffer sizes etc. can also be obtained, rather than point-estimates as detected by binary search when carrying out the techniques presented in [19, 21, 26].

3. BACKGROUND THEORY

In this paper we present a hybrid methodology that unifies two different approaches to complex real-time system analysis. This section briefly introduces the background theory necessary to better understand the main ideas.

The two techniques employed in this work, Real-Time Calculus (RTC) and Parametric Feasibility Analysis (PFA), share the view of the system as a composition of computational and communication components (resources). Tasks are executed on the computational elements and data packets are sent through the communication element. The order of task execution or data transmission is defined by a chosen scheduling policy, which is implemented inside the com-

ponents. In our representation, we do not distinguish between the computational and communication elements, as well as between tasks and data packets, and model each resource as a Greedy Processing Component (GPC) which is never idle unless there are no more tasks to be scheduled. The task activation, whether it is processed directly or added to the scheduling buffer for further processing, is marked with a start or input event, while the end of the task execution is marked with an end or output event.

Timed event traces for input and output events (see Section 3.1) are used to represent both the task activation pattern of a component and the pattern of output events. In multi-component systems such a pattern of output events may become an input pattern to down-streamed components. Timed event traces can be characterized by a pair $(\alpha_{in}^{low}, \alpha_{in}^{up})$ of arrival curves in the analytic approach. The translation of these curves into event-emitting cooperating Timed Automata (TA) [1] is presented in Section 3.3, as TA form the foundation of the state-based component model descriptions. Finally, Section 3.4 presents the Parametric Feasibility Analysis (PFA) framework and Parametric Timed Automata (PTA) as the underlying model of the PFA.

3.1 Arrival curves of Real-time Calculus

A timed τ-event is a pair (t, τ) where $\tau \in Act$ is some event label or type from a set Act of event labels or types, and $t \in \mathbb{R}_{\geq 0}$ is some non-negative time stamp. A timed trace or stream $tr := (t_1, \tau); (t_2, \tau); \ldots$ is a sequence of timed τ-events ordered by increasing time stamps, i.e., such that $t_i \leq t_{i+1}$. A stream tr can be abstractly characterized by a pair $(\alpha_{in}^{low}, \alpha_{in}^{up})$, of arrival curves [29]. These arrival curves bound the number of events seen on tr for any interval $\Delta \in [0, \infty)$. Let $R(t)$ denote the number of events that arrived on the stream tr in the time interval $[0, t)$. The upper and lower arrival curves must satisfy the following equation

$$\alpha_{in}^{low}(t-s) \leq R(t) - R(s) \leq \alpha_{in}^{up}(t-s) \text{ with } 0 \leq s \leq t. \quad (1)$$

As each event from a stream may trigger some behavior within a component, the arrival curves α_{in}^{low} and α_{in}^{up} can be seen as abstract lower and upper bounds of the amount of resources demanded for processing the events within a time interval $\Delta := t - s$. Furthermore, a given pair $(\alpha_{in}^{low}, \alpha_{in}^{up})$ represents a (possibly infinite) set of streams of input events, task activations resp., as it includes all streams whose counting function R satisfies (1). For simplicity, we denote the pair $(\alpha_{in}^{low}, \alpha_{in}^{up})$ as α_{in}. A stream whose cumulative counting function satisfies (1) is said to be *bound* by α_{in}. Arrival curves are a generic way for characterizing standard real-time traffic models ranging from strictly periodic event arrivals over PJD-streams (traces where event arrivals are periodic with jitter and with a minimum distance), up to sporadic event arrivals. In this work we restrict our attention to the case of *complete amounts* of events only, i. e., the case in which the number of events to be processed is an integer value. In RTC, such scenarios are modeled by staircase curves, as those shown in Fig. 1.

In this paper we model a staircase curve as the combination of elementary staircase curves composed via nested minimum and maximum operations. For the sake of clearness, we restrict the nesting of minimum and maximum operations and assume that upper arrival curves are always obtained as the minimum over a set I of staircase curves, and lower arrival curves as the maximum over a set J of staircase curves:

$$\alpha_{in}^{up}(\Delta) := \min_{I}(\alpha_i^{up}(\Delta)); \; \alpha_{in}^{low}(\Delta) := \max_{J}(0, \alpha_j^{low}(\Delta)) \quad (2)$$

where the elementary staircase curves $\alpha_i^{up}(\Delta)$ and $\alpha_j^{low}(\Delta)$ are of

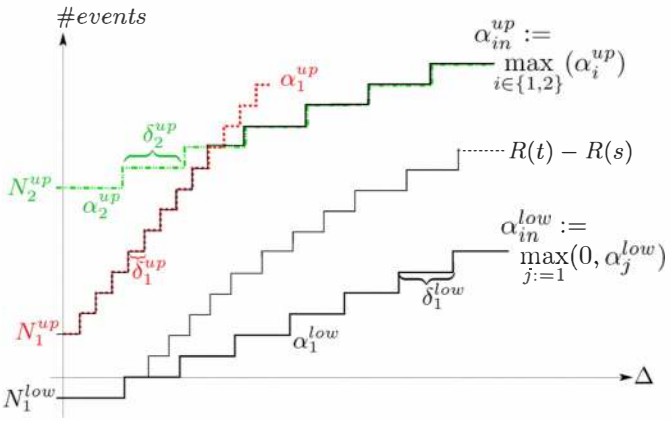


Figure 1: Upper and lower (staircase) arrival curves

the form

$$\alpha_{\{i,j\}}^{\{up,\,low\}}(\Delta) := N_{\{i,j\}}^{\{up,\,low\}} + \left\lfloor \frac{\Delta}{\delta_{\{i,j\}}^{\{up,\,low\}}} \right\rfloor. \quad (3)$$

The fact that upper (and lower) RTC curves are constructed by a single minimum (and maximum) operation (not nested min/max computations) considerably reduces the complexity of embedding TA into the RTC-driven performance analysis.

Example. An example is given in Fig. 1: the upper arrival curve α_{in}^{up} results from the minimum of two staircase curves. Its parameter N_1^{up} models the (burst) capacity, i. e., it is the number of events that can arrive at the same instant of time in any stream bounded from above by α_{in}^{up}. Its parameter δ_1^{up} yields than the minimum distance of two events once a burst had occurred. The lower curve α_{in}^{low} consists of the maximum of the constant 0-function and some staircase function α_1^{low}. Its step widths define the maximum distances between successive events bounded from below by α_{in}^{low}.

3.2 Timed Verification

Let C be a set of clocks and let $ClockCons$ be a set of constraints on these clocks. The constraints are conjunctions, disjunctions and negations of atomic (clock) guards, where the later are of the form $x \bowtie n, x \in C, n \in \mathbb{N}_0$ for $\bowtie \in \{<, \leq, >, \geq, =\}$. A *timed automaton* (TA) [1] is defined by a tuple $TA = (Loc, Loc_0, Act, C, \hookrightarrow, I)$ with Loc as the finite set of locations, $Loc_0 \subseteq Loc$ as the set of initial locations, Act as a the set of action (or edge) labels, C as the finite set of clocks, $\hookrightarrow \subseteq Loc \times ClockCons(C) \times Act \times 2^C \times \times Loc$ as its edge relation and with $I : Loc \rightarrow ClockCons(C)$ as the map of invariants which maps locations to clock constraints.

Let the active location be the location the TA currently resides in. The operational semantics associated with a TA can be informally characterized as follows:

Delay transitions. While location invariants of the active location(s) hold time may progress with all clocks increase at the same speed.

Edge executions. Traversal of an edge within a TA potentially changes the set of active locations; self-loops are possible. Such an edge "execution" is instantaneous and possible at any time, given that the source location of the edge is marked active and its guard evaluates to true. Upon edge executions clocks can be reset.

The above setting yields infinitely many behaviors, because an edge execution may occur at any time within the bounded interval of reals defined by location invariants and (clock) guards. However, with the concept of clock regions [1] it is possible to capture all possible behaviors in a finite state graph, such that timed reachability is decidable. In fact, modern timed model checkers incorporate *clock zones* as they often result in a coarser partitioning of the clock evaluation space in comparison to the original definition of clock regions. For conciseness we omit further details, and refer the interested reader to the literature on this topic [1, 32].

We exploit the timed model checker Uppaal [5], which implements timed safety automata extended with variables. Similarly to clocks, variables can be used within guards of edges and location invariants. Upon an edge execution, a variable can be updated, where the used values must built a finite set, which, however, does not need to be known on beforehand, i. e., it can be constructed on-the-fly upon the state space traversal. As we are also exploiting Uppaal's cooperation mechanisms we briefly recapitulate them.

Mechanisms of cooperation

A system model can be composed from a set of cooperating TA. These TA may either share (global) variables or may jointly execute dedicated edges, denoted as rendez-vous or synchronization:

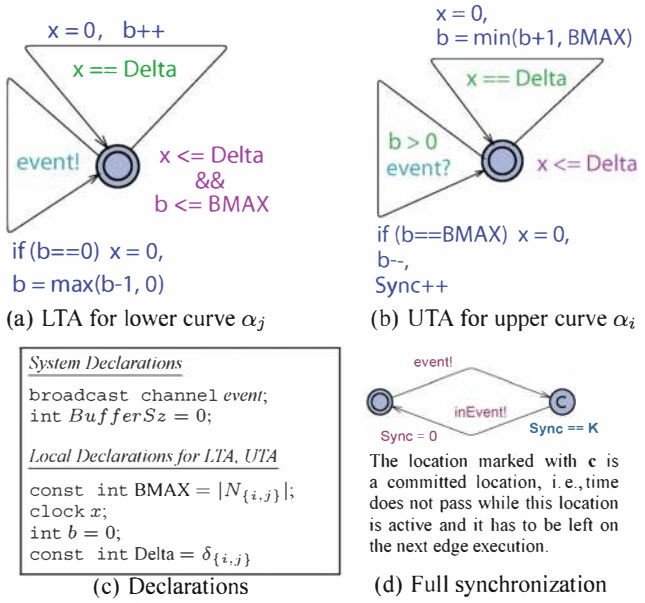
(A) Cooperation via shared variables.
Variables can be declared on the level of a network of TA, allowing the individual TA to read and manipulate them. In this context, it is important to note that the manipulation of variables which takes place upon the traversal of synchronizing edges needs to be commutative, as the order of the edge traversal of the participating TA is non-deterministic.

(B) Rendez-vous mechanisms.
Uppaal makes use of channels and signals. In fact this implements different rendez-vous mechanisms. TA may jointly execute their enabled edges if the involved edges carry the same (channel) identifier followed by a question or exclamation mark. The TA providing an edge labeled with a exclamation mark is commonly denoted sender, whereas the TA providing the edge with the question mark is denoted receiver. As the number of senders and receivers may vary, different rules apply:

Binary synchronization. If there are n senders and k receivers the state space generation engine picks non-deterministically a pair of sender and receiver and jointly executes their resp. edges. A sending or receiving edge is not allowed to be executed in isolation, i. e., if a sender or receiver has no resp. counterpart its sending/receiving edge is blocked from execution.

Arbitrary, n-ary synchronization. A broadcasting send is non-blocking, i. e., a broadcasting sender TA executes its sending edge and between 0 and n receivers execute one of their resp. receiving edge at the very same time. It is important to note, that each receiver with at least one enabled receiving edge has to execute the latter.

Full, $1 : n$ synchronization. Global variables and broadcast channels allow one to enforce a joint execution among one sender and n receivers which is achieved as follows: each receiver increments a global counter. The sender requires that after executing the sending edge the global counter equals some predefined constant. This is guarded by the location invariant of the target location of the resp. (broadcasting) edge.



(a) LTA for lower curve α_j (b) UTA for upper curve α_i

(c) Declarations (d) Full synchronization

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

3.3 From RTC-curves to event emitting TA

Upper and lower RTC-based staircase curves can be implemented by sets of cooperating TA, where the emitted event signals serve as input stimuli to the down-stream TA-based scheduler model. The different TA are coordinated by making use of broadcast channels as implemented in Uppaal. With our approach, we translate RTC staircase curves into an event-emitting TA as follows:

1. Curve parameters $\delta_{\{i,j\}}$ and $N_{\{i,j\}}$ employed in Eq. 3 are either chosen fixed or are in the parameter set of the TA. Each staircase curve $\alpha_{\{i,j\}}^{\{up,low\}}$ (Eq. 2) is implemented by its own event emitting TA, where in case of an upper curve α_i^{up} we speak of a timed automaton UTA i and in case of a lower curve α_j^{low} of a timed automaton LTA j. The generic implementation of a UTA and LTA within the timed model checker Uppaal is shown in Fig. 2 (a) and (b).
2. Full synchronization on the UTA which implements minimum computation on the set of upper event arrival curves (cf. Eq. 2) is enforced by the event emitting TA as depicted in part (d) of Fig. 2.Full synchronization means that all UTA have to execute their $event$-labeled edge in order to allow the event-emitter (TA of Fig. 2 (d)) to release an event `inEvent!`. If not all UTA can participate in the synchronization, event emission is blocked.
3. Maximum building on lower staircase curves (cf. Eq. 2) is realized by enforcing event emission whenever one of the involved LTA has reached its local threshold N_j, which is achieved by defining the respective invariant for each LTA.

A set of LTA and UTA cooperating via the event emitting TA implements an input generator $\mathcal{G}$ which produces streams w. r. t. a dedicated event type, e. g., of type $inEvent$. Generator $\mathcal{G}(\alpha_{in})$ allows to produce all streams whose cumulative bounding functions are bounded in the sense of Eq. 1 (the proof is provided in our previous work [20]). The events emitted by generator $\mathcal{G}(\alpha_{in})$ can now be employed for triggering subsequent behavior in any downstream TA, e. g., the scheduler model.

Example. An example is shown in Fig. 1 and 2. For translating upper curve α_{in}^{up} one needs 2 UTA instantiated with $BMAX :=$

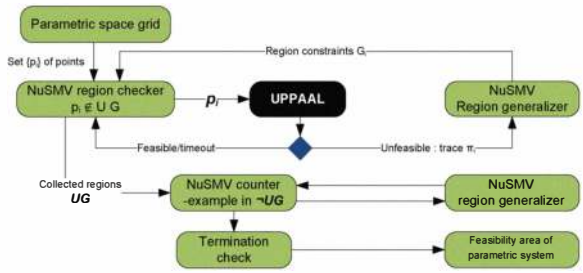


Figure 3: PFA tool chain

$N^{up}_{\{1,2\}}$ and $Delta := \delta^{up}_{\{1,2\}}$. For the event generator (Fig. 2 (d)) constant K is set to 2. For implementing the lower curve only a single LTA is needed.

3.4 Parametric Feasibility Analysis (PFA)

Parametric Feasibility Analysis (PFA) performs an analysis of schedulability for a collection of processes running on a CPU. This method identifies the feasibility region over the space of parameters of the processes. To do this, the system is modeled as software components (sets of tasks) that compete for one or more shared CPU. Each component is associated with a set of parameters that describe the activation pattern, the real-time constraints and the execution parameters of the different tasks. Some of the parameters are known. Others are known with a limited precision (e. g., computation times and activation rate). The objective is to find the region of the set of parameters that make the components feasible, or schedulable. The proposed technique has been described in [12].

The main idea behind the parametric feasibility analysis is the Parametric Timed Automata (PTA). Parametric Timed Automata extends standard Timed Automata [1] as follows: let X be the disjoint union of the finite set of clocks and the finite set of state variables that could be boolean or real variables. Moreover, let P be a finite set of parameters. The finite set $\mathcal{C}(X \cup P)$ contains the (complex) constraints over the variables and clocks of set X and the parameters of set P, once again formed by disjunction, conjunction and negation of atomic guards. A *Parametric Timed Automaton* is defined by a tuple $PTA := (Loc, Loc_0, Act, X, P, \Gamma, \hookrightarrow, I)$ where Loc, Loc_0 and Act are defined as before and where the sets X and P are defined as above. $\Gamma \subseteq \mathcal{B}(P)$ is the parameter space, where $\mathcal{B}(P)$ denotes the set of boolean combinations of constraints over P. The set of edges $\hookrightarrow \subseteq Loc \times Act \times \mathcal{C}(X \cup P) \times 2^{\mathcal{U}(X)} \times L$ is now enriched with complex constraints on clocks, state variables and parameters (elements of $\mathcal{C}(X \cup P)$) and sets of update statement from $U(X)$ that can reset the clocks to any value (not just 0 as in classical TA) or term. Alternatively, the clock can simply grow linearly in time in each location. The value of state variables can be changed only as a result of a reset action, i. e., when an edge is executed. Last, $I : Loc \rightarrow \mathcal{C}(X \cup P)$ is now the invariant map between locations and constraints over clocks, variables and parameters.

We first construct a model for the feasibility problem based on Timed Automata, along the lines defined by Wang Yi [24]. Some details are presented in Section 4.1. In PFA, we extend this approach by using cooperating Parametric Timed Automatae (PTA), which are manipulated by Uppaal and by the NuSMV tool [10]. The analysis performed by the tools goes through three phases: trace generation, trace analysis, and trace projection. During trace generation, traces from the initial to the error state in the PTA are generated. Each of these traces is an instantiation of a parameter

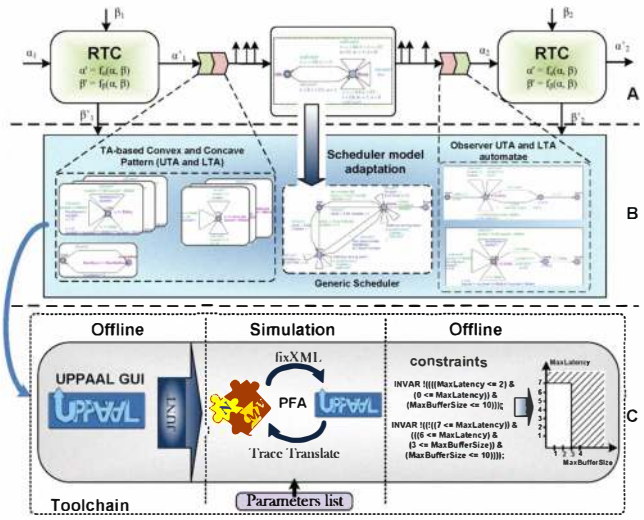


Figure 4: Overview of analysis approach

valuation in which the system is unfeasible. During trace analysis, these traces are evaluated in the context of the symbolic representation of the PTA and a symbolic formula describing the PTA reaching an error state is formulated. The trace projection phase projects this symbolic formula into the parametric space to derive constraints that describe an unfeasible region around the point. Thus, a single trace is generalized into a potentially large region, increasing the efficiency of the method. Because the region is unfeasible (since the system reaches an error state), it is ruled out from subsequent searches. In this way, the feasibility region is iteratively bounded. For a particular but important case it has been shown that this algorithm terminates in a finite number of steps [12].

The flow of the method that we currently employ is shown in Fig. 3. Given a PTA, and the parameter boundaries (the parameters space has to be limited for termination), we perform a quick feasibility check on a set of sample points in the parameter space $\{p_0, p_1, \ldots, p_n\}$ using non-parametric model checking of Uppaal. If a point p_i is unfeasible, an error trace is produced. Trace analysis and projection with NuSMV results in a region G_i. We collect this result as the set of unfeasible regions $\cup G$. Then we repeat this process for all sample points $\{p_{i+1}, p_{i+2}, \ldots, p_n\}$ that are not yet contained in $\cup G$. When the sample point search no longer returns new unfeasible regions, NuSMV is used to search for more error traces using Bounded Model Checking (BMC) in the remaining area, until no more error traces can be found. The feasible region of the PTA is then obtained from the negation of the collected unfeasible regions.

4. TOOL INTEGRATION

The tool integration flow is presented on Fig. 4. The figure is divided into three sections. Section A and B refer to the system model, while Section C shows how the tools are integrated.

4.1 System model

The top section A presents a system composed of three processing components connected in a dataflow manner. Two of these components, the first and the last, are RTC-based, while the middle component is TA-based. The output curves of the first component act as the input curves for the second one. Because the second component is TA-based, these curves are translated into the equivalent TA-based event generator (see Section 3.3), shown in the middle

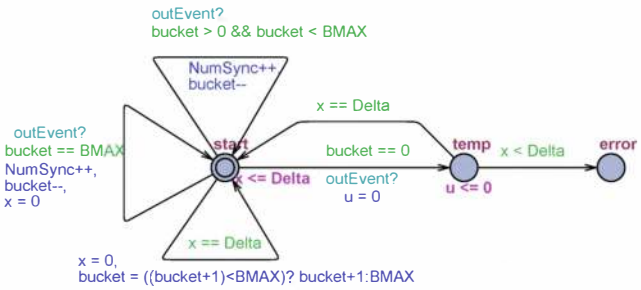


Figure 5: Observer UTA

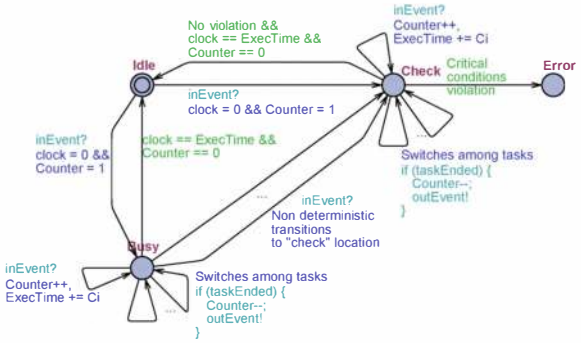


Figure 6: Generalized Scheduler Model

section B of Fig. 4. The tasks are then processed by the second components represented by a TA-based scheduler. The end of each task processing is marked with an output event that is then captured by two observer TAs, which model the output curves. These TAs set constraints on the output events in terms of burst sizes and minimum/maximum distance of events. An example of an upper observer automaton is given in Fig. 5, where the error state models the violation of the currently tested output event curve.

Similarly to the observer TA, in order to verify that the constrains on task scheduling and execution are satisfied, the scheduler TA of the second component needs to be extended with a check and an error states. A simplified generalized scheduler model required by the PFA tool is presented in Fig. 6. Let us first discuss only a part (*idle* and *busy* states) of the generalized scheduler model. The main idea behind this model is that we may represent any resource as a two-state TA. In one of its state the TA is *idle*, i. e., available for the tasks, while in the other state it is *busy*, locked for all tasks or tasks with lower priority for non-preemptive or preemptive priority-based scheduling policies, respectively. This TA is idle in its initial state until a first event (*inEvent?*) from the event generator described above arrives. Then it switches to the *busy* state, resetting on the transition the *clock* used to account for the time passed in the *busy* state. Since other events may arrive while the resource is busy executing the current task, the total remaining execution time will be recalculated and/or event counters incremented. The end of each task execution is marked with an output event broadcast to the system through the *outEvent* channel. Emission of output events is hereby tracked the observers TA described above (cf. Fig. 5). The scheduler can not go back to the *idle* state unless no more events are stored in the buffer and the system clock is equal to the total execution time. The self-transitions of the *busy* state (task switching conditions) are specific for a chosen scheduling policy.

To conduct feasibility analysis, the two-state *idle-busy* model should be extended with two more states: *check* and *error*. In the *check* state, the model functions similarly to when it is in the *busy* state, i. e., reacting to the arrival of new events, accounting for the remaining execution time and switching among tasks. In addition to that, it checks the violation of critical conditions (e. g., task deadline violation). When the condition of the critical cases is satisfied, the transition to the *error* state will be taken. The example shown in Fig. 6 refers to the extended model of a scheduler with load-dependent frequency adaptation.

4.2 Integrated Tool Chain

The set of TA-based models includes a number of parameters that characterize the performance of the system in terms of deadlines, execution times, and in terms of the shape of the input and output arrival curves. The system is feasible for a given choice of parameters if the error state of the timed automata in the model is unreachable. We use the parametric feasibility analysis tool to find out the region of parameters for which the system is feasible.

The tool chain is shown in Fig. 4, Section C. In this chain we make use of the Uppaal toolbox as part of the MPA-RTC/PFA integration flow, used as one of the translation chain links. Three TA-based models, the input event generator, the generalized scheduler and the two observer automata, are initially represented in the Uppaal toolbox. However, the PFA tool requires a symbolic representation of the TA using the NuSMV language. The conversion from the Uppaal model into the NuSMV model is achived through the Java Uppaal to NuSMV Translator (JUNT, which was developed for this application. The .XML file containing the Uppaal model is taken as input by the JUNT translator. The JUNT output file (.SMV) is the executable NuSMV file that is accepted by the PFA tool. The translation is done in "offline" mode and the .SMV file is then given to the PFA tool.

In order to speed up the analysis, the PFA tool employs both the NuSMV engine and the Uppaal engine. Thus, it is essential to be able to run slightly modified TA models back in Uppaal. We have therefore established a feedback loop from NuSMV to Uppaal that assigns the PFA defined values to the variables of the initial Uppaal models that were chosen as parameters for the PFA. This feedback loop is graphically represented in Fig. 4. The modified Uppaal model with changed variables values is then executed on the Uppaal model checking engine. After that, the traces to the error sates (if reachable) are translated back to NuSMV traces for further analysis.

The first step of the feedback loop to Uppaal is implemented as a Perl script that takes as input an initial Uppaal model and a .txt configuration file that includes a list of parameters (Uppaal variables) names and the fixed values that need to be assigned to them. The backward step is easier, since only a trace must be translated between two rather similar formats.

The results of the PFA analysis is actually a simple set of constrains on the set of parameters that can be then manually translated into a graph of a feasibility region. Any point of the feasible region (e.g., specific values of $N_{1..k}, \delta_{1..k}, N_{Obs}$, and/or δ_{Obs}) can then be given back to the MPA-RTC toolbox to adjust the system according to the given constrains.

From a user point of view, the use of Uppaal in the tool chain has the additional advantage of providing a graphical interface, which is more attractive over the text based NuSMV language for someone who would like to build new TA models. The JUNT translator is therefore useful in its own right, and has been designed to be more generic than simply translating event generators and schedulers. An XML file is used as a native Uppaal file format. We made use of the internal Uppaal XML parser in order to translate

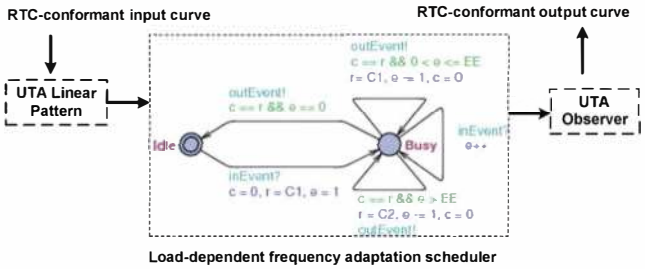


Figure 7: Analyzed System

the graphical user interface TA representation into an executable NuSMV file. The challenge in developing the translator was to match the explicit state model used in Uppaal to an equivalent implicit constraint based model, as used in NuSMV. Some difficulties arise since clocks and integer variables of Uppaal are represented in NuSMV as real values. Arrays are particularly challenging to translate, because array support in NuSMV is very limited. For example, when arrays are used to represent the transition guards or assignments in Uppaal, we are forced to introduce a number of transitions equal to the array size in the NuSMV module. An even more difficult situation occurs when the array size is defined by a template parameter that can vary from one instantiation to another. In that case, different instantiations of the same template must be translated into different NuSMV modules, increasing the size of the code.

4.3 Analysis flows

As discussed, parameters may appear in any part of the timed automata system model. Typically, the analysis is conducted by fixing the value of some of the parameters, and determining the region of feasibility of the remaining free parameters. For example, we can fix parameters of the input curves ($N_{1..k}$, $\delta_{1..k}$) and find the region of the output curves parameters (N_{Obs}, δ_{Obs}) that make the system satisfy the constraints, or vice versa. We call the analysis type performed while fixing the input curves parameters *Forward Analysis*. Likewise, one can fix the parameters of the output curves, and find the feasible region of the parameters of the input arrival curves. We call this type of analysis *Backward Analysis*. It is also possible to fix both input and output parameter values in order to find an area for such parameters as the buffer size, maximum execution latency and/or max execution time, which characterize the scheduler model. We call this *Processing Element Analysis*. Points in the feasible regions can then be selected, based on the desired performance metrics and robustness to parameter variations, to derive the appropriate actual curves conforming to the RTC formalism.

5. CASE STUDY

The example that we are going to show in this section is a simplified version of the general case presented in Section 4.2 and is presented in Fig. 7. We consider a system with one type of task having an input arrival pattern that is periodic with jitter. The curve characterizing this pattern can be modeled by the TA shown on Fig. 2. The load-dependent frequency adaptation scheduler for the computational element is adapted from [20]. The CPU represented by this scheduler operates at frequency $f1$ if the number of events in the input buffer is less then a threshold value and switches to frequency $f2$ otherwise, where $f1 < f2$. This scheduler model was generalized and extended with a check and error state, as described above. The TA of the generalized scheduler is shown in Fig. 8. Basically, the adapted scheduler can change its processing speed when the

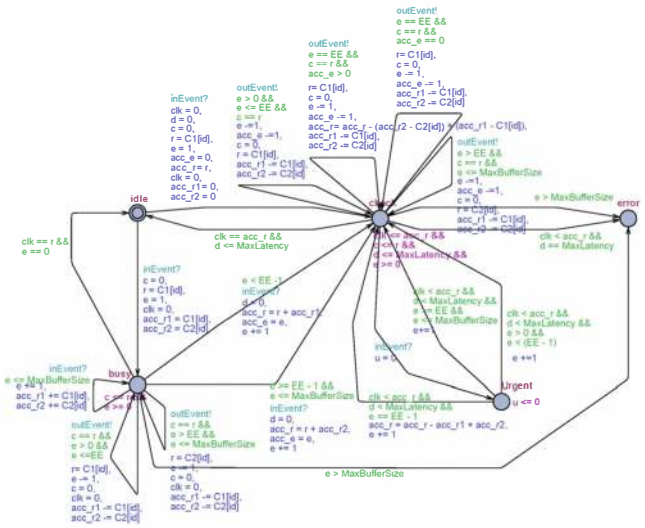


Figure 8: Generalized case-study scheduler

Exp	N_{in}	δ_{in}	*Buffer Size*	*Latency*	N_{obs}	δ_{obs}
1	0 .. 10	0 .. 10	10	20	5	2
2	4	3	0 .. 10	0 .. 10	5	2
3	4	3	10	20	1 .. 10	0..10

Table 1: Experiments values and parameters

number of tasks in the buffer exceeds a certain threshold. Thus, we define two different worst-case execution times for the task: one for when the number of tasks in the buffer is equal or below the threshold ($C1$) and another for when it is above the threshold ($C2$). For the observer we use the UTA observer presented in Fig. 5.

We have a set of three different experiments for this case study. The result of the experiments are regions of parameters expressed as a set of linear constraints in the parameter space. The dimensionality of the space is equal to the number of parameters used. For visualization, we typically limit our study to sets of at most two or three free parameters. The symbolic representation of the constraints, however, does not have this limitation, and can be used as an input for further automatic analysis, such as robustness analysis.

For our experiments we evaluated the system for groups of two parameters at a time. The settings for the simulations are as follows. For all three cases we have fixed the threshold value for switching CPU speed equal to 4. Upon the change in the CPU speed, the task computation time changes from $C1 = 4$, when the number of events in the buffer is below the threshold, to $C2 = 2$, otherwise. The rest of the parameters and values for the experiments are summarized in Table 1.

In experiment 1 we have performed a *Processing Element Analysis* where we have studied the feasibility area for the maximum buffer size (Max Buffer) and the maximum latency (Max Latency) within a range of $[0, \ldots, 10]$ for both parameters. The latency refers to the maximum time a task may spend in the processing state (starting from its insertion in the buffer) until it completes, and corresponds to the relative deadline of the task. The feasibility area of this experiment is presented in Fig. 9 and was obtained in only 18 seconds using 64 sample points in the parameter space. This experiment was executed on an AMD Athlon(tm) 64 X2 Dual Core Processor 5000+ CPU 1GHz machine. The results of this experiment show that the minimum buffer size should be equal to or

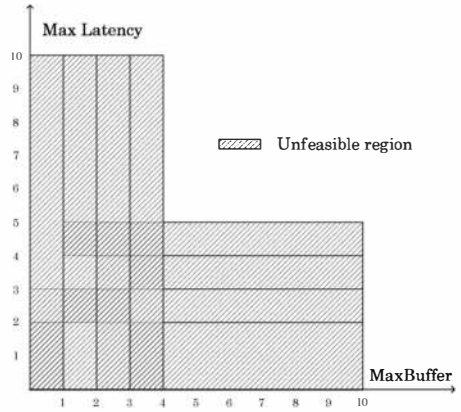


Figure 9: Feasibility region, experiment 1

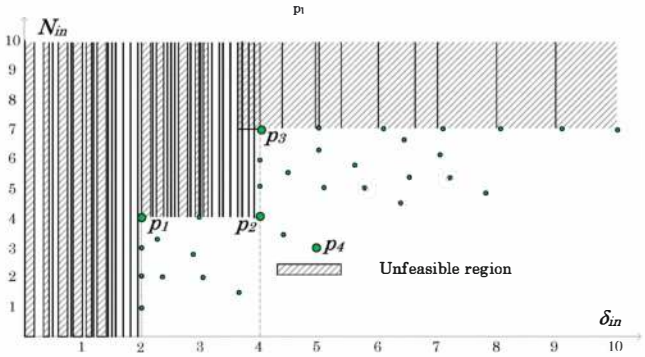


Figure 10: Feasibility region, experiment 2

greater than 4 elements, and that the maximum latency, or deadline, must be at least 5 time units with this system configuration. This kind of analysis is important in the early phases of design when we would like to choose a buffer size that will guarantee that no overflow will occur, without being too conservative. The parametric analysis also highlights the relationships between the values of the parameters. The estimate of the maximum latency values in turn is important for real-time analysis. If it is too large, we can decrease the execution time of the task by choosing a CPU with higher processing speed, therefore, choose the processing element that fits the system requirements.

The graph yields that the optimal point, i. e., the point that corresponds to the best choice of the parameter values, is the one with *buffer size* equal to 4 and *latency* equal to 5. The other points at the border are found automatically by the PFA tool using a combination of Uppaal and NuSMV, and could also be found by theoretical analysis or using a binary search algorithm for one of the parameters while the other is fixed to a certain value. The rest of the points that belong to the feasibility area are still valid points. For a fully deterministic systems, e. g., when we are sure that the worst-case execution time of the tasks is never violated, the best values of the parameters lies at the border of the feasibility graph.

Some of the parameters, however, may be affected by estimation errors, or may fail to provide the absolute worst case to avoid overly large pessimism. For example, in this paper we talk about the processing elements taking into account not only computational processing units (CPUs) but also the communication elements with an arbiter that manages packet transmission. The communication may introduce packet losses that are induced by some external physical factors. Thus, the packets that were lost should be added back to the buffer to be retransmitted. If this is the case, we can not assure that choosing the buffer size equal to 4 and the deadline to 5 will never result in a constraint violation. Choosing a point which is further away from the border of the feasibility area will give us parameters values that make the system more robust.

For experiment 2 we perform a *Backward Analysis*, where N_{in} and δ_{in} are chosen to have values within a bounded range, in this case $[0, \ldots, 10]$ for both of them. The values for the other variables should be chosen such that they do not affect the result in a negative way. For example, if we choose the values of the buffer size equal to 0 we will never find any feasible region for the proposed set of parameters, since the maximum buffer size will be immediately exceeded, which will provoke a violation of the constraints. Taking into account the results of the previous experiment, we set the buffer size to 10 and maximum latency to 20 making sure that they will never be violated. The resulting feasibility area is presented in Fig. 10. In this case, the PFA tool was unable to properly generalize unfeasible points in the vicinity of the border between the feasible and unfeasible region. This is due to both the structure of the model, and the chosen set of parameters. The problem arises because the traces that lead to the error state for unfeasible points become increasingly unique as we move towards the border of the region, while generalization requires that nearby points share the same trace. The generalization of the points eventually degenerates into a line.

Nonetheless, the information provided by the analysis is useful despite the incompleteness of the areas. In particular, we can plot several points on the graph that were found feasible and can help define a safe border between the two regions. Points that fall between two unfeasible areas are considered not safe. In simple cases, a fully symbolic search could be run to complete the areas. The result for experiment 2 that is depicted on Fig. 10 was obtained in 132 minutes and 9 seconds. The experiment was executed on an Intel(R) Pentium(R) 4 CPU 2.80GHz machine.

The found feasibility regions of the *Backward Analysis* are then used to build the RTC-based input curves. We have chosen three points from the border of the feasible region and one far from it and constructed the RTC-based curves using the parameters values from these points. The first three curves have the following parameters: $\delta_1 = 2$, $N_1 = 4$, $\delta_2 = 4$, $N_2 = 4$ and $\delta_3 = 4$, $N_3 = 7$; as for the fourth curve, we have chosen $\delta_4 = 5$ and $N_4 = 3$. The set of curves is presented in Fig. 11. From this graph we can see that the curves built using the parameter values of the point that lies further from the border is located below all the curves built out of the border point parameters values. This curve corresponds to the lower load of the system with input events, which makes the system more robust. The choice among points that are at the border depends on the particular application, since the resulting curves are incomparable and each constitutes a different trade-off between the parameters.

In experiment 3 we have performed the *Forward Analysis*. Here N_{obs} and δ_{obs} are chosen as parameters with a bounded range of $[0, \ldots, 10]$ for both of them. The feasibility area is presented on Fig. 12. For this experiment, similar to experiment 2, we encounter the degeneration of the generalized areas into lines and points while approaching the border between the feasible and unfeasible regions. N_{obs} represents the number of events that can arrive at the same time, and thus has an integer nature. Therefore, we set the step of the grid along N_{obs} axis equal to 1. The result for experiment 3 that is depicted on Fig. 12 was obtained in 7 hours and 56 minutes. The experiment was executed on an Intel(R) Pentium(R) 4 CPU 2.80GHz machine.

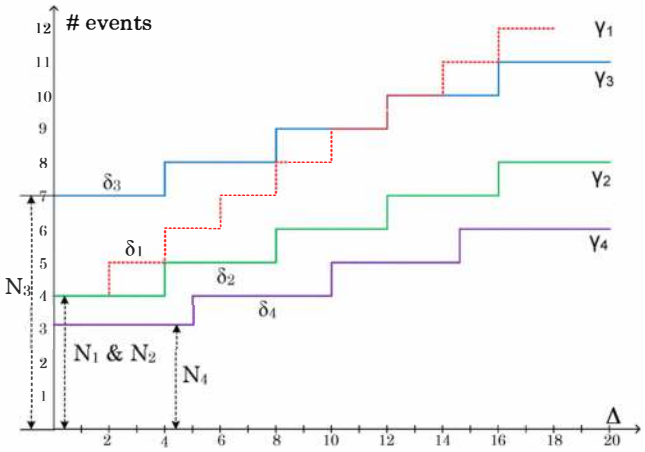


Figure 11: Set of input curves built by using values for N_{in} and δ_{in} from the feasibility region.

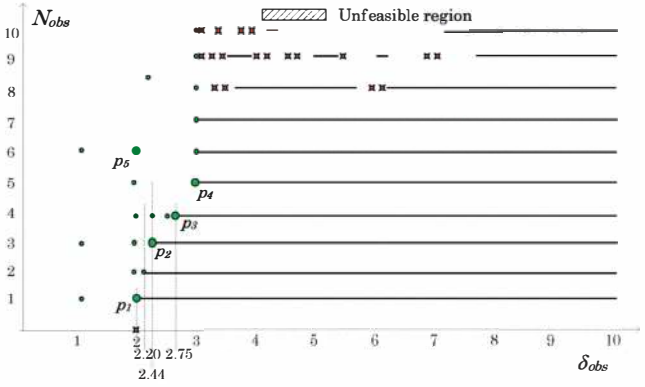


Figure 12: Feasibility region, experiment 3

For this experiment we have chosen four points from the border of the found feasible region and one far from it and constructed the RTC-based output curves using the feasible parameters values from this points. The chosen point are summarized in Table 2. There (b) next to the point number stands for a *border* point and (r) means the point taken from the feasibility area away from the border. The output curves built using the parameters values from the feasibility area are depicted in Fig. 13.

From this graph we can see that the curve built from the parameters values of the point that lies further from the border is located above all the curves built out of the border point values. This curve corresponds to the higher expected output rate of the events processed by the system. In this case, unpredictable events due to uncertainties in the estimation as discussed above are less important, as they tend to *decrease* the output rate of the system. Thus, the "real" curve may actually lie *below* the one built using the data points of the resulting area. This is an important observation when studying a system composed of a number of processing elements

Point	**1 (b)**	**2 (b)**	**3 (b)**	**4 (b)**	**5(r)**
N_{obs}	1	2	4	5	6
δ_{obs}	2	2.44	2.75	3	2

Table 2: Chosen points

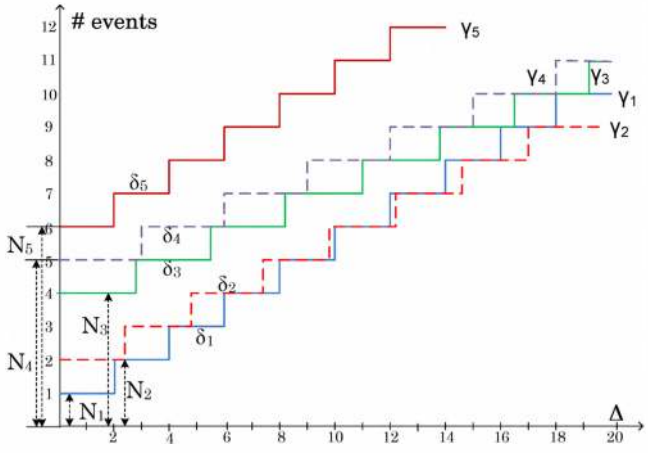


Figure 13: Set of input curves built by using values for N_{obs} and δ_{obs} from the feasibility region.

with limited resources when even a small reduction of the input rate can play en essential role in system robustness.

6. CONCLUSION

In this paper we presented a hybrid design and analysis methodology for distributed real-time systems. The proposed approach integrates Modular Performance Analysis (MPA-RTC) with Parametric Feasibility Analysis (PFA). It uses a simplified representation of arrival curves to interface heterogeneous modeling components. More specifically, the method automatically converts arrival curves as used by MPA-RTC to Timed Automata models, and uses these models to trigger a state-based and parameterized model of a processing or communication component. In a similar fashion, the output of the component is characterized by appropriate observer automata and automatically converted to arrival curves. The novelty of our approach consists in deriving feasible regions for various component parameters such as tolerable data rates or bursts for the input or the output of the component, and tolerable fill levels for its activation buffer. Our results extend previous analysis methods which permitted the evaluation of single design points only. For automatically deriving the region of feasible parameters for a component, we implemented a dedicated tool-chain which employs Uppaal and NuSMV. The resulting tool permits us to efficiently explore large design spaces and hence directly supports the design of complex distributed systems.

Acknowledgments

This work was supported in part by the EU project COMBEST n. 215543 and the EU NoE ArtistDesign n. 214373. We thank Prof. Palopoli and Dr. Cimatti for useful discussions.

7. REFERENCES

[1] R. Alur and D. L. Dill. Automata For Modeling Real-Time Systems. In M. Paterson, editor, *ICALP'90*, LNCS 443, pages 322–335. Springer, 1990.

[2] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *STOC'93*, pages 592–601, New York, NY, USA, 1993. ACM.

[3] E. André, T. Chatain, L. Fribourg, and E. Encrenaz. An Inverse Method for Parametric Timed Automata. *Electron. Notes Theor. Comput. Sci.*, 223:29–46, December 2008.

[4] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *Proc. of the Fourth IEEE Intl. Conference on Software Engineering and Formal Methods*, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

[5] G. Behrmann, R. David, and K. G. Larsen. A tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems: 4th Intl. School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, pages 200–236. Springer, 2004.

[6] E. Bini and G. C. Buttazzo. Schedulability Analysis of Periodic Fixed Priority Systems. *IEEE Trans. Comput.*, 53(11):1462–1473, 2004.

[7] M. Büker, A. Metzner, and I. Stierand. Testing real-time task networks with functional extensions using model-checking. In *Proc. of the 14th IEEE int. conference on Emerging technologies & factory automation*, ETFA'09, pages 564–573, Piscataway, NJ, USA, 2009. IEEE Press.

[8] G. C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[9] S. Chakraborty, L. T. X. Phan, and P. S. Thiagarajan. Event Count Automata: A State-Based Model for Stream Processing Systems. In *RTSS'05*, pages 87–98, 2005.

[10] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A New Symbolic Model Verifier. In N. Halbwachs and D. Peled, editors, *CAV'99*, volume 1633 of *Lecture Notes in Computer Science*, pages 495–499. Springer, 1999.

[11] A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic Computation of Schedulability Regions Using Parametric Timed Automata. *RTSS'08*, 0:80–89, 2008.

[12] A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic Computation of Schedulability Regions Using Parametric Timed Automata. In *RTSS'08*, Nov. 30 - Dec. 3 2008.

[13] H. Dierks, A. Metzner, and I. Stierand. Efficient Model-Checking for Real-Time Task Networks. In *Proc. of the 2009 Intl. Conference on Embedded Software and Systems*, pages 11–18, Washington, DC, USA, 2009.

[14] L. Doyen, T. A. Henzinger, A. Legay, and D. Nickovic. Robustness of Sequential Circuits. In *ACSD'10*, pages 77–84, 2010.

[15] M. González Harbour, J. J. Gutiérrez García, J. C. Palencia Gutiérrez, and J. M. Drake Moyano. MAST: Modeling and Analysis Suite for Real Time Applications. In *Proc. of 13th Euromicro Conference on Real-Time Systems*, pages 125–134. IEEE Computer Society, 2001.

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

[17] T. Hune, J. Romijn, M. Stoelinga, and F. W. Vaandrager. Linear Parametric Model Checking of Timed Automata. In *TACAS'01*, pages 189–203, London, UK, 2001. Springer-Verlag.

[18] K. Keutzer, S. Malik, A. R. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System Level Design: Orthogonolization of Concerns and Platform-Based Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12), December 2000.

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

[20] 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: Proc. of the 7th ACM int. conference on Embedded software*, pages 107–116, New York, NY, USA, 2009. ACM.

[21] 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.

[22] C. Lu, J. A. Stankovic, T. F. Abdelzaher, G. Tao, S. H. Son, and M. Marley. Performance Specifications and Metrics for Adaptive Real-Time Systems. In *RTSS'00*. IEEE Computer Society, 2000.

[23] M. Lv, G. Nan, W. Yi, and G. Yu. Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software. In *RTSS'10*, pages 339–349, Washington, DC, USA, 2010. IEEE Computer Society.

[24] C. Norström, A. Wall, and W. Yi. Timed Automata as Task Models for Event-Driven Systems. In *RTCSA '99*, page 182, Washington, DC, USA, 1999. IEEE Computer Society.

[25] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, R. Henia, R. Racu, R. Ernst, and M. G. Harbour. Influence of different system abstractions on the performance analysis of distributed real-time systems. In *Proc. of the 7th ACM & IEEE int. conference on Embedded software*, EMSOFT '07, pages 193–202, New York, NY, USA, 2007. ACM.

[26] L. T. X. Phan, S. Chakraborty, P. S. Thiagarajan, and L. Thiele. Composing Functional and State-Based Performance Models for Analyzing Heterogeneous Real-Time Systems. In *RTSS'07*, pages 343–352, Washington, DC, USA, 2007. IEEE Computer Society.

[27] A. L. Sangiovanni-Vincentelli. Quo Vadis, SLD? Reasoning About the Trends and Challenges of System Level Design. *Proc. of the IEEE*, 95(3):467–506, March 2007.

[28] K. Sen, M. Viswanathan, and G. Agha. Statistical Model Checking of Black-Box Probabilistic Systems. In *CAV'04*, LNCS 3114, pages 202–215. Springer, 2004.

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

[30] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System Architecture Evaluation Using Modular Performance Analysis: A Case Study. *STTT*, 8(6):649–667, 2006.

[31] E. Wandeler, L. Thiele, M. Verhoef, and P. Lieverse. System Architecture Evaluation Using Modular Performance Analysis: A Case Study. *STTT*, 8(6):649–667, 2006.

[32] S. Yovine. Model Checking Timed Automata. In G. Rozenberg and F. W. Vaandrager, editors, *European Educational Forum: School on Embedded Systems*, volume 1494 of *Lecture Notes in Computer Science*, pages 114–152. Springer, 1996.

[33] D. Zhang and R. Cleaveland. Fast On-the-Fly Parametric Real-Time Model Checking. In *RTSS'05*, pages 157–166, Washington, DC, USA, 2005. IEEE Computer Society.