Composing Functional and State-based Performance Models for Analyzing Heterogeneous Real-Time Systems

Linh T.X. Phan[1] Samarjit Chakraborty[1] P S Thiagarajan[1] Lothar Thiele[2]
[1]Department of Computer Science, National University of Singapore
[2] Computer Engineering and Networks Laboratory, ETH Zürich
E-mail:{phanthix, samarjit, thiagu}@comp.nus.edu.sg thiele@tik.ee.ethz.ch

Abstract

We present a performance analysis technique for distributed real-time systems in a setting where certain components are modeled in a purely functional manner, while the remaining components require additional modeling of state information. The functional models can be efficiently analyzed but have restricted expressiveness. On the other hand, state-based models are more expressive and offer a richer set of analyzable properties but are computationally more expensive to analyze. We show that by appropriately composing these two classes of models it is possible to leverage on their respective advantages. To this end, we propose an interface between components that are modeled using Real-Time Calculus [Chakraborty, Künzli and Thiele, DATE 2003] and those that are modeled using Event Count Automata [Chakraborty, Phan and Thiagarajan, RTSS 2005]. The resulting modeling technique is as expressive as Event Count Automata, but is amenable to more efficient analysis. We illustrate these advantages using a number of examples and a detailed case study.

1 Introduction

Modern real-time systems are increasingly becoming distributed and heterogeneous. They consist of multiple processing elements, hardware accelerators, memory units and communication subsystems. Each of these components run tasks that have very diverse execution/communication requirements and timing constraints. They support different scheduling and resource arbitration policies and often interact with the physical world (via sensors and as actuators). As a result, timing and performance analysis of such systems is a challenging problem.

There has been previous efforts to use multiple languages and formalisms for *specifying* such heterogeneous real-time and embedded systems (e.g. see [20]). However, the issue of composing different *analysis techniques* has not been sufficiently explored so far. Recently, there has been some progress on this front. For example, [10] outlined a scheme where some of the components of a system were analyzed using purely simulation-oriented techniques (based on SystemC models), while the remaining components were modeled and analyzed using a set of algebraic equations based on the formalism called Real-Time Calculus (RTC) [4]. The main contribution of [10] was an interface between these two classes of components at the analysis level. While the RTC-based analysis represented the arrival patterns of data and event streams in an abstract fashion, the SystemC-based models required concrete input traces to trigger the simulation. To compose the two analysis techniques, one must be able to inter-convert the two models. Deriving abstract representations of arrival patterns from concrete traces (the SystemC-to-RTC conversion) is relatively straightforward (e.g. by using standard event models such *periodic with jitter*). However, a transformation in the other direction, i.e. deriving a small set of representative traces from an abstract event model (the RTC-to-SystenC conversion) is a challenging problem, the details of which may be found in [10]. A similar approach was presented in [16], where certain system components were modeled and analyzed using SDF graphs [9, 11] with the remaining ones being subjected to classical real-time schedulability analysis techniques.

Contributions of this paper: Following this line of work, in this paper we propose an interfacing technique to compose purely functional analysis schemes with state-based modeling and analysis methods. While the former are based on solving a set of algebraic equations, the latter rely on reachability analysis on the state space of an automata-theoretic model [7]. The systems we analyze are made up of a number of different components that communicate via unidirectional data or event streams which might be buffered. Each of these components is either a processing or a communication element. They support concurrent tasks which might have variable execution demands and process one or more incoming event/data streams.

Suppose a specification of such a system is given in terms of its architecture, the tasks mapped onto each of the components, their execution demands and the scheduling policies used. Further, suppose the timing properties of each

incoming stream have also been specified. The goal then is to compute performance metrics such as the maximum end-to-end delay suffered by the stream, minimum buffer requirements and the utilization of the different components. The Real-Time Calculus (RTC) framework was proposed earlier to analyze such setups in [4], which was further extended in subsequent papers (e.g. see [5, 18]). The key feature of this framework is that it allows a very general modeling of event streams and resource availability beyond the classical event models such as periodic, sporadic, periodic with jitter, etc. Another important feature — which is particularly relevant for performance analysis — is that, rather than recording the precise arrival times of events, RTC uses a *count-based abstraction* which specifies upper and lower bounds on the possible number of events that can arrive (or can get processed) within any pre-specified time interval length. This count-based abstraction captures in a very natural way, bursty arrival patterns of events/data, variable execution demands and irregular resource availability.

RTC-based models use upper and lower bounds on the number of events that arrive at a component (processing or communication element) to get processed within any time interval of a specified length. Such bounds are represented as *functions* and are used to estimate the workload to be supported by the component. Similar functions are also used to represent the *service* offered by the component. While such functional modeling allows for efficient analysis, the main drawback of this approach is that it can not model any state-based information. For example, a common feature such as "the processor stalls when the output buffer is full" cannot be modeled since the service offered by the processor in this case depends on the *state* (fill level) of the buffer.

To get around this problem, but at the same time retain the count-based abstraction, in [6] we proposed an automata-theoretic model called Event-Count Automata (ECA). ECAs are syntactically similar to timed-automata [1] but have very different semantics. ECA models represent families of arrival patterns of data/event streams as an automaton whose language is the set of integer sequences representing all possible permissible arrival traces. The service offered by a component is also captured in a similar fashion using an automaton. Performance/timing analysis questions related to this model can be formulated as standard model checking problems [3, 8]. This model easily and transparently overcomes the limitations of the RTC framework in terms of expressiveness and the set of analyzable properties. However, as is the case with most state-based modeling techniques, it suffers from the crippling state explosion problem.

In this paper we propose to combine the advantages of RTC and ECA-based modeling. The idea is to subject the components that do not require state-based information to RTC-based modeling analysis and the remaining to ECA-based modeling and analysis. The main contribution of this paper is a combined system-level analysis method derived by creating *interfaces* between these two types of models.

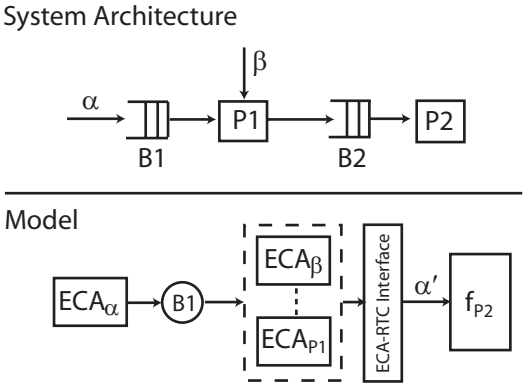


Figure 1. Overview of the proposed scheme.

To bring out the basic ideas underlying our method, consider the architecture shown in Figure 1. Suppose the processing element $P1$ has been modeled as an ECA, perhaps due to the fact the number of items it processes in unit time depends on the fill level of its output buffer $B2$. Suppose further, the arrival pattern of the data stream being buffered in $B1$ as well as the pattern of service available at the processing element for this task have been modeled in the RTC framework as the arrival curve α and the service curve β respectively (definitions of which are outlined later in the paper). Finally, assume that items arriving in $B2$ are further processed by the processor $P2$ which has been modeled as an RTC component whose semantics is captured by the function f_{P2}.

We first use an RTC-ECA interface to convert α (β) into ECA_α (ECA_β). We then analyze the performance of the *small* network of ECAs consisting of ECA_{P1}, ECA_α and ECA_β. We note that there is no buffering between ECA_β and ECA_{P1} and in fact, these two automata will be collapsed into a single ECA in the performance model. Next we use an ECA-RTC interface to convert the event stream being buffered at $B2$ by the ECA network to produce the arrival curve α'. This is then combined with f_{P2} to perform an RTC-level analysis at the next stage.

In this way, we switch between ECA-based and RTC-based modeling and analysis to obtain a system-level performance analysis method. In a heterogeneous distributed system where there is a good mix of RTC and ECA components, our method will avoid the state explosion that would be caused by converting the whole system into a large network of ECAs. At the same time, it would avoid the accuracy penalty incurred if the system were to be modeled as large network of RTC components. It could also be the case that the pure RTC alternative is not available due to the fact that the semantics of some of the processing elements are state-dependent or because the performance property we are validating cannot be formulated in the RTC framework.

Before concluding this section, we would like to point out that our approach is orthogonal to the previous efforts

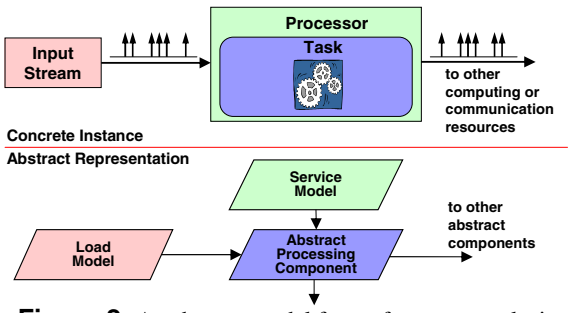


Figure 2. An abstract model for performance analysis.

on full-system performance analysis, for example, using holistic schedulability analysis [2, 12, 13, 14, 17]. Although these efforts were also directed towards analyzing systems with multiple resources and scheduling policies, the underlying analysis techniques used for the different components were similar and were mostly based on classical event models. Our goal, on the other hand, is to combine two fundamentally different performance models for system-level analysis.

Organization of this paper: The rest of this paper is organized as follows. In the next section, we briefly outline the RTC framework, following which we describe ECA-based modeling and analysis. Sections 3 and 4 contain the core technical results of this paper, i.e. the composition of RTC and ECA models and the construction of the interfaces between them. Finally, the advantages of this combined method are brought out in Section 5 with the help of some experimental results. The paper concludes by outlining some directions for future work.

2 Performance Models

This section outlines the Real-Time Calculus (RTC) and Event Count Automata (ECA) based modeling and analysis. Figure 2 gives a high-level overview of both the schemes. It shows how a processing element processing an event/data stream is modeled. At the model level, *load models* describe the timing characteristics of the event streams. Since the analysis aims at providing results for a *family* of possible input streams instead of a (single) concrete event stream, this information must be represented in an abstract way (e.g. using a *periodic* or *periodic with burst* or an even general event model). *Service models* provide information about the resources available within a system (e.g. their capacities like processor or bus bandwidth). Lastly, *processing models* represent the processing semantics (e.g. the scheduling and arbitration policies) that is used to execute the application tasks. A *performance model* of a whole system is obtained by connecting the load models and service models to the corresponding models of processing components. The main distinction between RTC and ECA-based models is that while the former uses functions to represent load, service and processing models, the later relies on explicit state-based (automata-theoretic) modeling.

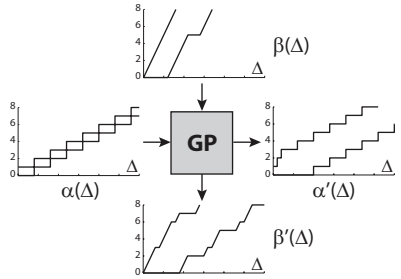


Figure 3. An abstract performance model in RTC.

2.1 Real-Time Calculus

Here, an event stream is described using upper and lower *arrival curves*, $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$, which provide upper and lower bounds on the number of events that are seen on the event stream within any time interval of length Δ. In particular, there are at most $\alpha^u(\Delta)$ and at least $\alpha^l(\Delta)$ events within the time interval $[t, t+\Delta)$ for all time instances t. If $R(t)$ denotes the number of events that arrive at a resource during the time interval $[0, t]$ then such a stream is bounded by $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$, where $\alpha^l(\Delta) \leq R(t+\Delta) - R(t) \leq \alpha^u(\Delta)$ for all $t \geq 0$ and all $\Delta \geq 0$.

Analogously, a resource is characterized using upper and lower service curves, $\beta^u(\Delta)$ and $\beta^l(\Delta)$, which provide upper and lower bounds on the available resource in any time interval of length Δ. The unit of resource depends on the kind of shared resource, for example processing cycles (computation) or bytes (communication), which can be transformed to the same unit as the arrival curves (number of events). The formulation presented in this paper assumes the service curves and arrival curves are being both expressed in terms of *number of events*. A more technical definition of arrival and service curves is given in the next section. The processing semantics of a processing/communication component (e.g. a processor or a bus) can be captured using an RTC component as shown on Figure 3. Here, an arrival curve $\alpha(\Delta)$ enters the component and is processed using the service curve $\beta(\Delta)$. The output is an arrival curve $\alpha'(\Delta)$ (which denotes the arrival pattern of the *processed events*) and the *remaining resource* is expressed as the service curve $\beta'(\Delta)$. Internally, the RTC component is specified by a set of functions, that relate the incoming arrival and service curves to the outgoing arrival and service curves. These functions are dependent on the processing semantics of the component (see e.g. [4]). Similar representations exist for resource scheduling disciplines such as EDF, TDMA, and GPS, when multiple streams or tasks are being processed by the component. The outgoing arrival curves might serve as input to another component, denoting further processing of the event stream. Similarly, the outgoing service curve represents the service that is available to other tasks running on the same component. Various performance properties can be computed analytically using this performance model such as end-to-end delays and buffer requirements. These methods have been implemented as a toolbox which is based on Matlab and Java (see [19]).

2.2 Event Count Automata

An Event Count Automaton (ECA) too describes arrival patterns of event streams (as well as arbitrary service patterns). It may be viewed as a device which records the arrival pattern of a stream and decides to accept or reject the stream. The accepted streams constitute the family of arrival patterns specified by the ECA. An ECA is an ordinary automaton augmented with *count variables*. These are integer variables that are used to keep track of the number of events that are seen on an event stream (or the number of events that can be processed by the available resource) over certain time intervals. In particular, an ECA consists of a set of states (modes), a set of count variables, and a transition relation that specifies possible transitions from one state to another and a guard associated with each transition which must be satisfied if the transition is to be taken. Each state has a rate vector $[l, u]$, which specifies that at least l and at most u events arrive in every unit of time when the system is in the state. The guard associated to a transition is a logical conjunction of formulas of the form $x \leq C$ or $x \geq C$, where x is a count variable of the ECA and C is a constant.

ECAs take a transition at discrete time instances. During every unit interval, an ECA stays at its current state and counts the number of events that arrive from the event stream associated with it. At the end of the current unit interval, it increases all its count variables by the number of events that have arrived in this interval. It then checks the guards associated with the out-going transitions from the current state. If one or more of the guards evaluates to true, the ECA will take -instantaneously and non-deterministically- one of the enabled transitions to the next state. If no transition is enabled then, it will stay put in the current state. During this move, it may reset some of its count variables (as specified in the transition) to zero. The purpose of the resets is to adjust the time interval over which the count variable keeps the information. Hence, at any instant, the current value of the count variable x denotes the number of events that have arrived since x was last reset. For modeling convenience, a count variable may also be reset to a non-zero value determined by a linear expression over the count variables.

Figure 4 shows an ECA that models an event stream that has at most 7 events arriving in every unit time interval when in mode S_0, at least 1 event and at most 6 events arrive per unit interval when in mode S_1 and exactly 5 events per unit interval arrive when in mode S_2. The behavior of an ECA that describes a resource is similar, except that the ECA counts the number of resource units (e.g. processor/bus cycles) available instead of the events seen on a stream. A run of the automaton will consist of a sequence of the form $s_0 n_1 s_1 n_2 \ldots n_k s_k$ where s_o is an initial state and n_i is the number of events that have arrived during the interval $[i-1, i)$. If s_k is an accepting state, then we say

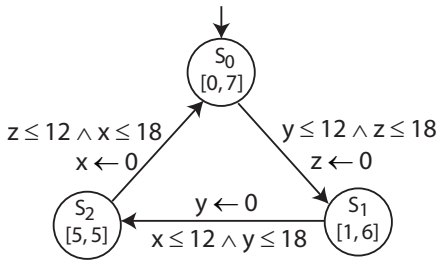


Figure 4. An example ECA.

that the ECA "accepts" the event stream $n_1 n_2 \ldots n_k$. It is in this sense, that an ECA defines a family of event streams. More details about the syntax and semantics of ECAs may be found in [6].

For system-level modeling, an ECA is augmented with a set of input and output buffers, and an update function using which items are removed from the input buffers and added to the output buffers during the execution of a transition. Various scheduling policies can be described using such update functions. The performance model consists of a network of ECAs (augmented with update functions) communicating with each other via shared buffers. Figure 5 depicts the ECA model of a system consisting of two input streams processed by a processor using some scheduling policy. The arrival patterns of the two streams are specified by the ECA_1 and ECA_2 while the semantics of P, the processing element, is captured by ECA_P. Such ECA networks can be analyzed by converting them into Petri nets or might serve as input programs for any of the verification tools that perform efficient state space explorations to do model checking.

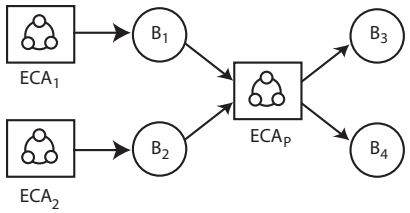


Figure 5. An abstract performance model using ECAs.

3 Composing RTC and ECA Models

In this section, we describe our compositional method that combines RTC-based and ECA-based modeling and analysis into a single framework. The abstract model for each component in the system can be represented using either the RTC or the ECA formalism, depending on the processing semantics of the computing/communication resource in the component. The performance model of the whole system is obtained by connecting the components together to reflect the flow of data and resources, with interfaces between the RTC components and ECA components. The construction of the performance model for a given system consists of the following steps: (i) Identify all components of the system and based on the processing semantics of the components, select between RTC and ECA mod-

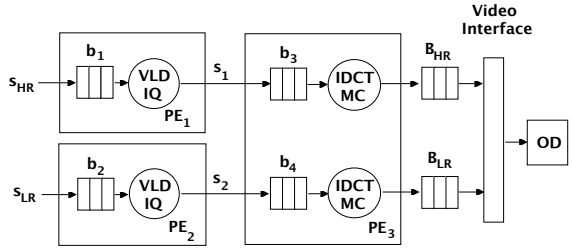


Figure 6. PiP application decoding two MPEG-2 streams.

els. (ii) Describe the components using the selected models. The construction of the component models is the same as outlined in Sections 2.1 and 2.2. (iii) Connect the component models using interfaces into a network that captures the performance aspects of the whole system.

Typically, RTC modeling is used for components in which the processing of the streams depends solely on known information about the event streams and the available computing resource. For example, components that process event streams based on their predefined priorities (e.g. rate monotonic, earliest deadline first) or sharing slots (e.g. TDMA). On the other hand, components in which the processing of the event streams depends on state information of the system or synchronization between the streams are modeled using ECAs. Examples of such components are the ones that implement blocking-read/write at some of the buffers, the ones in which the resource given to the streams are adjusted at run-time based on the fill-levels of the buffers (multi-level service) or ones that contain AND-task activation (synchronization between multiple streams).

RTC and ECA components are connected through the RTC-ECA interfaces. An RTC-ECA interface converts RTC models into ECA models and ECA-RTC interfaces work the other way round. The algorithms for building these interfaces are explained in the next section.

An illustrative example: We now present an example to illustrate typical hardware/software architectures that can be modeled and analyzed using our approach. Figure 6 shows an architecture consisting of three processing elements PE_1, PE_2 and PE_3 onto which an MPEG-2 decoder has been partitioned and mapped. The architecture implements a picture-in-picture (PiP) application in which two concurrent video streams are being decoded and displayed on the same output device. The first and the second processing elements implement the variable length decoding (VLD) and inverse quantization (IQ) tasks of the decoder, each of which processes a different video stream. PE_3 implements the inverse discrete cosine transform (IDCT) and the motion compensation (MC) tasks. It processes both the output streams from PE_1 and PE_2 using a Fixed Priority scheduling algorithm.

The first stream (s_{HR} in Figure 6) is associated with a higher frame resolution and generates a higher workload on PE_3, compared to the lower resolution video s_{LR}. The compressed bitstreams arrive over a network and enter the

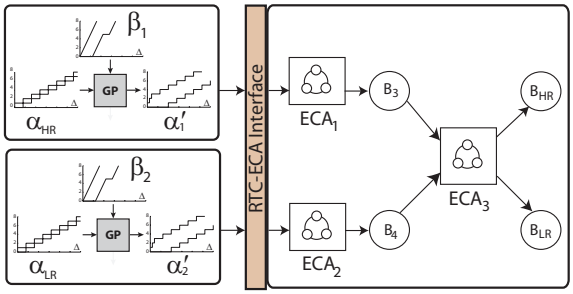


Figure 7. Modeling the architecture in Figure 6.

system after some initial processing at the network interface. They are processed by PE_1 and PE_2 and their outputs, which contain partially decoded macroblocks, are fed as input to PE_3. The fully decoded video streams after being processed by PE_3 are written into two playout buffers B_{HR} and B_{LR} associated with the high and low resolution streams respectively. These buffers are finally read by the display device at pre-specified constant frame rates r_{HR} and r_{LR}. Here, blocking-write is implemented at the play-out buffer associated with s_{HR}. In particular, PE_3 is allowed to write into buffer B_{HR} only if the buffer is not full. In this system, we are interested in answering questions such as (i) What is the maximum fill-level of the buffer b_4? (ii) What is the maximum end-to-end delay of the stream s_{LR}?

As can be seen from the architecture, the system consists of three components, corresponding to PE_1, PE_2 and PE_3. Since the processing of the input stream s_{HR} on PE_1 is done in a greedy fashion and does not depend on any state information, we use RTC to describe the component. Similarly, RTC is used to describe the component corresponding to PE_2. The processing on PE_3, however, depends on the internal state of the output buffers, which cannot be modeled using RTC. Therefore, we use ECA to model this component.

Based on the above observations, the two input streams are modeled using arrival curves α_{HR} and α_{LR}. The two resulting arrival curves α'_1 and α'_2 are then transformed into ECA_1 and ECA_2, respectively, using the RTC-ECA interface. Figure 7 sketches the performance model of the system. The maximum backlog at b_4 is computed using ECA-based analysis on the ECA component. The maximum end-to-end delay of s_{LR} is the summation of the end-to-end delay of α_{LR} on PE_2 component and the end-to-end delay of ECA_2 on the ECA component. Clearly, various other analysis questions can also be answered based on this same performance model. In the next section, we present algorithms for building the above-mentioned interfaces between RTC and ECA components.

4 RTC-ECA Interfacing

This section presents the construction of the interfaces between RTC and ECA components. In particular, we will show how to construct: (i) from an arrival (service) curve, an ECA which models the same set of arrival (service) pat-

terns described by the curve. This is done in Section 4.1. (ii) from an ECA component, an arrival (service) curve which models all arrival (service) patterns of the output stream (remaining service) of the ECA component. This is done in Section 4.2.

4.1 From RTC to ECA

Our translation relies on a specific representation of arrival/service curves, which we first describe. A discrete arrival and service curve, say $\gamma(\Delta)$, $\Delta \in \mathbb{N}^{\geq 0}$, can be compactly represented as a tuple $v = \{\Sigma_A, \Sigma_P\}$ where $\Sigma_A = \langle (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \rangle$ with $x_i < x_{i+1}$ is a segment sequence describing an initial aperiodic part of the curve $\gamma(\Delta)$ for $0 < \Delta \leq x_n$, and $\Sigma_P = \langle (x_1^p, y_1^p), (x_2^p, y_2^p), \ldots, (x_m^p, y_m^p) \rangle$ with $x_n < x_1^p$ and $x_i^p < x_{i+1}^p$ is a segment sequence describing a periodic part of the curve $\gamma(\Delta)$ that starts after the aperiodic part, i.e. $\Delta > x_n$.

The periodic part of γ is defined by the period $P = x_m^p - x_n$ and the vertical offset $E = y_m^p - y_n$ between two consecutive repetitions of the periodic section defined by Σ_P. The first occurrence of Σ_P starts at (x_n, y_n). The curve $\gamma(\Delta)$ corresponding to v can now be determined as follows:

$$\gamma_v(\Delta) = \begin{cases} 0 & \text{if } \Delta = 0 \\ y_1 & \text{if } 0 < \Delta \leq x_1 \\ y_i & \text{if } x_{i-1} < \Delta \leq x_i \\ y_1^p + kE & \text{if } x_n + kP < \Delta \leq x_1^p + kP \\ y_j^p + kE & \text{if } x_{j-1}^p + kP < \Delta \leq x_j^p + kP \end{cases}$$

for all $1 < i \leq n, 1 < j \leq m$ and $k \geq 0$. Figure 8 sketches the graphical representation of v.

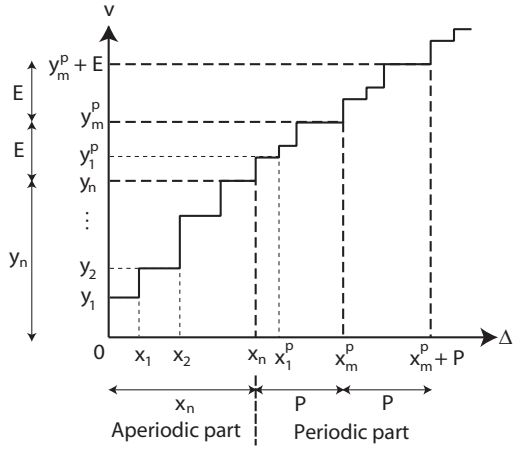


Figure 8. Compact representation of an arrival (service) curve.

The constraints given by an upper (lower) arrival curve represented by v can be interpreted as follows: (i) The aperiodic part Σ_A requires that the number of events that arrive in any interval of length x_i is at most (at least) y_i, for all $1 \leq i \leq n$. (ii) The periodic part Σ_P requires that the number of events that arrive in any interval of length $x_i^p + kP$ is at most (at least) $y_i^p + kE$, for all $1 \leq i \leq m$ and $k \geq 0$. The constraints given by an upper (lower) service curve corresponding to v are exactly the same as that of an upper (lower) arrival curve, except that the conditions are on the number of events that can be processed by the resource.

Given a compact representation v of an arrival (service) curve, we would like to construct an ECA A corresponding to the curve. In particular, an arrival (service) pattern of a given stream (resource) should satisfy the curve v if and only if it is an event stream accepted by A. Since guards on count variables are the means for an ECA to allow/reject a particular arrival pattern, we need to find a suitable set of count variables for A, the necessary and sufficient conditions on these variables under which an arrival pattern is said to satisfy v, i.e. accepted by A.

Constructing an ECA corresponding to an arrival curve: Let $v = \{\Sigma_A, \Sigma_P\}$ be the compact representation of an arrival (service) curve as described above. The ECA that models the event streams constrained by the upper arrival curve v is defined as $A = (S, S_0, CV, T, R, \delta, I)$ where $S = \{s_0, s_1, \ldots, s_N\}$ is the set of states, where $N = x_m^p - 1$, and s_0 is the initial state. $CV = \{f_1, f_2, \ldots, f_{x_n}, g_0, g_1, \ldots, g_{P-1}\}$ is the set of count variables. $I : CV \rightarrow \mathbf{N}$ is the initialization function, and $I(c) = 0$ for all $c \in CV$. $T = \{(s_i, s_{i+1}) \mid 0 \leq i < N\} \cup \{(s_N, s_{x_n})\}$ is the transition system. All transition in T are urgent. δ is the guard function associated with each transition $(s_k, s_{k'})$ in R, which is defined as

$$\delta(s_k, s_{k'}) = \begin{cases} \delta^A, & \text{if } k < x_n \\ \delta^A \wedge \delta^P(k, k'), & \text{otherwise} \end{cases}$$

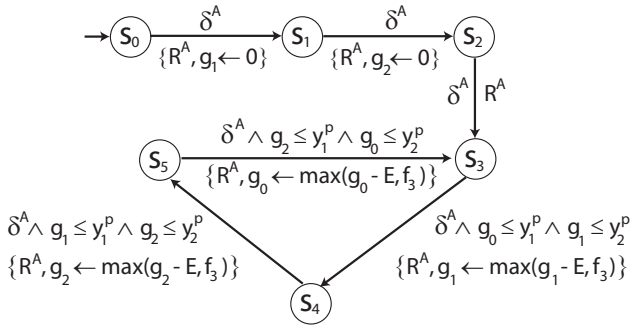
where $\delta^A = \bigwedge(f_{x_j} \leq y_j \mid 1 \leq j \leq n)$ and

$$\delta^P(k, k') = \bigwedge(g_h \leq y_i^p \mid h = (k' - x_i^p) \bmod P, \; 1 \leq i \leq m)$$

R is the reset of the count variables associated with the transitions. The reset of the count variables f_j and g_h along each transition (s_k, s_k') is defined as: (i) $f_j \leftarrow 0$ if $j = 1$, (ii) $f_j \leftarrow f_{j-1}$ if $1 < j \leq n$, (iii) $g_h \leftarrow 0$ if $h = k' < x_n$, (iv) $g_h \leftarrow \max(g_h - E, f_{x_n})$ if $h = k' - x_n \geq 0$, and (v) no reset on g_h, otherwise. The following lemma claims the correctness of the above construction.

Lemma 4.1. Let v be an arrival curve, and A be the corresponding ECA constructed using the procedure described above. Let $\tau = e_1 e_2 e_3 \ldots$ be an event stream, with e_i denoting the number of events that arrive in the time interval $[i-1, i)$. Then, τ satisfies the constraints given by v iff τ is accepted by A.

For the detailed technical basis of the above construction and a proof of this lemma, we refer the reader to `www.comp.nus.edu.sg/~phanthix/papers/rtss07.pdf`.



$$\delta^A = f_1 \le y_1 \wedge f_2 \le y_2 \wedge f_3 \le y_3$$
$$R^A = \{f_1 \leftarrow 0,\ f_2 \leftarrow f_1,\ f_3 \leftarrow f_2\}$$

Figure 9. The ECA model of the arrival curve v.

The ECA that models the upper service curve corresponding to v is exactly the same as the ECA which models an upper arrival curve v, with a different interpretation – the automaton counts the number of events which can be processed by the resource, and hence it models all possible service patterns described by v. As an example, Figure 9 shows the ECA corresponding to an arrival curve $v = \{\langle(1, y_1), (2, y_2), (3, y_3)\rangle, \langle(4, y_1^p), (6, y_2^p)\rangle\}$. Note that $x_i = i, 1 \le i \le 3$, $x_1^p = 4$, $x_2^p = 6$ and $P = 3$.

With a minor modification, we can also construct an ECA that models a given lower arrival (service) curve. In particular, the ECA in this case has a similar structure as the one modeling the upper curve, with the same set of count variables. The guards and resets associated with the transitions are modified as follows: (i) Replace "$\max$" by "$\min$" in all the resets $g_h \leftarrow \max(g_h - E,\ f_{x_n})$. (ii) Replace the "$\le$" sign by "$\ge$" in all the guards. The correctness of this construction follows the same reasoning as the one used for the upper bound case.

4.2 From ECA to RTC

Here, given an ECA component, we would like to compute a safe approximation[1] of the arrival curves (α'^u, α'^l) of an output stream and the remaining service curves (β'^u, β'^l) of a processing element from the component. For simplicity of exposition, the class of curves constructed in this section is a subset of the general curves presented in previous section. In particular, an approximated curve will have an initial aperiodic part and the periodic part will contain just a single entry (a long term period). This class of curves is also sufficient for representing most practical applications.

To reconstruct the arrival curves, we first introduce a specification size W, up to which the aperiodic part is defined. For each window of size Δ, $0 < \Delta \le W$, we compute the maximal and minimal number of output events seen in each window, resulting in $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$, respectively.

[1]ECAs are more expressive than arrival/service curves.

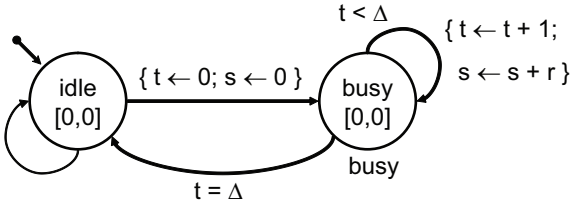


Figure 10. Observer ECA observes the output buffer.

In addition, to capture the long-term behavior (long-term average arrival rate) of the stream, we select a window size W^* which is large enough ($W^* \gg W$) and compute the arrival curves for this window size. The arrival curves are then safely extended for arguments larger than W as follows:

$$\alpha^u(\Delta) = \begin{cases} \alpha^u(W^*) & \text{if } W < \Delta \le W^* \\ \infty & \text{if } \Delta > W^* \end{cases}$$

$$\alpha^l(\Delta) = \begin{cases} \alpha^l(W^*) & \text{if } W < \Delta \le W^* \\ 0 & \text{if } \Delta > W^* \end{cases}$$

These curves can now be refined by guaranteeing super- and subadditivity (i.e. $\alpha^u(\Delta) \le \alpha^u(\Delta - \delta) + \alpha^u(\delta)$ and $\alpha^l(\Delta) \ge \alpha^l(\Delta - \delta) + \alpha^l(\delta)$ for all $0 \le \delta \le \Delta$, $\Delta > 0$). To this end, we perform a stepwise update of the upper and lower arrival curves for all Δ, starting at $\Delta = 2$:

$$\alpha^u(\Delta) := \min_{0 \le \delta \le \lfloor \frac{\Delta}{2} \rfloor} \left\{\alpha^u(\delta) + \alpha^u(\Delta - \delta)\right\} \quad \text{for } \Delta = 2, 3, ...$$

$$\alpha^l(\Delta) := \max_{0 \le \delta \le \lfloor \frac{\Delta}{2} \rfloor} \left\{\alpha^l(\delta) + \alpha^l(\Delta - \delta)\right\} \quad \text{for } \Delta = 2, 3, ...$$

The computation of the remaining service curves is done in the same manner by observing the service left from the component instead of the output stream.

We compute the maximal and minimal number of output events seen in any interval of a specified size Δ using a model checking procedure which we now outline. First we augment the ECA component with an ECA which observes the number of events written into the output buffer at every time unit. As shown in Figure 10, this ECA has two states and two count variables t, s.

The count variable t is used as the timer whereas s is used to count the number of output events from the instant the timer is reset. The variable r on the figure is the number of output events written into the output buffer observed at every unit interval. Initially, the observer ECA is in "idle" state. At the end of every time unit, it non-deterministically chooses to stay put at the current state or move to the "busy" state.

When the ECA moves to the "busy" state, it resets the count variable t (the timer) and s. When the ECA is in "busy" state, it increases s by the value of r (the number of output events observed) at the end of each time unit. When Δ time units have elapsed, the ECA moves back to the "idle" state. Since the rate vector associated with both states is $[0, 0]$, the ECA generates no events and therefore when $t = \Delta$, s contains the number of events of the output stream in an interval of length Δ. Since the interval

is non-deterministically chosen by the ECA, the maximum and minimum values of s are the maximum and minimum number of events in any interval of length Δ, or $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$, respectively.

Using standard verification tools, we can verify if s is at least C_1 and at most C_2, where $C_1 \leq C_2$ are two pre-specified values. This can be achieved by model checking the following LTL specification: $G(s \geq C_1 \wedge s \leq C_2)$. The maximum and minimum values of s are then obtained using binary search on the values of C_1 and C_2.

In addition, there are components which can be modeled using RTC, even though the analysis result thus obtained will be a pessimistic one. For example, in the AND-task activation example that we consider in Section 5.1. Now suppose that we need to compute the arrival curve of the output event stream of an ECA, then we may be able use RTC to model the component and use RTC analysis technique to derive the output arrival curve. The computed curves can be used as the initial value for the curves that we need to compute. The above procedure is then run on these initial bounds to get a tighter result.

Thus, the use of RTC for computing initial approximations is not applicable to all components but only for components which can pessimistically be modeled using RTC. When it can be used, the curve construction process will be speeded up.

5 Experimental Results

In this section, we illustrate how our compositional method can help to improve the performance analysis of practical real-time systems using two examples. The first example concerns a simple artificial system with AND-task activations, whereas the second one looks into the example Picture-in-Picture (PiP) application already described in Section 3. Our analysis aims at answering two important questions: (i) what is the maximum backlog at a buffer in the system? and (ii) what is the end-to-end delay of a stream? Two tools have been used to perform the necessary calculations: the MPA/RTC Toolbox[19] and the Symbolic Analysis Laboratory (SAL)[15]. We used Perl script to describe the interface between RTC and ECA components and to combine the analysis results from the two components.

The freely-available Real-Time Calculus (RTC) Toolbox is a toolbox for system-level performance analysis of distributed real-time and embedded systems. Built on top of Matlab, the toolbox supports most min-plus and max-plus algebra operators and it provides a library of functions for Modular Performance Analysis with Real-Time Calculus (see [4]).

The Symbolic Analysis Laboratory (SAL) is a verification tool for transition systems. The key parts of the tool consist of an intermediate language for describing transition systems and a collection of state-of-the-art LTL model checkers. An ECA component can be easily described in the tool as a composition of different modules, each of which is an ECA. The count variables of an ECA are coded as integer variables and the guards on the ECA transition are written as guarded commands. Finally, the system properties of an ECA component can be defined as temporal logic formulas, which will be verified using one or a combination of the three model checkers: the symbolic model checker based on BDDs, the bounded model checker based on SAT (Boolean Satisfiability Problem) solving, and the infinite bounded model checking based on SMT (Satisfiability Modulo Theories) solving.

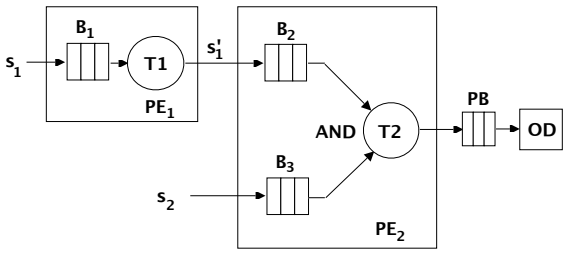


Figure 11. AND-task activation.

5.1 Case study 1: AND-task activation

The system contains two processing elements PE_1 and PE_2, which run two tasks T_1 and T_2. Two input streams, s_1 and s_2 enter the system at PE_1 and PE_2, respectively. s_1 will be first processed by T_1 and the partially processed stream (denoted as s_1') then enters PE_2. At PE_2, the AND-task T_2 is only activated if there are events from both streams s_1' and s_2. When activated, it takes an event from each stream, processes them and produces an output event for the play-out buffer PB. The system is depicted in Figure 11. In this example, both streams are periodic with burst/jitter and the processing elements have a bandwidth of one (i.e. one event can be processed per unit time). Since task T_1 processes a single input stream s_1, it can be easily analyzed using Real-Time Calculus. Task T_2, however, requires synchronization/correlation between the streams, which cannot be naturally modeled using Real-Time Calculus. As a result, we decide to use RTC to model PE_1 and ECA to model PE_2 and connect them using an RTC-ECA interface. This interface converts the arrival curves of the output stream s_1' after being processed by T_1 into an ECA, which will be fed as input to the ECA component.

Figures 12 and 13 sketch the maximum backlog at T_2 and end-to-end delay of s_1 (on the y-axis) with respect to different execution times of T_2 (on the x-axis)for the three methods: (1) RTC for both PEs (2) ECA for both PEs and (3) RTC for PE_1 and ECA for PE_2. Since the ECA-based only method performs an exhaustive search, its analysis result is always tight. The RTC-based only and the combined RTC+ECA methods both compute safe upper bounds of the backlog and delay; however the result obtained using only RTC is much more pessimistic than using RTC+ECA, as shown on the figure.

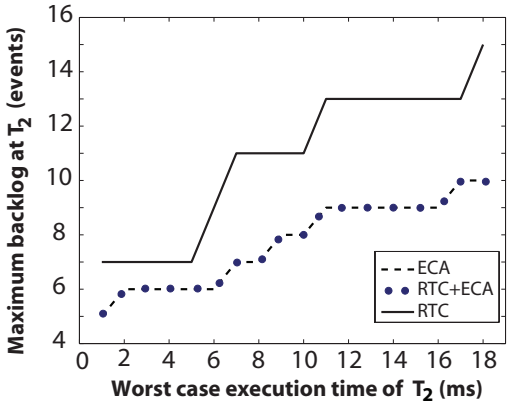


Figure 12. Maximum backlog at T_2.

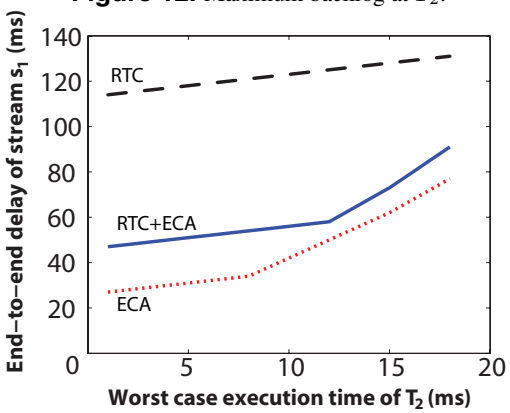


Figure 13. End-to-end delay experienced by stream s_1.

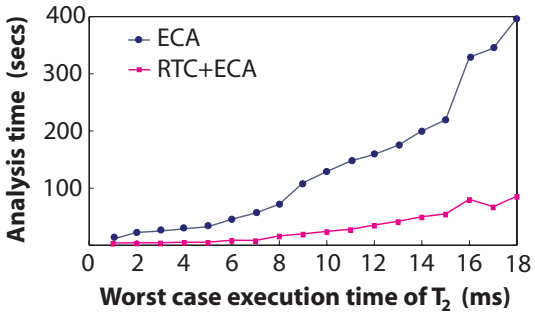


Figure 14. Analysis time for end-to-end delay calculation for s_1.

On the other hand, although the ECA-based analysis produces optimal results, it requires long running times. Using our combined RTC+ECA approach, we are able to reduce the analysis time by an average of 23% for the calculation of the maximum backlog of T_2 and by 78% for the end-to-end delay calculation of s_1, as shown in Figure 14. In other words, while achieving optimal or nearly optimal results, our method is substantially faster than using an ECA-only modeling and analysis. Note that the interface computation is done offline and was not counted into the analysis time. Although this may add additional cost to the running time of our method, we only need to do it once, before the analysis is repeated for several values of system parameters (e.g. different execution times of T_2).

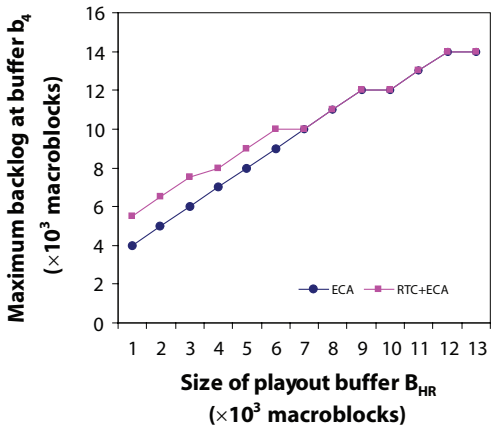


Figure 15. Maximum backlog at the buffer b_4.

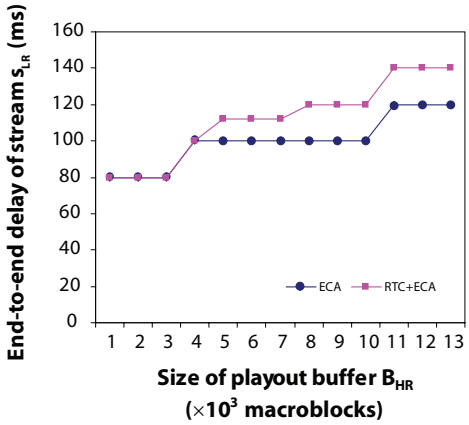


Figure 16. End-to-end delay experienced by the video stream s_{LR}.

5.2 Case study 2: PiP application

In this case study, we look at the PiP application described in Section 3. Our main objectives are to estimate the maximum fill-level of the buffer b_4 and the end-to-end delay of the lower resolution stream s_{LR} (please refer to Figure 6) with different sizes of the playout buffer B_{HR}. In our experiments, both the incoming streams have the same frame resolution of 704×576 pixels (the display device is assumed to perform window resizing for the video stream being displayed at the smaller window). They arrive at the system at a constant bitrate of 8 Mbps and the playout buffers are read at a constant rate of 25 frames/second. The frequencies for PE_1, PE_2 and PE_3 are set as 1.3 GHz, 1.25 GHz and 3 GHz, respectively.

Figures 15 and 16 show the maximum backlog at buffer b_4 and the end-to-end delay of the lower resolution stream for different sizes of the playout buffer PB_{HR}. As shown in these figures, the analysis result obtained using our method is relatively close to the optimal result given by ECA-only analysis. On the other hand, as can been seen in Figures 17 and 18, our method provides a substantial improvement in terms of analysis time. Note that the comparison with RTC-only analysis is omitted as the system cannot be modeled using this approach.

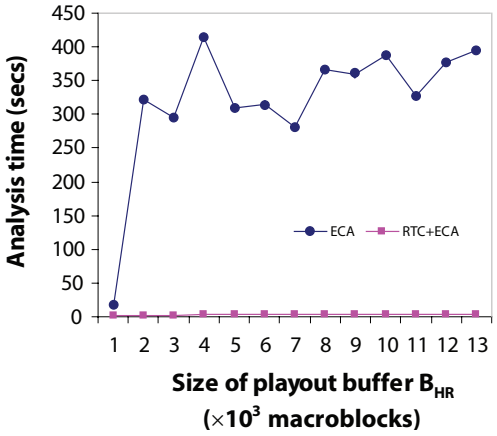


Figure 17. Analysis time for backlog calculation at b_4.

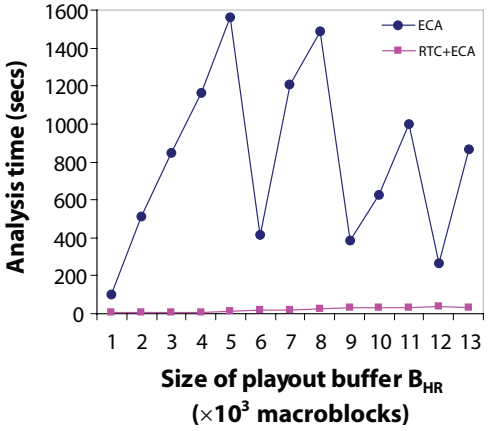


Figure 18. Analysis time for end-to-end delay calculation for s_{LR}.

In summary, we have shown through two examples that by using our compositional approach, we are able to model systems which cannot be modeled using an RTC-only framework, or which suffer from substantially long analysis times when modeled using ECAs only. Our proposed approach leads to acceptable accuracy (in terms of performance estimation) and at the same time has reasonable run times.

6 Concluding Remarks

In this paper we proposed an interfacing technique for composing functional and state-based performance models for distributed real-time systems. Given the growing complexity and heterogeneity of real-time and embedded systems, we believe that composing such fundamentally different modeling and analysis techniques will increasingly become important. As already mentioned, our current implementation relies on a number of independently-developed tools – such as the RTC Toolbox (developed in Matlab and Java) and SAL – which have been hooked together using Perl script. We plan to further develop this tool-chain, and at the same time add user-interfaces to aid in the specification of large-scale real-time systems. We also plan to provide hooks to such a tool-chain that may be used to connect other related tools, such as those for architecture design space exploration.

References

[1] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183 – 235, 1994.

[2] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-vincentelli. Scheduling for embedded real-time systems. *IEEE Design & Test of Computers*, 15(1):71–82, 1998.

[3] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: a model-checking tool for real-time systems. In *CAV*, 1998.

[4] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, 2003.

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

[6] S. Chakraborty, L. T. X. Phan, and P. S. Thiagarajan. Event count automata: A state-based model for stream processing systems. In *RTSS*, 2005.

[7] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[8] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS*, 1998.

[9] A. H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, A. J. M. Moonen, M. Bekooij, B. D. Theelen, and M. R. Mousavi. Throughput analysis of synchronous data flow graphs. In *ACSD*, 2006.

[10] S. Künzli, F. Poletti, L. Benini, and L. Thiele. Combining simulation and formal methods for system-level performance analysis. In *DATE*, 2006.

[11] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

[12] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *CODES*, 2002.

[13] K. Richter and R. Ernst. Event model interfaces for heterogeneous system analysis. In *DATE*, 2002.

[14] K. Richter, M. Jersak, and R. Ernst. A formal approach to MpSoC performance verification. *IEEE Computer*, 36(4):60–67, 2003.

[15] Symbolic analysis laboratory. `http://sal.csl.sri.com`.

[16] S. Schliecker, S. Stein, and R. Ernst. Performance analysis of complex systems by integration of dataflow graphs and compositional performance analysis. In *DATE*, 2007.

[17] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 40(2-3):117–134, 1994.

[18] E. Wandeler, A. Maxiaguine, and L. Thiele. Quantitative characterization of event streams in analysis of hard real-time applications. *Real-Time Systems*, 29(2-3):205–225, 2005.

[19] E. Wandeler and L. Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.

[20] D. Ziegenbein, K. Richter, R. Ernst, L. Thiele, and J. Teich. SPI: a system model for heterogeneously specified embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(4):379 – 389, 2002.