Composing Heterogeneous Components for System-wide Performance Analysis

Simon Perathoner, Kai Lampka, Lothar Thiele
Computer Engineering and Networks Laboratory
ETH Zurich, 8092 Zurich, Switzerland
Email: {firstname.lastname}@tik.ee.ethz.ch

***Abstract*—Component-based validation techniques for parallel and distributed embedded systems should be able to deal with heterogeneous components, interactions, and specification mechanisms. This paper describes various approaches that allow the composition of subsystems with different execution and interaction semantics by combining computational and analytic models. In particular, this work shows how finite state machines, timed automata, and methods from classical real-time scheduling theory can be embedded into MPA (modular performance analysis), a contemporary framework for system-level performance analysis. The result is a powerful tool for compositional performance validation of distributed real-time systems.**

I. Introduction

Modern architectures of embedded systems are increasingly heterogeneous as they consist of communicating and physically distributed computing devices and memory units. In addition, application software is partitioned into communicating and concurrently executing components in order to exploit the computational power of the hierarchically structured execution platforms. Overall, this yields a tremendously increased level of complexity which makes the design and implementation of embedded systems a challenging task, in particular if the correct functionality as well as timing properties need to be guaranteed.

In such a setting only mathematically sound methodologies, commonly denoted as formal analysis methods, are capable of (mathematically) asserting that system designs meet requirements w. r. t. their functionality and timing. The industrial practice of simulating and testing for validating system designs is very often not sufficient as they cover partial behavior of systems only.

Throughout the past decade, different formal methods for asserting system properties have been developed and have found their way into industrial design cycles, see e. g. [1], [2], [3]. For example, state-based modeling of systems has become popular due to the availability of powerful analysis tools as well as the possibility to provide very detailed system models which may even be automatically synthesized from a higher level specification. However, it is well known that state space explosion significantly limits the formal analysis of complex distributed embedded systems.

This work was supported by the EU FP7 projects COMBEST (grant no. 215543), EURETILE (grant no. 247846), and PRO3D (grant no. 249776).

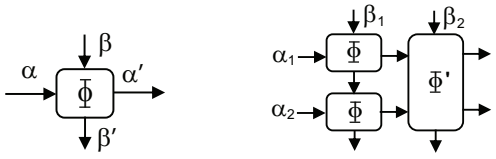
978-3-9810801-7-9/DATE11/© 2011 EDAA

Compositionality appears to be a helpful paradigm, not only for coping with complexity when designing and implementing embedded systems, but also when analyzing them in early design phases. The framework of Modular Performance Analysis (MPA) [4] is a formalism for the performance evaluation of distributed real-time systems. Its methodology origins from Real-Time Calculus (RTC) [5] which uses functions on the time-interval domain for abstractly representing system workloads and the availability of computation and communication resources (cf. Sec. II). Component interaction is modeled by sets of flow-bounding functions rather than concrete streams of events or data packets. The corresponding interaction between abstract components leads to a compositional approach to compute key performance metrics of system-level designs, such as buffer spaces, throughput bounds and end-to-end delays. However, the obtained worst and best case properties such as backlog-sizes and response times, are only tight bounds if the model of computation inherent to RTC is an adequate abstraction of the system under analysis.

The present paper describes various methods to combine computational, state-based modeling and analysis methods with compositional, analytic methods. This way, the expressiveness of state-based modeling formalisms can be leveraged, but the computational complexity is restricted to the intra-component level. The embedding into a inter-component analytic analysis framework enables a scalable, compositional analysis of system-wide properties. It appears that the interfacing of model classes that have different abstraction mechanisms is far from trivial. In the following, we will describe the necessary basic concepts that allow to embed models from classical real-time scheduling theory (Sec. III-A), finite state machines (Sec. III-B), and timed automata (Sec. III-C) into the MPA (modular performance analysis) framework.

II. Modular Performance Analysis

The Modular Performance Analysis (MPA) [4] is a framework for worst-case performance evaluation of distributed embedded systems. MPA is based on Real-Time Calculus (RTC) [5], an analytical formalism derived from Network Calculus [6]. MPA is employed to bound key performance metrics of a system such as worst-case end-to-end delays or buffer requirements. A key feature of MPA is compositionality. It represents each hardware/software unit of a system by an



(a) (b)

Fig. 1. (a) General abstract performance component. (b) Example of system performance model with three abstract components.

individual abstract component which simplifies the analysis of large and heterogeneous systems. In the following, we summarize the basic concepts.

A. Event Stream and Resource Model

In MPA, event streams are abstracted by a pair of arrival curves $\bar{\alpha}(\Delta) = [\bar{\alpha}^u(\Delta), \bar{\alpha}^l(\Delta)]$ which specifiy an upper and a lower bound on the number of event arrivals in *any* time interval of length Δ. Let $R(s,t)$ denote the number of events arriving in the time interval $[s,t)$. Then the relation

$$\bar{\alpha}^l(t-s) \leq R(s,t) \leq \bar{\alpha}^u(t-s) \quad \forall s \leq t \tag{1}$$

holds with $\bar{\alpha}^l(0) = \bar{\alpha}^u(0) = 0$. Any set of event arrival traces can be bounded by an appropriate pair of arrival curves.

The availability of processing or communication resources is abstracted by a pair of service curves $\beta(\Delta) = [\beta^u(\Delta), \beta^l(\Delta)]$ which specify an upper and a lower bound on the available service in any time interval of length Δ. The service is expressed in an appropriate unit such as number of cycles for a processor or number of bytes for a communication device. Let $C(s,t)$ denote the amount of service units in the time interval $[s,t)$. Then the relation

$$\beta^l(t-s) \leq C(s,t) \leq \beta^u(t-s) \quad \forall s \leq t \tag{2}$$

holds with $\beta^l(0) = \beta^u(0) = 0$. Any set of resource availability patterns can be bounded by an appropriate pair of service curves.

B. Workload Model

Events are used to model activation requests for software tasks or hardware components and hence imply a certain resource demand. For expressing the resource demand of an event stream, the corresponding event-based (or packet-based) arrival curves $[\bar{\alpha}^u, \bar{\alpha}^l]$ need to be converted to a resource-based representation $[\alpha^u, \alpha^l]$. In the simplest case in which each event imposes the same workload w on a component, the resource-based arrival curves are simply obtained by appropriately scaling the event-based arrival curves, i.e., $\alpha^u(\Delta) = w \cdot \bar{\alpha}^u(\Delta)$ and $\alpha^l(\Delta) = w \cdot \bar{\alpha}^l(\Delta)$.

In the more general case in which the resource demand varies for different events, the transformation is done by applying the concept of workload curves, see [7]. Workload curves capture the relation between the number of event arrivals and the requested resource demand. Let $W(e)$ denote the total resource demand for e consecutive events of an event stream. Then a pair of workload curves $[\gamma^u(e), \gamma^l(e)]$ satisfies

$$\gamma^l(v-u) \leq W(v) - W(u) \leq \gamma^u(v-u) \quad \forall u \leq v. \tag{3}$$

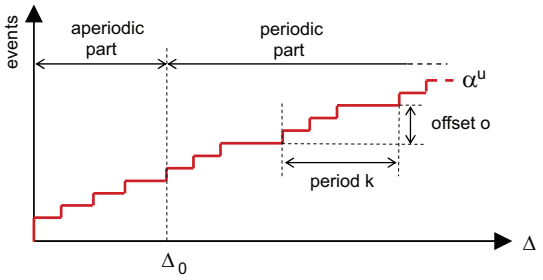


Fig. 2. Representation of an upper arrival curve in the RTC Toolbox.

Using workload curves, the resource-based arrival curves of an event stream can be determined as follows:

$$\alpha^u(\Delta) = \gamma^u(\bar{\alpha}^u(\Delta)) \tag{4}$$
$$\alpha^l(\Delta) = \gamma^l(\bar{\alpha}^l(\Delta)) \tag{5}$$

C. Component Model

Modular Performance Analysis (MPA) models the processing or communication components of a systems with abstract performance components that manipulate arrival and service curves. Fig. 1(a) shows the graphical representation of a general abstract component. The inputs α and β specify the resource demand and service, respectively. The output α' describes the resource demand for the following component whereas β' bounds the remaining service. Depending on the particular interaction between event streams and the available resources, abstract components have different transfer functions $(\alpha', \beta') = \Phi(\alpha, \beta)$. In other words, the transfer function depends on the processing semantics of the modeled component and the resource sharing strategy, i. e. the scheduling method. Besides specifying the transfer functions of components, MPA also permits to analytically bound key timing and memory properties of components. In particular, it provides expressions for the worst-case delay $d = \text{Del}_\Phi(\alpha, \beta)$ and worst-case backlog $b = \text{Buf}_\Phi(\alpha, \beta)$ experienced by some input event of a component.

The abstract performance components that model individual modules of a system are linked to form a system performance model. An example of a system performance model in shown in Fig. 1(b). The network of abstract components has to reflect the dataflow of the system (horizontal connections) as well as the mapping and scheduling of tasks on shared resources (vertical connections and dedicated abstract components). System-wide performance metrics are derived by aggregating local analysis results of the component network.

D. Efficient representation of curves

Arrival and service curves as introduced above are defined on an infinite domain. However, for a practical use of the MPA framework, an appropriate finite datatype has to be adopted. The RTC Toolbox [8], an implementation of MPA-RTC, represents curves by means of an initial aperiodic part followed by a periodic part which is indefinitely repeated (cf. Fig. 2). This specific representation permits to significantly decrease the computational effort when performing operations on arrival and service curves.

III. Combining Heterogeneous Analysis Methods

In order to combine heterogeneous analysis methods and embed them into the component-based analysis framework MPA, we need to define model interfaces. They formally relate the abstract interval-based stream and resource representations (arrival and service curves) to the representations used for the intra-component analysis methods. In this section, we illustrate by means of three different intra-component methods how other analysis methodologies can be embedded and made compositional. In all cases, arrival and service curves model interactions between the heterogeneous component models.

A. Embedding Classical Scheduling Analysis in MPA

The algorithms of classical scheduling theory are often based on a small set of simple task activation models such as periodic event streams or periodic event streams with jitter. These activation patterns are sometimes also denoted as PJD models. A PJD model is specified by three parameters P, J, and D. It represents an event stream in which events arrive periodically with period P but may have a jitter of up to J time units around the ideal periodic arrival time. The parameter D specifies the minimum time between consecutive event arrivals. Arrival curves are more general than PJD event models in the sense that every PJD event model can be tightly expressed by means of a pair of arrival curves but not vice versa. For deriving the arrival curves corresponding to a PJD event model, the following equations can be applied:

$$\bar{\alpha}^u(\Delta) = \min\left\{ \left\lceil \frac{\Delta + J}{P} \right\rceil, \left\lceil \frac{\Delta}{D} \right\rceil \right\} \quad \forall \Delta \geq 0 \quad (6)$$

$$\bar{\alpha}^l(\Delta) = \max\left\{ 0, \left\lfloor \frac{\Delta - J}{P} \right\rfloor \right\} \quad \forall \Delta \geq 0 \quad (7)$$

On the other hand, given a general tuple of arrival curves $[\bar{\alpha}^u, \bar{\alpha}^l]$, it is possible to find a PJD model which conservatively approximates the arrival curves as shown in Fig. 3. While such an approximation might induce pessimism into the analysis, its conservativeness is guaranteed. For deriving a PJD model that approximates $[\bar{\alpha}^u, \bar{\alpha}^l]$, the following algorithm can be used, see [9]. Note that the algorithm assumes that $\bar{\alpha}^u$ and $\bar{\alpha}^l$ have the same long-term slope which is reasonable for realistic systems.

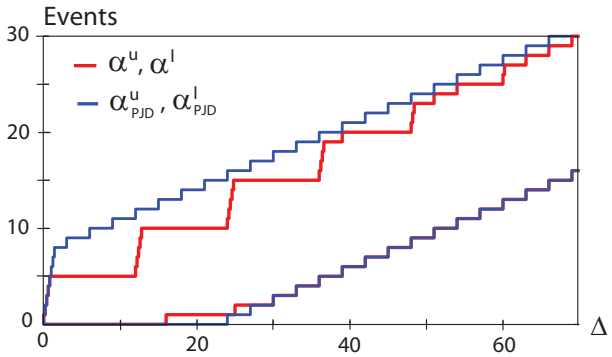


Fig. 3. Approximation of a general arrival curve with a PJD model.

Approximation of an arrival curve with a PJD model

Input: $\bar{\alpha}^u, \bar{\alpha}^l$ (Upper and lower arrival curve)
Output: P, J, D (Period, Jitter, and minimum Distance)

Step 1: *Determine P:* Given the starting point Δ_0 and the period k of the periodic part of $\bar{\alpha}^u$, compute the offset $o = \bar{\alpha}^u(\Delta_0 + k) - \bar{\alpha}^u(\Delta_0)$ and set $P = \frac{k}{o}$.

Step 2: *Determine initial J:* Define an upper curve κ^u corresponding to an event stream with period P, that is, a stream in which one event arrives every P time units. Determine $J^u = \min\{J_0 : \kappa^u(\Delta + J_0) \geq \bar{\alpha}^u(\Delta), \ \forall 0 \leq \Delta \leq \Delta_0 + k\}$.

Step 3: *Adjust J:* Define a lower curve κ^l corresponding to an event stream with period P. Determine $J^l = \min\{J_0 : \gamma^l(\Delta - J_0) \leq \bar{\alpha}^l(\Delta), \ \forall 0 \leq \Delta \leq \Delta_0 + k\}$. Set $J = \max\{J^u, J^l\}$.

Step 4: *Determine D:* If $\bar{\alpha}^u(0+) > 1$ then $D = 0$. Otherwise, $D = \min\{D_0 : \bar{\alpha}^u(D_0) = 2\}$.

Furthermore, it assumes that $\bar{\alpha}^u$ and $\bar{\alpha}^l$ are staircase curves with integer step heights. If this is not the case (e. g. in the presence of a fluid event model), the computation of D has to be adapted and can be replaced by a binary search in the interval $[0, P]$ such that the approximation is as tight as needed but still conservative. The pattern described above forms also the basis for the integration of more recent analysis tools for distributed systems such as MAST [1] or SymTA/S [3], see also [9] for details. It should be noted that recently the SymTA/S approach has been extended towards more general event models [10]. This extension permits to further increase the analysis accuracy when coupling MPA and SymTA/S.

B. Embedding State-based Workload Models in MPA

In complex embedded systems the worst-case workload imposed on a component by a sequence of input events is often smaller than the sum of the individual worst-case workloads. This effect can frequently be observed in multimedia systems, but is also common to many other embedded systems. There are basically two main reasons for the variability of the imposed workload. On the one hand, the input event streams are often not homogeneous themselves, meaning that they consist of events of different types where each event type is characterized by a different worst-case workload. MPEG-2 streams are a common example for event streams with multiple event types. On the other hand, the workload of a single event is sometimes also not fixed but depends on the internal state of the component. For instance, this is the case if a component adopts caching techniques. In the following, we describe two classes of Finite State Maschines (FSM) that permit to capture this behavior and show how FSMs can be used to derive workload curves for a component.

The variability of event types in input event streams can be expressed by means of *Event Sequence Automata* (ESA) [11]. An ESA specifies which are the admissible event type sequences in the input stream.

Definition 3.1: An Event Sequence Automaton F_σ is a tuple (S, S^0, Σ, T) where S is a set of states, $S^0 \subseteq S$ is a set of initial states, Σ is a set of event types, and $T \subseteq S \times \Sigma \times S$ is a set of transitions.

The semantics of an ESA is as follows: If at the arrival of an event of type σ the automaton is in a state s, it will move to a state s', provided that the transition $s \xrightarrow{\sigma} s'$ exists.

Example 3.2: An example of an ESA is shown in Fig. 4(a). The valid runs of the FSM model the admissible event sequences in an input event stream with three different event types *A*, *B*, *C*.

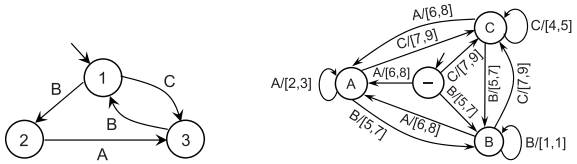


Fig. 4. Example of ESA (a) and WVA (b)

For modeling the component-based workload variability, we employ a second type of FSM, the *Workload Variability Automata* (WVA) [12]. A WVA specifies the workload imposed on the component by an incoming event, depending not only on the event type but also on the internal state of the component.

Definition 3.3: A Workload Variability Automaton F_γ is a tuple (S, S^0, Σ, D, T) where S is a set of states, $S^0 \subseteq S$ is a set of initial states, Σ is a set of event types, D is a workload function $D : S \times \Sigma \times S \rightarrow \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$, and $T \subseteq S \times \Sigma \times \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \times S$ is a set of transitions.

The semantics of a WVA is as follows: If at the arrival of an event of type σ the automaton is in a state s, it will move to a state s' and the component will experience a workload within $[d^l, d^u]$, provided that the transition $s \xrightarrow{\sigma/[d^l,d^u]} s'$ exists.

Example 3.4: An example of WVA is shown in Fig. 4(b). The FSM models the workload variability of a component with an LRU instruction cache. The component accepts three different event types *A*, *B*, *C*. The LRU cache has one cache block which can hold the program code for processing an event of either type *A*, *B*, or *C*. In the FSM the states are labeled according to the program code wich is contained in the cache. We assume that in the initial state the cache is empty. The FSM encodes the variability of the workload imposed by the various events depending on the cache state, that is, the workload is higher (in terms of execution demand) if at the arrival of an event the correct program code is not contained in the cache.

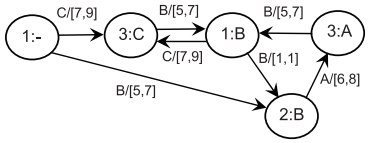


Fig. 5. Product automaton $F_\sigma \times F_\gamma$

Given an ESA F_σ that specifies the admissible event sequences of an input event stream and a WVA F_γ that models the workload variability of a component, we can compute workload curves $[\gamma^u, \gamma^l]$ for the component. In order to do so, we first build the product automaton $F_\sigma \times F_\gamma$. The product automaton resulting from the automata of Fig. 4(a) and 4(b) is shown in Fig. 5. For deriving the upper workload curve $\gamma^u(e)$ we interpret the upper workload bound d^u of each transition in the product automaton as the weight of the transition and compute the maximum-weight path of length e in this weighted product automaton. The procedure for the computation of $\gamma^l(e)$ is analogous.

The concept of Workload Variability Automata has been successfully applied for cache-aware timing analysis of streaming applications [13]. In particular, the authors of [13] discuss how to automatically generate a WVA by analyzing the possible evolutions of the instruction cache of a processor that executes a given streaming application.

C. Embedding Timed Automata in MPA

State-based modeling formalisms extended with some notion of time, e. g. Timed Automata (TA) [14], are well accepted for the verification of real-time systems [2]. Unfortunately, the state/transition systems underlying high-level model descriptions tend to grow exponentially with respect to the number of employed timers, clocks, clock constants, or concurrently executed activities. This effect, known as state space explosion, hampers the analysis of systems, if not making it (practically) impossible due to limited availability of computing resources. Contemporary works have shown how to embed state-based, real-time modeling methods such as TA into analytic frameworks such as modular performance analysis (MPA). Thereby, the following benefits can be achieved: (a) state space explosion is restricted to the level of individual component models. (b) State-based modeling methodologies often allow for less pessimism with respect to key performance measures. This is because analytic validation methods are often too coarse when abstracting system components.

The embedding of state-based, real-time models into an analytic analysis framework is, however, far from trivial: contrary to (formal) state-based methodologies, analytic models lack a concrete execution semantics. For instance, MPA operates on stream abstractions which are defined on time intervals, rather than the conventional time-line. Let us now describe how to embed a state-based and timed component model into MPA. For keeping things simple we restrict the discussion to the case where component interaction is modeled by unidirectional streams of discrete events. These streams, which in the context of MPA are bounded by upper and lower arrival curves, are routed from one system component to another. Within a component the processing of events may require complex computations a priori to the emitting of events on the component's output port. In such a setting one needs to provide the following interfaces:

- Input interface: Arrival curves are translated to state-based, real-time event generators. These generators are capable of producing all timed traces of input events bounded by a pair $\bar{\alpha}^{in}$ of upper and lower arrival curves.
- Output interface: Model checking of the composite of input generator, state-based component model and dedicated observer models allows to construct output arrival curves. These curves bound all traces of output events emitted by the state-based component model when it is triggered by the input generator.

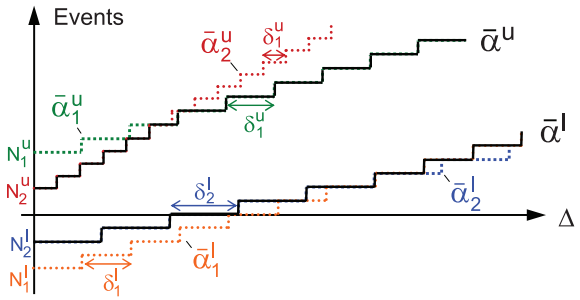
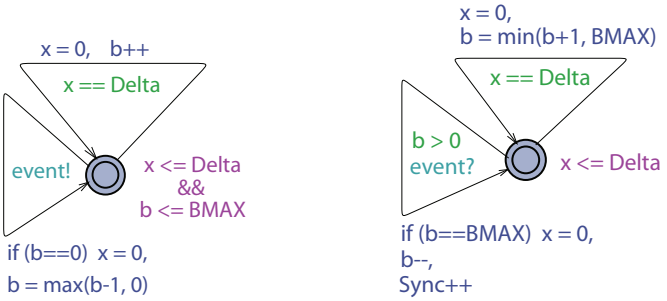


Fig. 6. Arrival curves as combination of staircase functions



(a) LTA for guarding α_j^l

(b) UTA for guarding α_i^u

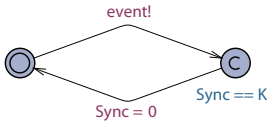
System Declarations

broadcast channel *event*;
int $Sync = 0$;

Local Declarations for LTA, UTA

const int BMAX = $|N_{\{i,j\}}^{\{u,l\}}|$;
clock x;
int $b = 0$; (LTA)
int b = BMAX; (UTA)
const int Delta = $\delta_{\{i,j\}}^{\{u,l\}}$

(c) Declarations



(d) TA for enforcing full synchronization

Fig. 7. TA-based implementation of arrival curves

In the following, we report on two major techniques for implementing such interfaces. It should be noted that the presented techniques are not limited to the couplings as discussed here but they are transferable to other state-based and analytic modeling formalisms. For instance, the embedding of timed Petri nets into the tool SymTA/S [3] can be reduced to a special case of what is discussed next.

C.1 The Approach of Dynamic Counters

The approach of [15] assumes that complex arrival curves can be modeled as nested minmum/maximum combinations of simple staircase functions of the following kind: $\bar{\alpha}_i(\Delta) := |N_i| + \lfloor \frac{\Delta}{\delta_i} \rfloor$. For instance, in Fig. 6 the overall upper arrival curve $\bar{\alpha}^u$ is given as the minimum of two staircase functions $\bar{\alpha}_1^u$ and $\bar{\alpha}_2^u$ and the lower staircase function can be obtained as the maximum of curve $\bar{\alpha}_1^l$ and $\bar{\alpha}_2^l$.

Each staircase function $\bar{\alpha}_i^{\{u,l\}}$ is encoded by means of a dynamic counter. A dynamic counter is a bounded variable $b_i^{\{u,l\}} \in \{0, BMAX_i^{\{u,l\}}\}$ which is incremented periodically, i.e. each $\delta_i^{\{u,l\}}$ time units. Upon event emission $b_i^{\{u,l\}}$ is decremented. The nested minimum/maximum at time t over all $b_i^{\{u,l\}}$ allows to deduce the number of events that are producible or have to be produced at this instant. In other words, each dynamic counter $b_i^{\{u,l\}}$ guards the invulnerability of $\bar{\alpha}_i^{\{u,l\}}$ at any time t. In the following we explain how one implements a dynamic counter with TA and how one coordinates them for obtaining complex arrival curves.

1) TA-Based Modeling of Input Curves: The input generator consists of a set of dynamic counters which are combined in nested minimum/maximum operations to guard an overall arrival curve $\bar{\alpha} = [\bar{\alpha}^u, \bar{\alpha}^l]$. Each counter $b_i^{\{u,l\}}$ is implemented by its own TA where in case of an upper staircase curve $\bar{\alpha}_i^u$ and its guarding counter b_i^u we speak of UTA i and in case of a lower staircase curve $\bar{\alpha}_j^l$ and a counter b_j^l of LTA j. Their TA-templates are shown in Fig. 7 (a) and (b). The minimum and maximum combination of staircase curves on the level of TA can be implemented by exploiting the rendezvous mechanisms as provided by timed model checkers like Uppaal [16].

More specifically, in the TA network the execution of the *event* edges is coordinated in order to obtain the desired overall arrival curve. For instance, the minimum operation of different upper staircase curves can be implemented by full synchronization of the respective UTA. On the other hand, the maximum operation of lower staircase curves can be achieved by appropriate location invariants (cf. Fig. 7 (b)). As a result, the event generator produces all the traces and only the traces of input events which are bounded by the overall arrival curve $\bar{\alpha}$ (the corresponding proofs are provided in [15]).

Example: An example for the encoding of an arrival curve by means of TA is provided by Fig. 6 and Fig. 7: $\bar{\alpha}^u$ is given as the minimum of two staircase curves $\bar{\alpha}_1^u$ and $\bar{\alpha}_2^u$, $\bar{\alpha}^l$ is the maximum of two staircase curves $\bar{\alpha}_1^l$ and $\bar{\alpha}_2^l$. The individual staircase curves are modeled by four corresponding automata UTA 1, UTA 2, LTA 1, LTA 2. Their generic implementations together with a coordinator for event emission is shown in Fig. 7; we assume a correct instantiation of the variables employed in Fig. 7 with the values as provided by Fig. 6, where $BMAX_{\{1,2\}}^{\{u,l\}} := N_{\{1,2\}}^u$ and $Delta_{\{1,2\}}^{\{u,l\}} := \delta_{\{1,2\}}^u$ for the UTA and $BMAX_{\{1,2\}}^{\{u,l\}} := |N_{\{1,2\}}^l|$ and $Delta_{\{1,2\}}^{\{u,l\}} := \delta_{\{1,2\}}^l$ for the LTA and $K = 2$ for the coordinator.

2) TA-Based Detection of Output Curves: On the output interface, the output of a TA-based component model needs to be bounded by upper and lower arrival curves. The parameters for these arrival curves can be deduced by jointly executing the input event generator and the user-defined component model together with a set of dedicated observer TA. Thereby, the individual parameters are derived by means of binary search. As an observer may guard a dynamic counter, its parameters can be used for constructing an upper or lower staircase curve. Once again, the overall output curves are constructed by computing the nested minimum and maximum on a set of staircase curves, each guarded by its own observer TA.

C.2 The Approach of Interval Guards

This technique makes either use of timers or locations for guarding the minimum or maximum number of input or output events. These approaches were implemented in the CATS tool

[17] and on top of a modeling formalism denoted as Event Count Automata (ECA) [18], respectively.

1) Interfaces With Location Invariants: The work of [18] is based on Event Count Automata (ECA). With ECA, the user must specify the minimum and maximum number of event arrivals taking place while the ECA resides in a given location. As a consequence of the implicit notion of time, the modeling of systems with time dependencies may experience severe state space explosion. For translating arrival curves into an ECA, the authors of [18] use the principle of a ring buffer. Each counter represents the number of events associated with the respective number of unit intervals.

For deriving output arrival curves from ECA specifications, the authors of [18] suggest the usage of observer ECA. They use binary search for extracting the maximum and minimum number of events seen in a window size Δ via reachability analysis. As upper (lower) arrival curves are sub-additive (super-additive), a finite number of Δ-values suffice for constructing an upper or lower arrival curve.

2) Interfaces Based on Timers: The approach of [17] samples finitely many pairs of upper and lower point-values from an input arrival curve. Each of these pairs is associated with a respective timer t_i which guards an interval Δ_i. Upon expiration of timer t_i, i.e. every Δ_i time units, it is guaranteed that at least $\bar{\alpha}^l(\Delta_i)$ and at most $\bar{\alpha}^u(\Delta_i)$ events have been produced; in the timed model checker streams where this does not hold are simply excluded by ending in a timelock.

For deriving output arrival curves one reverses the above technique, i.e., with a binary search one finds and upper and a lower bound on the number of output events emitted by the state-based component model. As before, sub- and super-additivity allows to construct (pessimistic) upper and lower output arrival curves from a finite set of samples.

IV. Conclusion

Compositional validation techniques are fundamental for coping with the complexity and heterogeneity of modern embedded system designs. Common system-level methods for compositional worst-case performance analysis such as modular performance analysis (MPA) or SymTA/S are, however, limited to particular analytical component models. If some components in a system exhibit a more complex behavior, e.g. if the system contains state-based components, then the common analytical methods lead to overly pessimistic performance bounds, as they can only conservatively approximate the behavior of complex components. In this paper, we showed how this problem can be mitigated by embedding different modeling formalism into MPA. In particular, we illustrated how finite state machines, timed automata, and methods form classical real-time scheduling theory can be used to refine or substitute component models in MPA. The basic principle behind the coupling of different formalisms is to use arrival curves as general interface between heterogeneous component models. The result is an accurate and scalable framework for performance analysis of complex embedded systems.

References

[1] 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 *Proceedings of 13th Euromicro Conference on Real-Time Systems*. IEEE Computer Society, 2001, pp. 125–134.

[2] C. Norström, A. Wall, and W. Yi, "Timed automata as task models for event-driven systems," in *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications*. IEEE Computer Society, 1999, p. 182.

[3] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System Level Performance Analysis - the SymTA/S Approach," *Computers and Digital Techniques, IEE Proceedings* -, vol. 152, no. 2, pp. 148–166, March 2005.

[4] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *Design Automation and Test in Europe (DATE)*. Munich, Germany: IEEE Press, 2003, pp. 190–195.

[5] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *International Symposium on Circuits and Systems ISCAS 2000*, vol. 4, Geneva, Switzerland, 2000, pp. 101–104.

[6] J. Y. Le Boudec and P. Thiran, *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*, ser. Lecture Notes in Computer Science (LNCS). Springer, 2001, no. 2050.

[7] A. Maxiaguine, S. Künzli, and L. Thiele, "Workload characterization model for tasks with variable execution demand," in *Design Automation and Test in Europe (DATE)*. Paris, France: IEEE Computer Society, 2004, pp. 1040–1045.

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

[9] S. Künzli, A. Hamann, R. Ernst, and L. Thiele, "Combined approach to system level performance analysis of embedded systems," in *5th Intl. Conference on Hardware/Software Codesign and System Synthesis(CODES+ISSS'07)*. ACM, 2007, pp. 63–68.

[10] S. Schliecker, J. Rox, M. Ivers, and R. Ernst, "Providing Accurate Event Models for the Analysis of Heterogeneous Multiprocessor Systems," in *6th Intl. Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'08)*, Atlanta (GA), USA, October 2008.

[11] E. Wandeler, A. Maxiaguine, and L. Thiele, "Quantitative characterization of event streams in analysis of hard real-time applications," *Real-time Systems*, vol. 29, no. 2, pp. 205—-225, 2005.

[12] E. Wandeler and L. Thiele, "Abstracting functionality for modular performance analysis of hard real-time systems," in *Asia and South Pacific Desing Automation Conference (ASP-DAC)*, Shanghai, P.R. China, 2005, pp. 697—-702.

[13] S. Chakraborty, T. Mitra, A. Roychoudhury, and L. Thiele, "Cache-aware timing analysis of streamingapplications," *Real-Time Systems*, vol. 41, pp. 52–85, 2009.

[14] R. Alur and D. L. Dill, "Automata For Modeling Real-Time Systems," in *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, M. Paterson, Ed., 1990, pp. 322–335.

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

[16] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on UPPAAL," in *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, ser. LNCS, M. Bernardo and F. Corradini, Eds., no. 3185. Springer–Verlag, September 2004, pp. 200–236.

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

[18] 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 *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS 2007)*. IEEE Computer Society, 2007, pp. 343–352.