Compositional Analysis of Multi-Mode Systems*

Linh T.X. Phan Insup Lee Oleg Sokolsky
Department of Computer and Information Sciences, University of Pennsylvania
Email: {linhphan, lee, sokolsky}@cis.upenn.edu

***Abstract*—The paper presents a model for multi-mode real-time applications and develops new techniques for the compositional analysis of systems that contain multiple such applications. An algorithm for constructing an interface for a single multi-mode application is presented. Then, a method for computing an interface of a composite application is presented, which uses only the interfaces of constituent applications. A case study of an adaptive streaming system demonstrates that multi-mode analysis offers more precise results compared to a unimodal worst-case analysis.**

I. Introduction

The increasing scale and complexity of real-time embedded systems have prompted the need for advanced analysis techniques that facilitate component-based design. In this design paradigm, a large complex system is first decomposed into smaller and simpler components – which are developed independently – before recomposing them into a complete system using interfaces that abstract away their internal complexities.

To achieve component-based design, interface abstraction and interface composition must adhere to the principle of compositionality, i.e., properties that have been established at the component level hold at the system level. Besides functional and behavioral aspects, real-time embedded systems are concerned with time-constrained resource demands. This necessitates timing analysis frameworks that are compositional, i.e., system-level schedulability analysis should be done by combining component interfaces that abstract component-level timing requirements.

Compositional analysis has therefore been a topic of great interest within the real-time and embedded systems community. Numerous frameworks have been proposed (e.g., [6], [7], [15], [16]) and continuously extended. These frameworks typically assume a static system execution semantics, where a fixed set of tasks/event streams are always active and demanding a fixed amount of resource. They too abstract away event communication between components in the system, replacing it with resource demand placed on the system by event streams. Such an abstraction is adequate for systems that have stable resource-use patterns. In practice, however, systems are often required to operate in multiple *modes* that exhibit vastly different resource demands. Modal behaviors are exhibited by a wide spectrum of embedded systems, ranging from safety-critical aircraft control systems to adaptive streaming servers and smart devices. A networked streaming server, for example, often needs to process several applications concurrently, including audio, video and graphics processing, as well as system-related tasks. These applications are becoming network-aware and adaptive. They may adapt themselves (e.g., select a different encoding algorithm) during runtime depending on the network connection. They may also synchronize with one another, intentionally or inadvertently due to the network conditions.

In such a system, each mode may be characterized by a different set of tasks, different data arrival rates, and a different scheduling policy. Mode switches may be both time-triggered and event-triggered, for instance, due to a time-triggered interrupt or a buffer in the system exceeding a certain fill-level. An incoming event from another component may also trigger a mode switch in the receiving component. Such mode-dependent behavior and mode-switching communication need to be taken into consideration. While internal event triggered may be encapsulated in the resource demand, mode-switching behavior in response to incoming external events needs to be exposed on the component interface.

Our contributions: In this paper, we propose a framework for the compositional analysis of real-time systems which execute multiple multi-mode applications concurrently under a hierarchical scheduling policy on a single processing resource. We present here a multi-mode automaton model (MMA) for modeling such multi-mode applications and an interface-based technique for analyzing them compositionally. Our MMA model refines the previously proposed timed- and event-triggered automata (TET) model described in [11]. While TET is designed for complete system architecture with both processing loads and resource availability, the MMA is concerned only with the application loads and its execution semantics. The service function, that is, resource availability characterization, is calculated during the analysis. This makes the MMA model more declarative than TET and suitable for open systems. Further, the TET model assumes all the task parameters are reset when the system moves to a new mode; the MMA, however, takes into account that some tasks are unaffected by a mode change. As a result, MMA analysis is more precise. Most importantly, we extend the analysis framework in [11] with new techniques for the compositional analysis of hierarchical multi-mode systems.

Our key contributions are summarized as follows:

- We describe the MMA model for multi-mode applications, which refines the previously proposed model in [11] to better represent the application semantics without a specification of resource supply. Compared to [11], the MMA model also employs a more realistic mode change protocol where tasks that are not affected by the mode switch need not be reset when switching to a new mode. We characterize two general ways for handling

*Research is supported in part by NSF grants CNS-0720703 and CNS-0834524, and AFOSR grant FA9550-07-1-0216.

1068-3070/10 $26.00 © 2010 IEEE
DOI 10.1109/ECRTS.2010.35

IEEE computer society

pending event streams during a mode change, and present the composition semantics of MMA in the context of hierarchical scheduling.

- We propose an interface representation for MMA that hides the application-level tasks and scheduling details while exposing timing guards and external events required for synchronization with other component interfaces. Each multi-mode interface is a state machine, where each state is augmented with a service function that represents the resource requirement of a corresponding mode in the application.
- We develop a method for computing the interfaces of MMA that is capable of accurately capturing the effects of mode changes on the resource requirements during transitional periods. Our method also considers buffer conditions and timing guards in deriving the service requirement of a mode. Unlike most previous techniques where pending events in the buffer during a mode change must be finished in the immediate destination mode (hence, no cascading pending events), our technique is designed for the general case. We discuss, however, cases in which such a restriction should be imposed for certain modes in an MMA.
- We illustrate the utility of our model and compositional analysis using a case study of an adaptive streaming system, and compare the obtained results against that of the unimodal compositional analysis.

Related work: Several compositional analysis techniques have been proposed for unimodal systems (see e.g., [1], [6], [16], [17], [19]). These techniques support multiple levels of hierarchy and well-known scheduling policies for real-time systems such as FP (Fixed Priority) and EDF (Earliest Deadline First). They can be broadly divided into two categories: (a) classical periodic and sporadic task models and (b) stream-based task models. In the former, resources are often modeled as periodic, bounded delay, and explicit deadline periodic [6], [16], [17]. Resource interfaces in the latter are presented using service functions, which capture the minimum resource requirements in any interval of a given length [4], [19]. Both approaches abstract away communication between components, allowing analysis to be achieved based on the calculation of resource model interfaces using demand and supply bound functions. A formalism for describing multi-modal components, however, will be useful in system development only if accompanied by a host of analysis techniques that can compute component interfaces as well as detect incompatibilities in communication and resource use of components in a composite system. Hence, these previous techniques become insufficient with the addition of mode-switching behaviors.

Several techniques have been proposed to extend models and timing analysis techniques from the real-time systems literature to accommodate multi-mode behaviors. For example, the framework presented in [3] allows certain tasks to intentionally change their execution periods, which is a type of mode change. Different *mode change protocols* have been studied in [8], [14], [18] and have been classified in [13]. Techniques developed in these papers ensure that no deadlines are violated in each of the modes or during the transition intervals in a single multi-mode component. Our mode change protocol is similar to the one proposed in [18], but is more general as we do not require that buffered events be executed before the mode switch take place and can remain in the buffer through several subsequent mode switches. It is worth noting that these studies of mode switching protocols do not consider the compositional analysis of systems with multiple multi-mode components that are hierarchically scheduled.

Finally, we recently proposed a *multi-mode* extension to the RTC framework [12], which is different when compared to the MMA model that we shall be presenting in this paper. The arrival and service automata in [12] aim at modeling (independently) complex arrival patterns of event streams and resource availability patterns following a relatively simple processing semantics. Here, the MMA allows for a richer processing semantics, with different tasks and scheduling policies explicitly captured and adapted to the dynamic characteristics of the application. The MMA model studied in this paper resembles the TET model proposed in [11] except that MMA has no knowledge of resource supply. In fact, the method proposed here computes such a suitable resource supply that was already given to TET. To the best of our knowledge, our paper is the first attempt to consider the compositional analysis of multi-mode systems.

Organization of the paper: In the next section, we revisit the compositional analysis of unimodal systems. Section III formulates the MMA model and its composition semantics. We present our interface representation and generation technique in Section IV, followed by the interface composition in Section V. We then describe the case study in Section VI before concluding the paper.

II. Compositional Analysis of Unimodal Systems

This section revisits the interface computation of a unimodal component developed in [4], [9], [19], and extends the results for the compositional analysis of unimodal systems that consist of multiple unimodal components that share the same resource under a hierarchical scheduling policy.

A. System description and basic models

A unimodal system consists of a finite set of tasks running on a single processing element, each of which processes an input event stream and produces an output event stream. Each task has an input buffer (to store the input stream) and an output buffer (to store the output stream) that are unshared with other tasks. All tasks process their input events in a FIFO manner. As soon as a task completes its processing of an event, it will remove the event from its input buffer and write the results (as an output event) to its output buffer.

Figure 1 depicts the unimodal model of a system with three tasks T_1, T_2 and T_3 that share the same resource under EDF, where B_i (B'_i) denotes the input (output) buffer of T_i.

In a unimodal system, all tasks are always active and their attributes stay constant throughout the system execution. Each task T is characterized by

$$T = \left(id_T, B_T, B'_T, E_T, D_T, \pi_T, \tilde{\alpha}_T, \alpha_T\right)$$

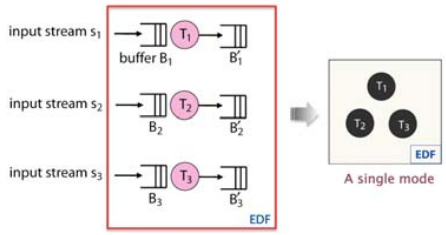

Fig. 1. A unimodal system architecture, modeled by a single mode.

with id_T being the task identifier, B_T the associated input buffer, B'_T the associated output buffer, E_T the execution demand, D_T the relative deadline, π_T the priority if FP is used (and $\pi_T = 0$ otherwise), and $\tilde{\alpha}_T$ the event-based arrival function of the input event stream of T, respectively. Here, $\tilde{\alpha}_T = (\tilde{\alpha}^u_T, \tilde{\alpha}^l_T)$ where $\tilde{\alpha}^u_T(\Delta)$ and $\tilde{\alpha}^l_T(\Delta)$ give the maximum and minimum number of events that can arrive from the associated input stream in any interval of length Δ for all $\Delta \geq 0$. Further, $\alpha_T = E_T \tilde{\alpha}_T$ is the workload-based arrival function of the input stream (i.e., in terms of number of execution units). The *backlog* (*fill-level*) of a buffer at time t refers to the *number of execution units required to process the events* in the buffer at time t. We assume that a granularity of time has been chosen, and events arrive at discrete point in time. Further, the unit of resource supply is an execution time unit. As we are mainly concerned with α^u_T, we shall refer α_T to α^u_T wherever a single function applies.

Scheduling Policies. Given a set of active workloads and a number of execution units provided by a resource within a time interval, a scheduling policy computes the number of execution units allocated to each workload during the interval. Scheduling decisions are often made at discrete points in time, either for the next time unit or until a new event arrives. Each workload denotes the execution requirement of a task, a finite set of tasks or a system component. Denote by $\mathcal{W}$ the set of all workloads. We describe below a general notion of scheduling policy, which is required to define the precise semantics of hierarchical scheduling. Note that this definition captures both work-conserving and non-work-conserving scheduling policies. However, the analysis techniques we present next assume work-conserving scheduling policies.

Definition 1 (Scheduling Policy)**.** *A scheduling policy is a function* $\mathsf{SC} : (\mathcal{W} \times \mathbb{N}) \to (\mathcal{W} \to \mathbb{N})$ *which takes as inputs a finite set of workloads* $W \subseteq \mathcal{W}$*, a non-negative integer* $k \in \mathbb{N}$ *denoting the number of resource units available during a time duration before a new event arrives, and returns as results the number of resource units provided to each workload in* W *in this duration. In other words,* $\mathsf{SC}(W,k)$ *is a function* $g : W \to \mathbb{N}$ *where* $g(w)$ *gives the number of execution units allocated to each* $w \in W$*, such that* $\sum_{w \in W} g(w) \leq k$.

Greedy processing. Suppose T is the only task in the system. Denote $b(t)$ as the backlog (i.e., number of execution units required) of the input buffer B_T at time t. Suppose k is the number of execution units available in an interval $[t, \Delta)$ before a new event arrives. Then, the number of execution units that are provided to B_T during this interval is $\min\{k, b(t)\}$.

Example 1. (Fixed Priority (FP)) *Consider a set of tasks* $T_1, T_2, \ldots, T_n$ *that are scheduled under FP, where* T_i *has higher priority than* T_j *if* $i < j$*. Denote* $b_i(t)$ *as the backlog of* B_{T_i} *at time* t*. Suppose* k *is the number of resource units available in an interval* $[t, t+\Delta)$ *during which there are no new events. Since the resource is first allocated to* T_1 *then to* T_2 *and so on, one can verify that the number of resource unit* h_i *allocated to* T_i *is* $h_1 \stackrel{\text{def}}{=} \min\{k, b_1(t)\}$ *and* $h_i \stackrel{\text{def}}{=} \min\{k - (h_1 + \cdots + h_{i-1}), b_i(t)\}$ *for all* $2 \leq i \leq n$*. Thus,* $\mathsf{FP}(W,k) \stackrel{\text{def}}{=} g$ *where* $g(T_i) = h_i$.

B. Hierarchical scheduling of unimodal systems

In a hierarchical scheduling framework, the system is partitioned into a tree of components that are scheduled in a hierarchical manner as illustrated in Figure 2. Each internal node of the tree represents a *composite component*, whose children are its sub-components. Each leaf represents an *elementary component*, which is a finite set of tasks in the system. Each component has its own scheduling policy under which its sub-components are scheduled, which may differ from its parent's scheduling policy.

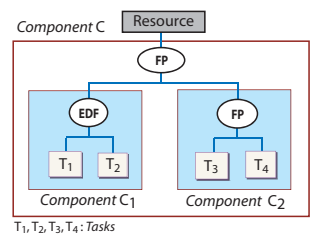

Fig. 2. Hierarchical scheduling of a system component.

C. Compositional analysis of unimodal systems

Consider an elementary component $\mathcal{C} = \langle \tau, \mathsf{SC} \rangle$ consisting of a set of tasks τ that is scheduled under a scheduling policy SC. Denote $\mathcal{F}$ as the set of all functions $f : \mathbb{N} \to \mathbb{N}$. The interface of $\mathcal{C}$ is a minimum *service function* β in $\mathcal{F}$ for which τ is schedulable and none of the input buffers overflows. Recall that a service function $\beta(\Delta)$ specifies the number of execution units available in any interval of length $\Delta \geq 0$.

Suppose SC is EDF. In order for the tasks to be schedulable, the service function β_T allocated to each task T in τ must be at least equal to the demand bound function of T, which is given by $\mathsf{dbf}_T(\Delta) = \alpha_T(\Delta - D_T)$ for all $\Delta \geq 0$. Recall that $\mathsf{dbf}_T(\Delta)$ specifies the maximum possible execution units required by T over any interval of length Δ.

Further, the input buffer B_T does not overflow iff $\alpha_T(\Delta) - \beta_T(\Delta) \leq \mathsf{size}(B_T)$ for all $\Delta \geq 0$. Thus, the minimum service function required by a task T alone is defined by: for all $\Delta \geq 0$:

$$\beta^{\mathsf{uni}}_T(\Delta) = \max\{\alpha_T(\Delta - D_T), \alpha(\Delta) - \mathsf{size}(B_T)\} \qquad (1)$$

where $\mathsf{size}(B)$ denotes the capacity of buffer B. Hence, the interface of $\mathcal{C}$ is the service function $\beta^{\mathsf{uni}}_{\mathsf{edf}}(\mathcal{C}) \stackrel{\text{def}}{=} \sum_{T \in \tau} \beta^{\mathsf{uni}}_T$.

Now suppose SC is FP. We assume without loss of generality (w.o.l.g.) that $\tau = \{T_1, \ldots, T_n\}$ where $\pi_{T_i} < \pi_{T_j}$ if $i < j$. In other words, the total resource will be first allocated to T_1 and its remaining resource will be given to T_2, and so on. Combined with Eq. 1, we imply that the minimum service

function required by T_i under FP is $\widehat{\beta}^{\mathsf{uni}}_{T_n} \stackrel{\mathsf{def}}{=} \beta^{\mathsf{uni}}_{T_n}$, and $\forall 1 \le i < n$,

$$\forall \Delta \ge 0: \ \widehat{\beta}^{\mathsf{uni}}_{T_i}(\Delta) \stackrel{\mathsf{def}}{=} \max\left\{ \mathsf{Serv}\left(\widehat{\beta}^{\mathsf{uni}}_{T_{i+1}}, \alpha_{T_i}\right)(\Delta),\ \beta^{\mathsf{uni}}_{T_i}(\Delta) \right\}$$

where $\mathsf{Serv}(\beta', \alpha)$ denotes the smallest service function $\beta \in \mathcal{F}$ required by a task with arrival function α_T such that the remaining service function after processing the task is at least β', which is given by [4]: $\mathsf{Serv}(\beta', \alpha_T) = \beta'(\Delta - \lambda) + \alpha_T(\Delta - \lambda)$ where $\lambda = \sup\{\varepsilon \mid \beta'(\Delta - \varepsilon) = \beta'(\Delta)\}$. Thus, the interface of $\mathcal{C}$ is the service function $\beta^{\mathsf{uni}}_{\mathsf{fp}}(\mathcal{C}) \stackrel{\mathsf{def}}{=} \widehat{\beta}^{\mathsf{uni}}_{T_1}$.

Next, consider a component $\mathcal{C}$ comprising m components $\mathcal{C}_1, \ldots, \mathcal{C}_m$ that are scheduled using SC. Let β_i be the interface of $\mathcal{C}_i$ for all $1 \le i \le m$. Again, we assume w.o.l.g. that when SC is FP, C_i has higher priority than C_j if $i < j$. The interface of $\mathcal{C}$ is the minimum service function $\beta^{\mathsf{uni}}_{\mathsf{sc}}(\mathcal{C})$ for which all C_i are schedulable. It is given by $\sum_{i=1}^{m} \beta_i$ if SC is EDF and by $\widehat{\beta}^{\mathsf{uni}}_{\mathcal{C}_1}$ if SC is FP, with $\widehat{\beta}^{\mathsf{uni}}_{\mathcal{C}_m} = \beta_m$ and $\widehat{\beta}^{\mathsf{uni}}_{\mathcal{C}_i} = \mathsf{Serv}(\widehat{\beta}^{\mathsf{uni}}_{\mathcal{C}_{i+1}}, \beta_i) \forall 1 \le i < m$.

III. Multi-mode Automata (MMA)

Multi-mode systems execute at multiple modes of operation, each of which performs a unique functionality. The set of tasks that are active, their attributes, as well as the scheduling policy during the system execution vary with respect to the mode at which the system is currently in. Changes from one mode to another can be triggered by a timing guard and/or an event. Such systems can be modeled by a multi-mode automaton, or MMA in short. A complex multi-mode system can be composed of a collection of multi-mode systems that share the same resource under a hierarchical scheduling policy, which are modeled as an MMA each.

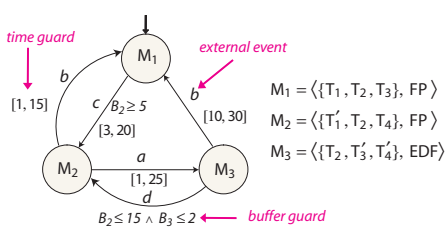

Fig. 3. An example of a multi-mode automata.

An MMA is a finite automaton whose states represent operating modes and transitions represent mode changes. Each mode (state) of an MMA specifies (i) a set of tasks (and their corresponding attributes) that are active in the mode, and (ii) a scheduling policy used to schedule the tasks. Each transition of an MMA can be triggered by an external event (e.g., interrupt), a condition on the fill-levels of the buffers associated with the tasks, or a timing constraint. Denote INT as the set of intervals $[a, b]$ with $0 \le a \le b$ and $a, b \in \mathbb{N}$.

Definition 2 (MMA)**.** *An MMA is a tuple* $\mathcal{A} = (\mathcal{M}, M_{\mathsf{in}}, \mathsf{Inv}, \Phi, \Sigma, R, \mathcal{T})$ *where*

- $\mathcal{T} = \{T_1, \ldots, T_m\}$ *a finite set of tasks of the application.*
- $\mathcal{M}$ *is a finite set of modes. Each mode* $M \in \mathcal{M}$ *has the form* $\langle \tau, \mathsf{SC} \rangle$ *with* $\tau \subseteq \mathcal{T}$ *denoting the set of active tasks and* SC *the scheduling policy at* M*. We say* B_T *is active in* M *iff* T *is active in* M*.*
- $M_{\mathsf{in}} \in \mathcal{M}$ *is the initial mode of* $\mathcal{A}$*.*
- Φ *is a set of constraints on the fill-levels of the buffers in* $\mathcal{B}$*, where* $\mathcal{B} = \bigcup_{T \in \mathcal{T}} B_T$*.*
- $\mathsf{Inv}: \mathcal{M} \to \mathsf{INT}$ *is an invariant function that assigns to each mode in* $\mathcal{M}$ *an interval* $[I_l, I_u]$*, where* I_l *is the minimum amount of time that the system must stay in the mode and* I_u *is the maximum amount of time that the system may stay in the mode. We require that* $I_l \ge 1$*.*
- Σ *is a set of signals that trigger the mode changes.*
- $R \subseteq \mathcal{M} \times \Sigma \times \Phi \times \mathsf{INT} \times \mathcal{M}$ *is a transition relation. Each transition in* R *is of the form* (M, a, φ, I, M') *where* M *and* M' *are the origin and destination modes,* a *is an external signal that triggers the transition,* φ *is a guard on the fill-levels of the input buffers, and* $I \in \mathsf{INT}$ *is the interval (relative to the instant the system enters* M*) during which the transition can be enabled.*

All modes are urgent and when multiple transitions are enabled concurrently, the MMA non-deterministically selects one.

Fig. 3 depicts an example of MMA. Here, T_i and T_i' correspond to the same task of the application, thereby associated with the same input/output buffer. Their timing attributes, however, are different. Initially, the system is in mode M_1, where T_1, T_2, T_3 are active and scheduled under FP. When the system detects – after it has been in M_1 for at least 3 and no more than 20 time units – that a new task T_4 arrives and the arrival function of T_1 is changed, and if the fill-level of the input buffer B_2 of T_2 is more than 5, then it will move to mode M_2. At M_2, the system deactivate the least important task T_3 and T_4 is executed together with T_1' and T_2.

Mode change protocols. Whenever a transition is enabled, the system moves instantaneously to the new mode, and the new parameters will be in effect for all new incoming events immediately. Unchanged tasks (appearing in both modes) are not affected by the mode change, and no new events from an old task (appearing only in the old mode) arrives in the new mode. New input events of the new tasks (only appear in the new mode) and changed tasks (whose parameters are modified) may arrive immediately.[1] Pending events in the input buffer of an old or a changed task may be handled differently depending on the application, which can be generalized into two cases:

(i) Both the physical pending events and their timing requirements (i.e., execution demand and deadline) associated with the old mode are preserved in the new mode.

(ii) Only the physical pending events are preserved in the new mode; their timing requirements follow the modified parameters of the associated task in the new mode.

Note that the execution demanded by an event depends on the current mode of the system when the event arrives at the system in case (i), and on the mode at which it is processed in case (ii). For the rest of the paper, we assume the former case; however, results for the latter can be established in a similar (and simpler) fashion. We assume also that all tasks follow the priorities specified in the new mode regardless of which event it is processing. Lastly, when the system is in a mode, only active tasks in the mode are being executed.[2]

[1] Note that most common protocols require a delay before the system moves to the new mode. This can be done by adding an intermediate mode with a time invariant equal to the delay and old tasks having zero arrival functions.

[2] Pending events of an old task can be processed in the new mode by adding the task into the active task set of the new mode and assigning its arrival function to be a zero function.

A. Composition of multi-mode automata

Systems that consist of multiple applications running concurrently can be reasoned by means of composition of MMA. There are two types of compositions: with and without resource sharing. In this paper, we shall focus on the composition of independent MMA that execute on the same resource under some scheduling policy.

B. Mode composition

Let $M_1 = \langle \tau_1, \mathsf{SC}_1 \rangle$ be a mode of an MMA $\mathcal{A}_1$ and $M_2 = \langle \tau_2, \mathsf{SC}_2 \rangle$ be a mode of an MMA $\mathcal{A}_2$. Suppose $\mathcal{A}_1$ and $\mathcal{A}_2$ share the same resource under a scheduling policy SC_3. Consider any time interval $[t, t+\Delta)$ during which $\mathcal{A}_1$ is in M_1, A_2 is in M_2, and no new event arrives at both automata. Suppose during $[t, t+\Delta)$, the processor has k resource units available to execute $\mathcal{A}_1$ and $\mathcal{A}_2$. By Definition 1, the number of resource units allocated to τ_1 and τ_2 are $k_1 = \mathsf{SC}_3(\tau, k)(\tau_1)$ and $k_2 = \mathsf{SC}_3(\tau, k)(\tau_2)$, respectively, with $\tau = \tau_1 \cup \tau_2$. As a result, the number of resource units allocated to each task $T \in \tau$ is: $g(T) = \mathsf{SC}_1(\tau_1, k_1)(T)$ if $T \in \tau_1$, and $g(T) = \mathsf{SC}_2(\tau_2, k_2)(T)$ otherwise. Define $\mathsf{SC}(\tau, k) \stackrel{\text{def}}{=} g$. The composition of M_1 and M_2 under SC_3, denoted by $M_1 \|_{\mathsf{SC}_3} M_2$, is a mode $M = \langle \tau, \mathsf{SC} \rangle$.

Definition 3. *Let $\mathcal{A}_1 = (\mathcal{M}_1, M_{\mathsf{in1}}, \mathsf{Inv}_1, \Phi_1, \Sigma_1, R_1, \mathcal{T}_1)$ and $\mathcal{A}_2 = (\mathcal{M}_2, M_{\mathsf{in2}}, \mathsf{Inv}_2, \Phi_2, \Sigma_2, R_2, \mathcal{T}_2)$ be two MMA. The asynchronous composition of $\mathcal{A}_1$ and $\mathcal{A}_2$ under SC, denoted by $\mathcal{A}_1 \|_{\mathsf{SC}} \mathcal{A}_2$, is a tuple $\mathcal{A} = (\mathcal{M}, M_{\mathsf{in}}, \mathsf{Inv}, \Phi, \Sigma, R, \mathcal{T})$ where:*

- *$\mathcal{M}$ is the set of modes, defined by $\mathcal{M} = \{ M_1 \|_{\mathsf{SC}} M_2 \mid M_1 \in \mathcal{M}_1 \wedge M_2 \in \mathcal{M}_2 \}$.*
- *$M_{\mathsf{in}} = M_{\mathsf{in1}} \|_{\mathsf{SC}} M_{\mathsf{in2}}$ is the initial mode of $\mathcal{A}$.*
- *Inv is the invariant, where $\mathsf{Inv}(M_1 \|_{\mathsf{SC}} M_2) = [1, \min(U, U')]$, with $\mathsf{Inv}(M_1) = [L, U]$, $\mathsf{Inv}(M_2) = [L', U']$.*
- *Φ is the set of constraints on the fill-levels of the buffers B_T where $T \in \mathcal{T}$.*
- *$\Sigma = \Sigma_1 \cup \Sigma_2$ is the set of external triggering signals.*
- *$R \subseteq \mathcal{M} \times \Sigma \times \Phi \times \mathsf{INT} \times \mathcal{M}$ is the transition relation, defined as follows. For each $(M_1, a_1, \varphi_1, [L, U], M_1')$ in R_1 and each $(M_2, a_2, \varphi_2, [L', U'], M_2')$ in R_2:*
  - (i) *$(M_1 \|_{\mathsf{SC}} M_1', a_1, \varphi_1 \wedge \varphi_2, I, M_2 \|_{\mathsf{SC}} M_2')$ is a transition in R if $a_1 = a_2$, $M_1 \neq M_2$ and $M_1' \neq M_2'$ where $I = [\min(L, L'), \min(U, U')]$.*
  - (ii) *$(M_1 \|_{\mathsf{SC}} M_1', a_1, \varphi_1, [1, \min(U, U')], M_2 \|_{\mathsf{SC}} M_1')$ is a transition in R if $a_1 \neq a_2$, $M_1 \neq M_2$ and $M_1' = M_2'$.*
  - (iii) *$(M_1 \|_{\mathsf{SC}} M_1', a_2, \varphi_2, [1, \min(U, U')], M_1 \|_{\mathsf{SC}} M_2')$ is a transition in R if $a_1 \neq a_2$, $M_1 = M_2$ and $M_1' \neq M_2'$.*
- *$\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$ is the set of tasks in the system.*

Note that in case (ii), it is possible that $\mathcal{A}_1$ enters M_1 at time t and $\mathcal{A}_2$ enters M_2 at time $t+L$. Because $\mathcal{A}_1$ has already spent L time units in M_1, it can move to M_1' immediately or after at least 1 time unit. If $\mathcal{A}_1$ moves immediately to M_1', then both $\mathcal{A}_1$ and $\mathcal{A}_2$ move to a new mode at time $t+L$, which can be captured by case (i). By similar arguments, we obtain the time intervals associated with the transitions in cases (i) and (iii).

The synchronous composition of MMA can be defined similarly; however, only synchronized transitions are allowed (i.e., case (i)) and each composed mode (transition) is associated with an interval $I = I_1 \cap I_2$, where I_1 and I_2 are the intervals associated with the two component modes (transitions).

IV. Compositional Analysis of MMA

Similar to the unimodal case, the compositional analysis of multi-mode systems begins with the system being partitioned into a finite set of components that are hierarchically scheduled. The only difference here is that each elementary component in the hierarchy is now a multi-mode system, modeled as an MMA. Hence, we first compute an interface that captures the resource requirements for each MMA, and subsequently generate the interface of a composite component from the computed interfaces of its children.

A. MMA's interface representation

The resource requirement of a multi-mode component $\mathcal{C}$ can be captured by a *multi-mode resource interface*, which is a finite state machine where each state is augmented with a minimum service function that is demanded by $\mathcal{C}$. The different service functions associated with different states of the interface represent the different resource requirements of $\mathcal{C}$ when it is at different (set of) modes. Each transition in the interface signifies a "service change request" by the component, which is triggered by a signal or a time invariant. Multi-mode resource interfaces share the same set of external triggering signals as that of their components. In other words, they expose communication between components, and hence allowing detection of incompatibilities in communication and resource use of components in a composite system during interface composition. On the other hand, internal events (in the form of buffer constraints) that exist in a component but are unobservable to other components are abstracted away and replaced with the service functions of the interface. This enables information hiding, at the same time limits the interface's complexity.

Definition 4 (Multi-mode Resource Interface)**.** *A resource interface of a multi-mode component $\mathcal{C}$ is a finite state machine $A = (S, s_{\mathsf{in}}, \beta, \Sigma, R)$ where*

- *S is a finite set of states, each of which characterizes the resource requirement of one or more modes in $\mathcal{C}$.*
- *$s_{\mathsf{in}} \in S$ is the initial state.*
- *$\beta : S \to \mathcal{F}$ is a service mapping, which specifies for each state $s \in S$ a minimum service function $\beta(s)$ that must be guaranteed at s for $\mathcal{C}$ to be schedulable.*
- *Σ is a set of external signals.*
- *$R \subseteq S \times \Sigma \times \mathsf{INT} \times S$ is a set of transitions. Each transition $\mathsf{tr} = (s, a, [L, U], s')$ in R represents a change in the resource requirement of the component (i.e., from $\beta(s)$ to $\beta(s')$), which is triggered by a signal a and a time interval $[L, U]$ during which the transition can be enabled.*

All transitions in R are instantaneous and all states in S are urgent.

As an example, Fig. 4(b-c) shows an adaptive stream processing system that consists of an audio application and a video application (besides others), which change modes according to the network condition (i.e., upon presence of one of the external events *unloaded*, *loaded*, and *congested*). The multi-mode resource interface for the system is given in Fig. 5. In the figure, the service function β_s where $s \in \{\mathsf{PCM15}, \mathsf{ADM7.5}, \mathsf{PLC7.5}, \mathsf{PCM7.5}\}$ gives the minimum

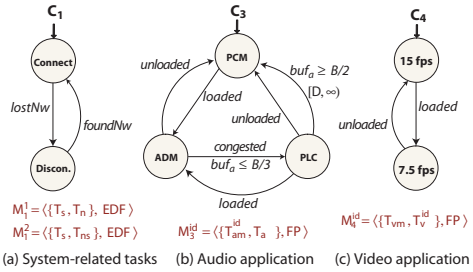

Fig. 4. MMA models of concurrent applications in a streaming system.

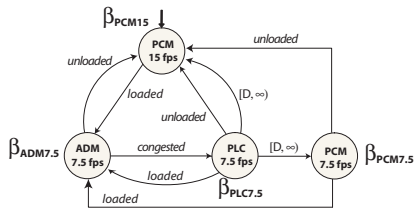

Fig. 5. The multi-mode interface for the composite component $\mathcal{C}_2 = \mathcal{C}_3 \parallel \mathcal{C}_4$.

resource that must be provided to the applications when they are in the corresponding modes. For instance, β_{PCM15} is the minimum service function that must be guaranteed when the audio application encodes data using PCM algorithm and the video application sends data at 15 fps (frames per second). As illustrated in the figure, buffer constraints in the applications have been hidden while information necessary for synchronizing with other components (i.e., timing and network condition) are being exposed on the interface.

Before presenting the method for computing such an interface, we introduce concepts and technical results needed for the computation. In what follows, $[L_M, U_M]$ denotes $\mathsf{Inv}(M)$.

B. Properties of pending events during mode changes

In an MMA, when the system moves from one mode to another, there may be pending events in the buffers. We characterize below two intertwined terms that capture these events: the former specifies their execution demand over time, and the latter specifies their backlog.

Definition 5 (Carried-in demand bound function)**.** *The carried-in demand bound function of a buffer B at mode M, denoted by $\mathsf{cidf}_{M,B}$, specifies for each $\Delta \geq 0$ the maximum number of execution units demanded by the pending events in B (when the system enters M) whose deadlines are within Δ time units from the instant the system enters M.*

Definition 6 (Carried-in backlog)**.** *The carried-in backlog of a buffer B at mode M, denoted by $\mathsf{bin}_{M,B}$, gives the maximum backlog of B when the system enters M.*

Fig. 6 gives an example of the carried-in demand bound function and backlog. In the figure, an up (down) arrow denotes the point at which an event arrives (must be finished).

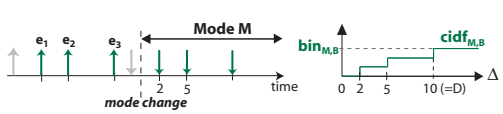

Fig. 6. Carried-in demand bound function and backlog.

Observe that the latest instant at which these pending events must be completed is the smallest D such that $\mathsf{cidf}_{M,B}(\Delta) = \mathsf{cidf}_{M,B}(D)$ for all $\Delta \geq D$. We call D the "deadline" of $\mathsf{cidf}_{M,B}$. Since the maximum backlog of B when the system enters M is the total execution demands of the pending events, the following corollary holds.

Corollary IV.1. *The carried-in backlog of B at M is given by $\mathsf{bin}_{M,B} = \mathsf{cidf}_{M,B}(D)$, where D is the deadline of $\mathsf{cidf}_{M,B}$.*

Further, due to these pending (old) events, the execution demand required by the (old and new) events in the buffer during the initial period after the system enters the new mode is often larger than the usual execution demand required at a later time interval. We call this the *initial demand*, which is defined for absolute time intervals that begin at the instant the system enters the new mode.

Definition 7 (Initial demand bound function (IDBF))**.** *The initial demand bound function of a buffer B at mode M, denoted by $\mathsf{idbf}_{M,B}$, specifies for each $\Delta \geq 0$ the maximum number of execution units demanded by the events in B in the interval $[0, \Delta]$ relative to the instant the system enters M.*

One can easily verify that the overall demand bound function of a buffer B at a mode M, which gives the maximum execution demand of the events in B in any interval of length Δ when the system is at M, is at most $\max\{\mathsf{idbf}_{M,B}, \mathsf{dbf}_{M,B}\}$. Here, for all $\Delta \geq 0$, $\mathsf{dbf}_{M,B}(\Delta) = \alpha_T(\Delta - D_T)$ if B is active at M, and $\mathsf{dbf}_{M,B}(\Delta) = 0$ otherwise. This implies Lemma IV.2, the proof of which is available in [10]. For convenience, we denote $T_{M,B}$ as the active task associated with B in mode M, and $T_{M,B} = \emptyset$ if B is not active in M.

Lemma IV.2. *The minimum service function required by the events in B to ensure (i) B does not overflow and (ii) all the events in B meet their deadlines while the system is at M is:*
$\beta_{M,B} \stackrel{\text{def}}{=} \max\{\mathsf{bin}_{M,B} + \alpha - \mathsf{size}(B), \mathsf{idbf}_{M,B}, \mathsf{dbf}_{M,B}\}$, *where $\alpha = \alpha_{T_{M,B}}$ if B is active in M and $\alpha(\Delta) = 0 \forall \Delta \geq 0$ otherwise.*

Computing initial demand bound function. The IDBF of a buffer B at a mode M can be computed with respect to (w.r.t.) an execution path that leads to M. Suppose p is a path from the initial mode M_{in} to M. If $M \equiv M_{\mathsf{in}}$ or B is inactive in all preceding modes of M in p, then $\mathsf{cidf}_{M,B}(\Delta) = 0$ for all $\Delta \geq 0$ and $\mathsf{idbf}_{M,B} = \mathsf{dbf}_{M,B}$. Otherwise, let $\rho = M_1 \xrightarrow{[L_1,U_1]} M_2 \xrightarrow{[L_2,U_2]} \cdots M_k \xrightarrow{[L_k,U_k]} M_{k+1} \equiv M$ be the longest sub-path of p along which the task associated with B does not change its parameters, i.e., (i) $k = 1$ and $T_{M_1,B} = \emptyset$, or (ii) $T_{M_i,B} \neq \emptyset$ and is unchanged for all $1 \leq i \leq k$; further, $\rho = p$ or $T_{M_0,B} \neq T_{M_1,B}$, where M_0 is the immediate ancestor of M_1 in p.

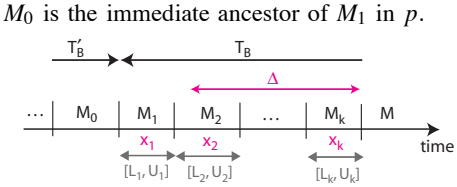

Fig. 7. The service function of a path that leads to M.

As illustrated in Fig. 7, $[L_i, U_i]$ is the interval (relative to the instant the system enters M_i) during which the system can move from M_i to M_{i+1}, i.e., $[L_i, U_i] = \mathsf{Inv}(M_i) \cap [L'_i, U'_i]$ where $[L'_i, U'_i]$ is the interval associated with the transition from M_i to M_{i+1} in the MMA. Let $L_p = \sum_{i=1}^{k} L_i$ and $U_p = \sum_{i=1}^{k} U_i$. Then,

L_p and U_p are the minimum and maximum total amount of time the system can spend at $M_1, \ldots, M_k$ before moving to M. The path ρ is called the "unchanged-arrival preceding path" of B at M w.r.t. p, denoted by $p^{\alpha}_{M,B}$.

Definition 8. *The overall service function $\beta^p_B(\Delta)$ given to B w.r.t. a path p that ends with M specifies the minimum number of execution units allocated to B in the interval of length Δ immediately before the system enters M.*

Lemma IV.3. *Suppose β_i is the service function allocated to B at mode M_i for all $1 \le i \le k$. Then, for all $\Delta \in [0, U_p]$,*

$$\beta^p_B(\Delta) = \min\Big\{\sum_{i=1}^{k} \beta_i(x_i) \mid \Delta = \sum_{i=1}^{k} x_i \wedge (L_i \le x_i \le U_i \vee \big(x_1 = \cdots = x_{i-1} = 0 \wedge x_i < L_i\big)\Big)\Big\}.$$

Proof: Consider an interval of length Δ immediately before the system enters M (see Fig. 7). Let x_i be the number of time units the system spends at M_i. The total number of execution units given to B is then $\sum_{i=1}^{k} \beta_i(x_i)$. Now let M_j be the mode at which the interval begins. Then, the system spends zero time units at all modes before M_j, and from L_i to U_i time units at each mode M_i after M_j. Thus, $x_1 = \ldots = x_{j-1} = 0$ and $L_i \le x_i \le U_i$ for all $i > j$. Hence the lemma. ■

Lemma IV.4. *Suppose t is the instant the system enters M by taking p. The maximum number of execution units demanded by the pending events in B when the system enters M_1 that need to be fulfilled in interval $[t-x, t+\Delta]$, with $x \le U_p$, is*

$$\mathsf{cidf}^p_{M,B}[x,\Delta] = \mathsf{cidf}_{M_1,B}(U_p + \Delta) - \mathsf{cidf}_{M_1,B}\big((L_p - x)_+\big).$$

Fig. 8 demonstrates the results stated by Lemma IV.4. Details of its proof can be found in [10].

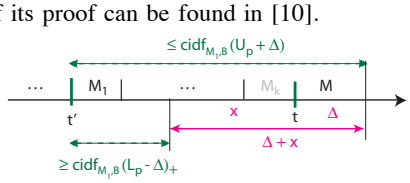

Fig. 8. Carried-in demand bound function w.r.t. a path.

Lemma IV.5. *Let φ be the buffer guard associated with the transition from M_k to M, and B_φ be the largest fill-level of B that satisfies φ. Let $T \equiv T_{M_1,B}$ and $\lambda = (D_T - \Delta)_+$. Then, for all $\Delta \ge 0$,*

$$\mathsf{cidf}^p_{M,B}(\Delta) = \min\Big\{B_\varphi, \sup_{0 \le x \le U_p}\big\{\alpha_T(x-\lambda) + \mathsf{cidf}^p_{M,B}[x,\Delta] - \beta^p_B(x)\big\}\Big\}$$

The proof of the lemma is available in [10].

Recall that $\mathsf{dbf}_{M,B}(x)$ denotes the maximum execution demand of the events that arrive at B in any time interval of length x when the system is at M_i. It is given by $\mathsf{dbf}_{M,B}(x) = \alpha_T(x - D_T)$ if the task T associated with B is active in M, and $\mathsf{dbf}_{M,B}(x) = 0$ otherwise. Lemma IV.6 states the relationship between the initial and carried-in demand bound functions.

Lemma IV.6. *If $T_{M,B} \ne T_{M_1,B}$, then $\mathsf{idbf}^p_{M,B}(\Delta) = \mathsf{cidf}^p_{M,B}(\Delta) + \mathsf{dbf}_{M,B}(\Delta)$. Otherwise,*

$$\mathsf{idbf}^p_{M,B}(\Delta) = \min\Big\{\mathsf{cidf}^p_{M,B}(\Delta) + \mathsf{dbf}_{M,B}(\Delta), \sup_{0 \le x \le U_p}\big\{\mathsf{dbf}_{M_1,B}(x+\Delta) + \mathsf{cidf}^p_{M,B}[x,\Delta] - \beta^p(x)\big\}\Big\}.$$

The proof of Lemma IV.6 can be established using similar arguments to that of Lemma IV.5. Observe that our computation above considers the complete unchanged-arrival path, starting from M_1. The results can be tighten by taking the minimum of the computed values for all paths starting from M_i where $1 \le i \le k$.

C. Computing service function of a mode w.r.t. a path

Let $M = \langle \tau, \mathsf{SC} \rangle$ be a mode of the MMA, reachable through a path p. Let $\rho = M_1 \xrightarrow{[L_1,U_1]} M_2 \xrightarrow{[L_2,U_2]} \cdots M_k \xrightarrow{[L_k,U_k]} M_{k+1} \equiv M$ be the unchanged-arrival preceding path of B at M w.r.t. p, i.e., $\rho = p^{\alpha}_{M,B}$. Based on Lemma IV.5 and IV.6, we compute the carried-in demand bound function $\mathsf{cidf}^p_{M,B}$ and the initial demand bound function $\mathsf{idbf}^p_{M,B}$ of B at M w.r.t. p. Apply $\mathsf{cidf}^p_{M,B}$ to Corollary IV.1, we imply the carried-in backlog of B at M, i.e., $\mathsf{bin}^p_{M,B} = \mathsf{cidf}^p_{M,B}(D)$ where D is the *deadline* of $\mathsf{cidf}^p_{M,B}$. From $\mathsf{idbf}^p_{M,B}$ and $\mathsf{bin}^p_{M,B}$, we then derive minimum service function $\beta^p_{M,B}$ that is required by B at M (without considering other tasks) using Lemma IV.2.

Requirements of the active tasks at M. In order for the events in the buffers to meet their deadlines, we require that $T_{M,B}$ is active for all buffers B such that $\beta^p_{M,B}(\Delta) > 0$ for some $\Delta \le U_M$. Otherwise, in the worst case, the system may stay at M for up to U_M units of time; in which case, some event in B will miss its deadline.

Overall service function required by M. The minimum service function β^p_M that is required by a mode M assuming the system entering M through p can be computed from all β^p_{M,B_i} in the same fashion as done in the unimodal case. First, we assume w.o.l.g. that $\tau = \{T_1, \ldots, T_n\}$ where T_i has higher priority than T_j if $i < j$. Denote B_i as the buffer associated with T_i and α_i as the arrival function of T_i at M.

If SC is EDF. Then, the overall service function for M is the sum of all service function required by all the buffers B_i, i.e., $\beta^p_M \stackrel{\text{def}}{=} \sum_{i=1}^{n} \beta^p_{M,B_i}$. Now suppose SC is FP. Then, the overall service function for M is $\beta^p_M \stackrel{\text{def}}{=} \widehat{\beta}^p_{M,B_1}$, where $\widehat{\beta}^p_{M,B_n} = \beta^p_{M,B_n}$ and $\widehat{\beta}^p_{M,B_i} = \max\big\{\mathsf{Serv}(\widehat{\beta}^p_{M,B_{i+1}}, \alpha_i), \beta^p_{M,B_i}\big\}$ for all $1 \le i < n$. One can verify that $\beta^p_M \ge \beta^{\mathsf{uni}}_M$ and $\beta^p_{M,B_i} \ge \beta^{\mathsf{uni}}_{T_i}$ for all $1 \le i \le n$. The equality occurs if p comprises a single mode M ($\equiv M_{\mathsf{in}}$).

D. Computing interfaces of multi-mode applications

With the above results, we now proceed to compute the interface $\textsc{inf}(\mathcal{C})$ of an elementary component $\mathcal{C}$, where $\mathcal{C}$ is a multi-mode application modeled as an MMA. The computation for the interface of a composite component made of multiple sub-components will be outlined in the next section.

Basic ideas: Consider $\mathcal{C} = (\mathcal{M}, M_{\mathsf{in}}, \mathsf{Inv}, \Phi, \Sigma, \mathrm{R}, \mathcal{T})$. To compute the interface $\textsc{inf}(\mathcal{C})$, the idea is to allocate as little resource as possible to each mode of $\mathcal{C}$ while maintaining schedulability of the active tasks as well as no buffer overflows condition when the system is at the mode. This resource needs to be sufficient to take care of the initial carried-in execution demand of pending events when the system enters a mode, as considered in the preceding sections. To achieve this, we construct a reachable tree of $\mathcal{C}$ by exploring $\mathcal{C}$, starting from the initial mode as the root of the tree. At each reachable mode M in the tree along a path p, we compute the service

function $\beta^p_{M,B}$ required by each buffer B at M and the overall service function β^p_M required by M w.r.t. p. We then add M' into the set of reachable modes to be explored if there is a transition $\mathsf{tr} = (M, a, \varphi, [L', U'], M')$ in $\mathcal{C}$ and the carried-in backlog $\mathsf{bin}^p_{M',B}$ satisfies φ for all buffer B in the system.

During the above exploration, if we reach a mode M' that has been visited earlier and the newly computed service function of M' is less than or equal to its most recently computed value, we mark M' as "leaf". Otherwise, we update the service function of M to be the maximum between its recently computed value and the newly computed one, and explore M' further. We repeat this process until there is no more reachable modes to be explored.

We denote β_M and $\beta_{M,B}$ as the most recently updated values of the service function required at M by all the buffers and for B, respectively. When the superscript p is involved, they refer to the new values computed w.r.t. a path p.

The exploration procedure:

a) Initialization: For each mode M and each buffer B in $\mathcal{C}$, we initialize $\beta_M(\Delta) = 0$ and $\beta_{M,B}(\Delta) = 0$ for all $\Delta \geq 0$. We start with the initial mode M_{in}. Since all the buffers are initially empty, for each input buffer B in $\mathcal{C}$, $\mathsf{bin}_{M_{\mathsf{in}},B} = 0$ and $\mathsf{cidf}_{M_{\mathsf{in}},B}(\Delta) = 0$ for all $\Delta \geq 0$. We include $\langle M_{\mathsf{in}}, p\rangle$ as the first element in the set of modes to be explored $\mathcal{S}$, where $p = M_{\mathsf{in}}$.

b) Computation for each node $v = \langle M, p\rangle$ in $\mathcal{S}$: We first remove v from $\mathcal{S}$. Follow the technique outlined in Section IV-B and IV-C, we compute (i) the service function $\beta^p_{M,B}$ required by each input buffer B at M and (ii) the overall service function β^p_M for M, both with respect to p. If $\beta_M \geq \beta^p_M$, then we mark v as a leaf node. Otherwise, we assign $\beta_{M,B}$ (resp. β_M) to the maximum of its current value and $\beta^p_{M,B}$ (resp. β^p_M), and proceed to explore the outgoing transitions from M as follows.

For each outgoing transition $\mathsf{tr} = (M, a, \varphi, [L', U'], M')$, we compute, for each input buffer B in $\mathcal{C}$, the carried-in backlog $\mathsf{bin}^{p'}_{M',B}$ of B at M' w.r.t. $p' = p \cup (M \xrightarrow{[L,U]} M')$, where $[L, U] = \mathsf{Inv}(M) \cap [L', U']$. We add $\langle M', p'\rangle$ into $\mathcal{S}$ if $\mathsf{bin}^{p'}_{M',B}$ satisfies φ for all input buffer B in $\mathcal{C}$. In this case, we also compute $\mathsf{cidf}^{p'}_{M',B}$ and $\mathsf{idbf}^{p'}_{M',B}$, and then assign $\mathsf{bin}_{M',B}$, $\mathsf{cidf}_{M',B}$ and $\mathsf{idbf}_{M',B}$ to be the maximum between their current values and the newly computed ones associated with p'. We additionally mark M' as a *reachable* mode.

c) Termination condition: The exploration process will stop when there is no more reachable modes to be explored, i.e., $\mathcal{S} = \emptyset$. Since the number of pending events in each buffer is upper bounded by the size of the buffer, the service function $\beta_{M'}$ of M' is always upper bounded. Further, as its newly computed value is always larger or equal to the previously computed one, $\beta_{M'}$ will reach a fixed point after a finite number of steps. In other words, the computation is always decidable.

The interface of $\mathcal{C}$: Recall $\mathcal{C} = (\mathcal{M}, M_{\mathsf{in}}, \mathsf{Inv}, \Phi, \Sigma, \mathrm{R}, \mathcal{T})$. The interface of $\mathcal{C}$ is the finite automaton $\textsc{inf}(\mathcal{C}) = (S, M_{\mathsf{in}}, \beta, \Sigma, \mathrm{R}')$ where S is the reachable modes of $\mathcal{C}$ and $\beta : S \to \mathcal{F}$. Each state M in S is associated with the service function $\beta(M)$ equal to the fixed point value of β_M computed above. Further, there is a transition $(M, a, [L, U], M')$ in R' iff there is a transition $(M, a, [L', U'], M')$ in R such that $[L, U] = \mathsf{Inv}(M) \cap [L', U']$.

As the exploration procedure computes the smallest fixed-point values of service functions that ensure schedulability and buffer constraints, the correctness of the computed interface follows directly from the soundness of the results established in Section IV-B and IV-C.

V. Interface composition

Consider a component $\mathcal{C}$ consisting of n components $\mathcal{C}_1, \ldots, \mathcal{C}_n$ that share the same resource using a scheduling policy SC. When SC is FP, we assume w.o.l.g. that $\mathcal{C}_i$ has higher priority than $\mathcal{C}_j$ if $i < j$. Let $\textsc{inf}(\mathcal{C}_i)$ be the resource interface of $\mathcal{C}_i$ for all $1 \leq i \leq n$. The resource interface of $\mathcal{C}$ is a composition of all $\textsc{inf}(\mathcal{C}_i)$ with respect to SC, given by

$$\textsc{inf}(\mathcal{C}) = \textsc{inf}(\mathcal{C}_n) \parallel_{\mathsf{SC}} \textsc{inf}(\mathcal{C}_{n-1}) \parallel_{\mathsf{SC}} \cdots \parallel_{\mathsf{SC}} \textsc{inf}(\mathcal{C}_1)$$

where $\parallel_{\mathsf{SC}}$ is defined as follows.

Let $\mathcal{A}_1 = (S_1, s_{\mathsf{in1}}, \beta_1, \Sigma_1, \mathrm{R}_1)$ and $\mathcal{A}_2 = (S_2, s_{\mathsf{in2}}, \beta_2, \Phi_2, \Sigma_2, \mathrm{R}_2)$ be two resource interfaces. The asynchronous composition of $\mathcal{A}_1$ and $\mathcal{A}_2$ under SC, denoted by $\mathcal{A}_1 \parallel_{\mathsf{SC}} \mathcal{A}_2$, is a state machine $\mathcal{A} = (S, s_{\mathsf{in}}, \beta, \Sigma, \mathrm{R})$ where:

- $S \subseteq S_1 \times S_2$ is the set of states.
- $s_{\mathsf{in}} = (s_{\mathsf{in1}}, s_{\mathsf{in2}})$ is the initial state of $\mathcal{A}$.
- $\beta : S \to \mathcal{F}$ is the service function associated with the states, defined by: For all $s = (s_1, s_2) \in S$,
$$\beta(s) = \begin{cases} \beta_1(s_1) + \beta_2(s_2), & \text{if } \mathsf{SC} \text{ is EDF} \\ \mathsf{Serv}\big(\beta_1(s_1), \beta_2(s_2)\big), & \text{if } \mathsf{SC} \text{ is FP} \end{cases}$$
- $\Sigma = \Sigma_1 \cup \Sigma_2$ is the set of service change signals.
- $\mathrm{R} \subseteq S \times \Sigma \times \mathsf{INT} \times S$ is the transition relation, defined as follows. For each $(s_1, a_1, [L, U], s'_1) \in \mathrm{R}_1$ and each $(s_2, a_2, [L', U'], s'_2) \in \mathrm{R}_2$:
  (i) $\langle (s_1, s'_1), a_1, I, (s_2, s'_2)\rangle$ is a transition in R if $a_1 = a_2$, $s_1 \neq s_2$ and $s'_1 \neq s'_2$ where $I = [\min(L, L'), \min(U, U')]$.
  (ii) $\langle (s_1, s'_1), a_1, [1, \min(U, U')], (s_2, s'_2)\rangle$ is a transition in R if $a_1 \neq a_2$, $s_1 \neq s_2$ and $s'_1 = s'_2$.
  (iii) $\langle (s_1, s'_1), a_2, [1, \min(U, U')], (s_2, s'_2)\rangle$ is a transition in R if $a_1 \neq a_2$, $s_1 = s_2$ and $s'_1 \neq s'_2$.

In the case of synchronous composition, only transitions in (i) are allowed and $I = [L, U] \cap [L', U']$. One can verify that $\beta(s)$ gives the minimum service that guarantees $\beta(s_1)$ and $\beta(s_2)$. Similarly, the interval associated with a transition indeed captures the time interval during which the corresponding transition(s) in $\mathcal{A}_1$ and $\mathcal{A}_2$ can be enabled.

As an example, consider the components $\mathcal{C}_3$ and $\mathcal{C}_4$ shown earlier in Fig. 4(b) and 4(c). The interface $\textsc{inf}(\mathcal{C}_3)$ ($\textsc{inf}(\mathcal{C}_4)$) has the same structure as that of $\mathcal{C}_3$ ($\mathcal{C}_4$), except that each of its states is associated with a service function and all buffer guards in the component are abstracted away. Interface of the composite component $\mathcal{C}_3 \parallel_{FP} \mathcal{C}_4$ is depicted in Fig. 5, which is obtained by composing $\textsc{inf}(\mathcal{C}_3)$ and $\textsc{inf}(\mathcal{C}_4)$ using our composition technique. Observe that by exposing communication between components on the interfaces, we are able to eliminate illegal composite states during the interface composition. For instance, the combination of ADM and 15fps is invalid in the composite component, which has been ruled out by the interface composition.

VI. Case Study

In this section, we present a case study of a smart networked embedded system that supports multiple concurrent adaptive streaming audio/video applications. We shall show through our case study how our compositional analysis framework can be used to model, analyze and optimize such adaptive systems.

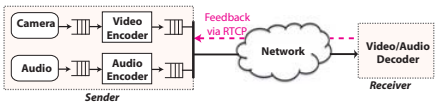

Fig. 9. An adaptive embedded networked system.

Fig. 9 depicts the overall architecture, consisting of two network-aware end systems sending/receiving data via a network. The sender (e.g., a video phone) captures the live video/audio, encodes it, and sends it over the network. The receiver, while receiving the data, provides feedback (regarding delay, packet loss, etc.) to the sender using a real-time transport control protocol (RTCP). Based on this feedback, the audio/video application managers adapt the audio and video sending rates accordingly. Here, we assume the audio application manager adapts its application to switch between three compression algorithms that have different bandwidths – PCM (Pulse Code Modulation) at 64 kb/s, ADM (Adaptive Delta Modulation) at 48 kb/s, and LPC (Linear Predictive Coding) at 4.8 kp/s – during run time depending on the network condition. Similarly, the video application manager adapts the video application to send at a lower frame rate in case the network is congested, and at a higher frame rate when the network is unloaded. The sender additionally runs other real-time critical system- and network-related tasks, as highlighted in Fig. 10. We shall focus on evaluating the sending system. Specifically, we shall estimate the minimum resource that must be guaranteed for the system to be schedulable. This is done by computing a resource interface for the system, using our multi-mode analysis and the unimodal techniques.

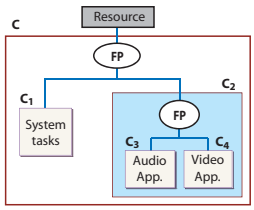

Fig. 10. Hierarchical scheduling of the sender.

As shown in Fig. 10, the sender employs a two-level hierarchical scheduling. It is partitioned into two components, $\mathcal{C}_1$ (system-related) and $\mathcal{C}_2$ (multimedia), which are scheduled using FP with $\mathcal{C}_1$ having higher priority than $\mathcal{C}_2$. Component $\mathcal{C}_2$ further comprises $\mathcal{C}_3$ (audio) and $\mathcal{C}_4$ (video) components, which are scheduled using FP with $\mathcal{C}_3$ having higher priority than $\mathcal{C}_4$. The execution semantics of $\mathcal{C}_1, \mathcal{C}_3$ and $\mathcal{C}_4$ are modeled by the MMA depicted in Fig. 4 (cf. Section IV-A).

Component $\mathcal{C}_1$ contains three tasks that are scheduled under EDF: T_k (kernel task), always executed; T_{ns} (network search task), initiated when a connection is lost; and T_{np} (network protocol task), executed to maintain a connected network.

Next, $\mathcal{C}_3$ (Fig. 4(b)) runs an audio encoding task (T_a) and an audio manager task T_{am}, where T_{am} always has higher priority than T_a. As mentioned above, T_a switches between three compression schemes, captured by three states of $\mathcal{C}_3$. Since the encoding time of an PLC task is an order of magnitude longer than that of PCM and ADM tasks, the audio application only selects PLC algorithm if the current backlog of the audio input buffer (buf_a) is no more than one third the buffer size (denoted by B in the figure). Further, as soon as the buffer is more than half filled and the application has been in PLC mode for more than D seconds, it switches back to PCM (i.e., the fastest encoding scheme).

Similarly, $\mathcal{C}_4$ shown in Fig. 4(c) consists of a video encoding task T_v and a video manager task T_{vm}, with T_{vm} having higher priority than T_v. Task T_v is assumed to change between two different sending rates, whereas T_{vm} remains unchanged.

We assume that the camera captures the video at the sampling rate equal to the playout rate, i.e., 15 fps and 7.5 fps. The deadline of T_v is set equal to its respective period, i.e., 66 ms (for 15 fps) and 133 ms (for 7.5 fps). Based on the sampling rate and the execution time of T_v for each of I, P, B frames [5], we compute the input arrival function (in terms of cycles) for T_v at each of the modes in $\mathcal{C}_4$. The audio task T_a is set to have a constant period and deadline of 20 ms. The ratio for its execution time at each of the modes PCM, ADM, and LPC are set to 1:13:110 [2]. All the remaining tasks are assumed to be periodic, with deadlines equal to periods.

Using unimodal techniques. Due to the adaptive characteristics of the system, unimodal modeling techniques are no longer capable of describing the system behavior precisely. We therefore resorted to approximating its worst-case workloads. We evaluated two approaches:

(U1) No consideration of the initial backlogs of the buffers when a mode change occurs.

(U2) Assuming the initial backlog is at most equal to the size of the buffers (as buffers must not overflow).

The hierarchical scheduling tree remains the same as in the multi-mode case; nevertheless, each of C_1, C_3 and C_4 is a set of tasks instead of an MMA. The timing parameters for T_a (T_v) are chosen to be the worst values in all its modes. For $\mathcal{C}_1$, since either T_k and T_n, or T_k and T_{ns} can run at any instant in time, we substitute T_n and T_{ns} with a single task whose workload equal to the maximum between that of T_n and T_{ns}. The rest of the tasks take their original (unchanged) values. It is worth noting that other approximating approaches might exist; however, we think that a more sophisticated and proven technique would require much effort which countervails the modeling convenience of unimodal models.

Analysis Results. The interface $\text{INF}(\mathcal{C}_2)$ of the multimedia component $\mathcal{C}_2$ is shown earlier in Fig. 5 (cf. Section IV-A). Each state of $\text{INF}(\mathcal{C}_2)$ corresponds to a valid combination of the audio and video components, and it is associated with a service function presenting the service requirements of $\mathcal{C}_2$ in a particular mode. Observe that the effects of internal triggered mode change events (i.e., buffer guards) in the original automata (cf. Fig. 4(b)) have been encapsulated in the computed service functions of the interface. On the other

hand, timing guards are exposed at the interface, which can be used when synchronizing with other interfaces at a higher level. The multi-mode interface $\text{INF}(\mathcal{C})$ for the entire system can be achieved by further composing $\text{INF}(\mathcal{C}_2)$ with $\text{INF}(\mathcal{C}_1)$. It has 8 states, corresponding to eight service levels required by the system depending on the traffic condition. We shall focus on the three highest service functions.

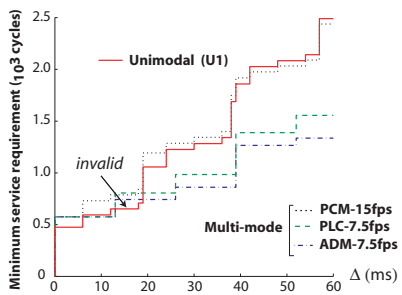

Fig. 11. Unimodal (U1) vs. multi-mode service requirements.

Fig. 11 shows the service function β_{U1}^{uni} computed using the unimodal approach (U1) in contrast to the three highest service functions of $\text{INF}(\mathcal{C})$. Note that β_{U1}^{uni} exhibits an unpredictable behavior where it crosses the service functions of $\text{INF}(\mathcal{C})$ at multiple points. Clearly, by ignoring mode change effects in the modeling, the unimodal technique fails to safely bound the service requirement at any mode of the system.

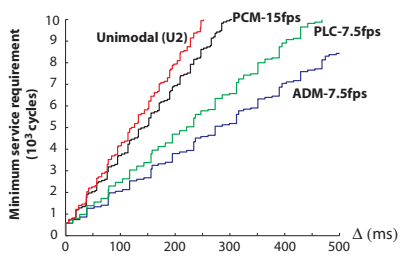

Fig. 12. Unimodal (U2) vs. multi-mode service requirements.

Now by taking into consideration the initial backlog due to a mode change, the second approach (U2) improves upon (U1) in terms of accuracy. As illustrated in Fig. 12, the computed service function β_{U2}^{uni} using (U2) falls above all the other service functions of the multi-mode interface. However, it is observed that β_{U2}^{uni} is very pessimistic. Its average long-term rate is 17%, 80%, and 134% higher than that of the three highest service levels of $\text{INF}(\mathcal{C})$.

The computed multi-mode interface also provides valuable insights to the system behavior, which can be used for optimizing the system. For instance, the service requirement when the system uses PCM-15 fps scheme is much higher (twice) than when it uses PLC-7.5 fps, even though PLC processing load is more than 100 times that of PCM (Fig. 12). This shows that when processing load is concerned, it is more effective to optimize video application instead of audio. Further, for a fixed video frame rate 7.5 fps, while switching from PCM to ADM increases only 3% processing load, switching from ADM to LPC increases 30% processing load. Thus, it is better to adapt from PCM to ADM than from ADM to LPC.

Finally, the multi-mode interface can easily be adapted to use in online algorithms such as dynamic frequency and voltage scaling. One may, for instance, derive a frequency corresponding to the long-term rate of each service function, and adapt between different frequencies based on the same condition as the guard between different states.

VII. Concluding Remarks

We have proposed a multi-mode automata model and an interface-theoretic technique to enable compositional analysis and correct-by-construction design of multi-mode systems. Our results extend existing work in two dimensions: from *performance analysis* to *compositional analysis* of multi-mode systems, and from compositional analysis of *unimodal* to *multi-mode* models. The applicability and benefits of our proposed technique have been demonstrated in a smart networked streaming system.

It would be interesting to investigate abstraction techniques for refining the multi-mode interface computed from an MMA or a composition of interfaces. One potential direction would be to abstract states that share similar service functions, and transitions that are triggered by a common set of events to limit the size of an interface without sacrificing accuracy.

References

[1] K. Albers, F. Bodmann, and F. Slomka. Advanced hierarchical event-stream model. In *ECRTS*, 2008.

[2] J.-C. Bolot and A. Vega-Garcia. Control mechanisms for packet audio in the internet. In *INFOCOM*, 1996.

[3] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *RTSS*, 1999.

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

[5] S. Chakraborty, T. Mitra, A. Roychoudhury, and L. Thiele. Cache-aware timing analysis of streaming applications. *Real-Time Systems*, 41(1):52–85, 2009.

[6] A. Easwaran, M. Anand, and I. Lee. Compositional analysis framework using edp resource models. In *RTSS*, 2007.

[7] A. Easwaran, I. Shin, O. Sokolsky, and I. Lee. Incremental schedulability analysis of hierarchical real-time components. In *EMSOFT*, 2006.

[8] G. Fohler. Changing operational modes in the context of pre run-time scheduling. *IEICE Transactions on Information and Systems*, E76-D(11):1333–1340, 1993.

[9] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo. Adaptive power management for real-time event streams. In *ASP-DAC*, 2010.

[10] L. T.X. Phan, I. Lee, and O. Sokolsky. Compositional analysis of multi-mode systems. http://www.cis.upenn.edu/~linhphan/papers/ecrtsTR.pdf.

[11] L.T.X. Phan, S. Chakraborty, and I. Lee. Timing analysis of mixed time/event-triggered multi-mode systems. In *RTSS*, 2009.

[12] L.T.X. Phan, S. Chakraborty, and P.S. Thiagarajan. A multi-mode real-time calculus. In *RTSS*, 2008.

[13] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26:161–197, 2004.

[14] L. Sha, R. Rajkumar, J. Lehoczsky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):244–264, 1989.

[15] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *ECRTS*, 2008.

[16] I. Shin and I. Lee. Compositional real-time scheduling framework. In *RTSS*, 2004.

[17] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems*, 7(3):1–39, 2008.

[18] N. Stoimenov, S. Perathoner, and L. Thiele. Reliable mode changes in real-time systems with fixed priority or edf scheduling. In *DATE*, 2009.

[19] E. Wandeler and L. Thiele. Real-time interfaces for interface-based design of real-time systems with xed priority scheduling. In *EMSOFT*, 2005.