8-1-2011

Towards a Compositional Multi-Modal Framework for Adaptive Cyber-Physical Systems

Linh T.X. Phan
University of Pennsylvania, linhphan@cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

1st International Workshop on Cyber-Physical Systems, Networks, and Applications (CPSNA), Toyama, Japan, Aug 2011 (Invited paper).

Towards a Compositional Multi-Modal Framework for Adaptive Cyber-Physical Systems

Abstract
Among the key characteristics of cyber-physical systems are the ability to adapt to changes during operation, the multidimensional complexity of multi-functionality and the underlying heterogeneous distributed architecture, as well as resource use efficiency. In this paper, we propose a compositional multi-modal approach to modeling, analyzing, and designing such systems. We introduce a general framework for modeling and compositional analysis of multi-mode systems on a distributed architecture that facilitates adaptivity, efficient use of resources, and incremental integration. We present some preliminary results, and we describe some of the remaining challenges and future directions.

Disciplines
Computer Sciences | Physical Sciences and Mathematics

Comments
1st International Workshop on Cyber-Physical Systems, Networks, and Applications (CPSNA), Toyama, Japan, Aug 2011 (Invited paper).

Towards a Compositional Multi-Modal Framework for Adaptive Cyber-Physical Systems

Linh T.X. Phan Insup Lee
University of Pennsylvania
{linhphan,lee}@cis.upenn.edu

***Abstract*—Among the key characteristics of cyber-physical systems are the ability to adapt to changes during operation, the multidimensional complexity of multi-functionality and the underlying heterogeneous distributed architecture, as well as resource use efficiency. In this paper, we propose a compositional multi-modal approach to modeling, analyzing, and designing such systems. We introduce a general framework for modeling and compositional analysis of multi-mode systems on a distributed architecture that facilitates adaptivity, efficient use of resources, and incremental integration. We present some preliminary results, and we describe some of the remaining challenges and future directions.**

I. Introduction

An essential yet challenging characteristic of cyber-physical systems is their ability to respond and adapt quickly to changes during operation, such as hardware/software defects, resource changes, and non-continual feature usage. For example, an unmanned aircraft avionics system must adapt its configuration to collisions or aircraft system failures in order to allow continued safe operation, and an adaptive cruise control system must adapt its operating speed according to the current road condition in order to avoid an accident. Such adaptive behaviors can naturally be captured using a multi-mode modeling formalism. In this formalism, the system operates in multiple *modes*, which correspond to system configurations or modes of operation. Each mode can be characterized by a unique set of tasks, resource configurations, and scheduling policies. *Mode switches*, or mode changes, reflect changes in the system or in the environment, which can be time-triggered (e.g., by a periodic timer interrupt) or event-triggered (e.g., by a detected collision). For instance, an adaptive cruise control (ACC) system in a car can be modeled as a multi-mode system that consists of two modes (among others): (i) the Speed Control mode, in which the ACC system sets the car's speed to a predefined speed value, and (ii) the Time Gap Control mode, in which the ACC system dynamically adjusts the car's speed to maintain a minimum clearance distance between the car and other leading vehicles. The system switches between these two modes depending on whether there are slower-moving vehicles in front of the car [21].

However, applications of multi-mode extend beyond a modeling and design technique for adaptivity. Recent advances in microprocessor technology have allowed embedded systems manufacturers to pack more functionality into a single chip, thereby reducing device cost. A modern smartphone is now capable of running a host of applications, such as GPS localization, video decoding, voice processing, and email and web access. Inherently, such a system is required to operate in multiple modes; in each mode, a subset of tasks of these applications is executed, and mode switches may be caused by events such as an incoming phone call or an activation of the GPS algorithm. Although it is possible to model such a system using the traditional unimodal approach, the unimodal abstraction of the above multi-modal behavior does not only result in significant pessimism and resource over-dimensioning but may potentially lead to invalid analysis results [18]. Multi-mode modeling and analysis techniques provide a tighter abstraction of the system, thus improving resource use efficiency.

In fact, multi-modal behavior manifests itself across all layers of cyber-physical systems, ranging from individual tasks (e.g., control tasks in switched systems), to applications (e.g., an adaptive cruise control), processing nodes (e.g., a dynamic voltage scaling (DVS) server), and entire heterogeneous architectures (e.g., a fault-tolerant system where each mode consists of a set of active nodes and a network configuration). This characteristic makes multi-mode an attractive approach for modeling, analyzing, and designing cyber-physical systems – one that not only reduces the abstraction overheads incurred by the conventional unimodal approach but is also necessary to achieve adaptability and resource use efficiency.

In this paper, we propose a compositional multi-modal framework for adaptive cyber-physical systems that combines the above benefits of a multi-mode approach with the benefits of a compositional analysis, such as a lower analytical complexity. In this framework, a system is modeled as several multi-mode components, and its compositional analysis is done by means of multi-mode interfaces and interface composition. Such a framework inherently requires a new set of models and theories, and inevitably introduces a host of new challenges. We present preliminary solutions for some of these challenges, and we highlight various open issues and promising future directions.

II. Related Work

At the task level, several task models and schedulability techniques have been developed to support variable computation times or execution periods, which is a type of mode change (e.g., [3], [5], [6], [11], [19]). At the system level, multi-modal operation is usually modeled by an automaton, whose states represent operating modes of the system [10], [14], [18],

[24], [26]. Multi-mode techniques have also been explored at the architecture level, where a mode corresponds to a configuration. Each mode is a set of active resources, a local schedule on each resource, and a network configuration [16]. A multi-mode approach has also been used to achieve efficient use of resources; for instance, dynamic resource allocation techniques, such as adaptive servers [1], [2], [8], [9], have been developed for this purpose. Similarly, energy-efficient task allocation techniques that consider mode execution patterns have been proposed (e.g., [12], [13], [23]).

The analysis techniques in the above work have two main themes: (i) given a multi-mode model, compute performance-related metrics such as end-to-end delay, maximum buffer requirement, or schedulability [18]; and (ii) compute suitable parameters for all tasks, so that the system is schedulable [24]. None of them addresses the compositional analysis of multi-mode systems, however. Recently, we have developed a compositional analysis technique for multi-mode systems that are executed on a single resource [20]. The framework proposed here extends this technique to a distributed setting.

Another major line of multi-mode research in the real-time systems domain is to design mode change protocols. During a mode transition, the system may need to process both pending tasks of the current mode and incoming tasks of the new mode. This co-execution could lead to a temporal overload that causes some tasks to miss their deadlines. A mode change protocol specifies how tasks are executed during a mode transition to avoid overloading the new mode, thereby guaranteeing schedulability during the transition. Different protocols have been developed (see e.g., [4], [14], [15], [17], [25]–[28]) and classified in [22]. Their sole objective is to ensure schedulability; hence, they must be significantly extended to be applicable in our setting. In our prior work, we have developed a mode change protocol model that generalizes these existing protocols, as well as an associated technique for checking whether a protocol is feasible for a multi-mode system [21]. The framework proposed in this paper uses these mode change protocol models and analysis techniques as a basic building block.

III. A Compositional Multi-modal Framework

In this section, we describe a general framework for the compositional analysis of multi-mode systems. The setting we consider consists of a set of multi-mode applications executing on a distributed heterogeneous platform as described below.

A. System Description

Each application consists of a finite set of disjoint modes, where each mode is characterized by a finite set of tasks that process input streams and produce output streams. Each task has an input buffer that stores its input stream and an output buffer that stores its output stream. Mode switches may be triggered by a timing constraint, an external event from the environment, and/or a condition on the state of the buffers. Each application is associated with a mode change requirement that gives the constraints on the behavior of the task execution during a mode switch. These applications are mapped onto an architectural platform according to a given task mapping.

The platform consists of an interconnected set of distributed heterogeneous resources, where each resource is either a processing element (e.g., an ECU) or a shared communication network (e.g., a CAN bus). To achieve timing separation, each resource schedules the mapped (tasks of the) applications in a hierarchical manner, forming a tree of components as illustrated in Fig. 2. In this figure, each leaf component $\mathcal{C}_{i,j}$ contains the subset of tasks of the application i that are mapped on the resource j. Each intermediate component represents a composition of the child components. Each component in the tree has a local scheduling policy under which its subcomponents (tasks) are scheduled.

We have two objectives in this setting: (1) to analyze the performance-related metrics (e.g., schedulability, end-to-end delay, buffer requirements) of the applications, and (2) to dimension the resource requirements for the platform to meet a given set of performance-related constraints. To achieve efficiency and to support open systems, the methods we aim to develop should preserve compositionality and enable incremental analysis.

B. Multi-Mode Modeling and Compositional Analysis

Fig. 1 shows the basic blocks in the component modeling stage. As shown in the figure, we first model each application App_i as a *multi-mode automaton* MMA_i and the mode-change requirements as a *mode change protocol* P_i. The *Model Extraction* block takes as inputs a multi-mode automaton MMA_i and the task mapping, and it extracts for each resource j a sub-automaton A_i^j. The sub-automaton A_i^j is made of the modes of MMA_i that contain only the tasks that are mapped onto the resource j.

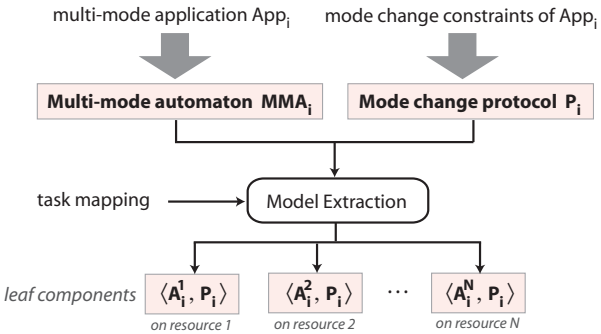

Fig. 1. Modeling multi-mode components.

The extracted sub-automatons A_1^j to $A_{m_j}^j$ of the applications that are scheduled on each resource j, together with the respective mode change protocols P_1 to P_{m_j}, constitute the leaf components in the scheduling hierarchy of the resource j. The complete system model is a composition of components $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_N$ representing the N resources communicating via shared buffers, where each component $\mathcal{C}_j$ is again a hierarchical composition of the components $\mathcal{C}_{1,j} = \left\langle A_1^j, P_1 \right\rangle$, $\mathcal{C}_{2,j} = \left\langle A_2^j, P_2 \right\rangle$, …, and $\mathcal{C}_{m_j,j} = \left\langle A_{m_j}^j, P_{m_j} \right\rangle$. Fig. 2 gives an example of a component C_j, corresponding to resource j,

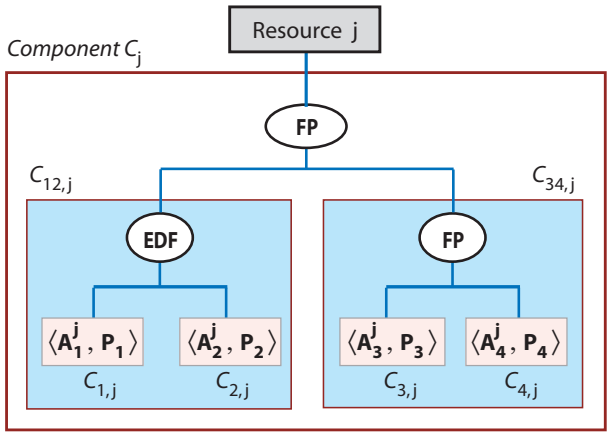

Fig. 2. An example component of a resource j.

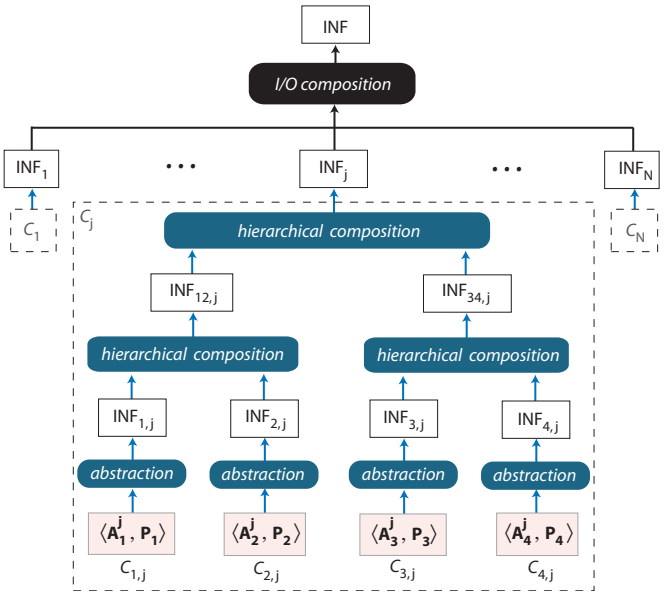

Fig. 3. Compositional analysis via interface abstraction and composition.

that is composed of four multi-mode applications scheduled hierarchically.

The compositional analysis of the obtained system model is done by means of component abstractions (interfaces) and interface composition. As described above, a component in our setting can be a multi-mode automaton of an application's tasks mapped on a resource and its associated change protocol (e.g., $\mathcal{C}_{1,j}$), a hierarchical composition of different multi-mode components that share the same resource (e.g., $\mathcal{C}_j$), or a composition of multiple such hierarchical components (e.g., composition of $\mathcal{C}_1$ and $\mathcal{C}_2$). We distinguish two types of composition: (i) *hierarchical composition*, which is a composition of different components sharing the same resource under a scheduling policy (e.g., composition of $\mathcal{C}_{1,j}$ and $\mathcal{C}_{2,j}$ under EDF scheduling in Fig. 2); and (ii) *I/O composition*, which is a composition of different components communicating via shared buffers (e.g., composition of two connected components $\mathcal{C}_j$ and $\mathcal{C}_k$, where the output streams from $\mathcal{C}_j$ are the input streams to $\mathcal{C}_k$).

An interface of a component encapsulates the total resource requirement of the component's tasks and the arrival patterns of the component's input and output streams. A composition of the interfaces of components $\mathcal{C}$ and $\mathcal{C}'$ computes a composed interface from the two interfaces; this composed interface captures the resource requirements and the arrival functions of the input and output streams of the composition of $\mathcal{C}$ and $\mathcal{C}'$. There are two types of interface composition that correspond to the two types of component composition, i.e., hierarchical composition under a scheduling policy and I/O composition.

We analyze the system component-wise as illustrated in Fig. 3. At each component $\mathcal{C}_j$, we first abstract each leaf component $\mathcal{C}_{i,j}$ of the scheduling hierarchy as an interface $\text{INF}_{i,j}$. We then hierarchically compose these interfaces bottom-up until we reach the root of the tree, where we obtain an interface INF_j of the component $\mathcal{C}_j$. The interface INF of the complete system is then obtained by composing these computed interfaces INF_j using I/O interface composition.

By examining the system interface INF, we can then derive various performance-related properties of the system with respect to a priori resource configuration (i.e., the processing speeds of the processing elements and network communication bandwidths). Likewise, the resource requirements exposed on the interface of the system also allow us to dimension the resources required by the applications.

In the next section, we describe the basic blocks of the modeling and compositional analysis, including models for multi-mode components, mode change protocols, and multi-mode interfaces. We then briefly describe a method for interface generation and interface composition that generalizes our preliminary results in [18], [20] and [21].

IV. Multi-Mode Models and Interface Techniques

A. Multi-Mode Component Model

Our multi-mode component model generalizes the multi-mode component model developed in [20] and [21] to enable composition with external components in a distributed architecture. Towards this, we extend the task model in [21] to include an output buffer and bounds on the output stream. Unlike in [20] where a fixed mode change semantics is enforced as part of the semantics of the multi-mode automaton, in this framework, we separate the semantics of the multi-mode automaton from the mode change semantics to allow for any general mode change behavior. Thus, a multi-mode component is a pair of multi-mode automaton and an associated mode change protocol.

Tasks and input events. The tasks of the applications process input event streams and produce output event streams. Each task has a finite input buffer and a finite output buffer that store the input and output event stream, respectively. Whenever an event arrives at the buffer of a task, an instance of the task is released to process the event. Events in the same buffer are processed sequentially in the order of their arrivals. Each task T is characterized by a tuple $T = (B, B', E, D, \alpha, \alpha', \pi)$, where B is the input buffer, B' the output buffer, E the worst-case execution demand, D the relative deadline, α the

arrival function of the input stream, α' the arrival function of the output stream, and π an additional scheduling parameter. Examples of the scheduling parameter π include the priority of a task under a Fixed Priority (FP) scheduling policy or a slot size under TDMA scheduling policy. An example of a task and its parameters is shown in Fig. 4.

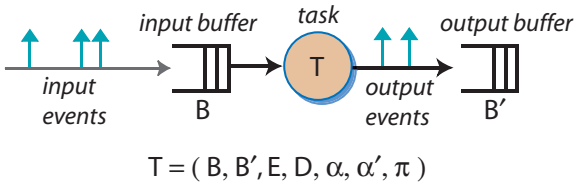

Fig. 4. A task and its parameters.

The *arrival function* α of an event stream is defined as in the standard Real-Time Calculus technique [7]. That is, $\alpha = (\alpha^u, \alpha^l)$, where $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ specify the maximum and minimum number of events that can arrive over any time interval of length Δ, respectively, for all $\Delta \in \mathbb{N}$. The arrival function α of T captures all the acceptable arrival patterns of the input events of T, which are also the release patterns of the instances of T.

Similarly, the output arrival function α' captures the bounds on the arrival patterns of the output stream after being processed by T. Given an input arrival function α, one can compute the corresponding output arrival function α' based on the timing property of T and the resource allocated to T, and vice versa [7]. The processing requirement of an event in the buffer of T is described in the same manner as in [21].

Resource. The availability of a resource (e.g., processing element or a communication network) is modeled in the same way as in [20], i.e., by a *service function* $\beta(\Delta)$ that specifies the minimum number of execution units (e.g., processor cycles, communication bandwidth) that can be provided by the resource over any time interval of length Δ.

Multi-mode automata. The behavior of a multi-mode application mapped on a resource is modeled by a multi-mode automaton, which is a finite state machine whose states represent operating modes and whose transitions represent mode changes. Each state of the automaton specifies the set of tasks that are active and the scheduling policy when the system is executing at the corresponding mode. The guard associated with a transition specifies a mode change triggering event. The multi-mode automaton model is the same model used in [20], except that it also includes not only a set of input buffers $\mathcal{B}$ of the tasks in the automaton but also the set of output buffers $\mathcal{B}'$.

Fig. 5 shows the multi-mode automaton of an application consisting of four tasks T_1, T_2, T_2' and T_3 that process input events from the buffers B_1, B_2 and B_3, where $T_1 = (B_1, B_1', 2, 5, \alpha_1, \alpha_1', 1)$, $T_2 = (B_2, B_2', 3, 7, \alpha_2, \alpha_2', 2)$, $T_2' = (B_2, B_2', 3, 14, \alpha_2, \alpha_2', 2)$ and $T_3 = (B_3, B_4', 4, 20, \alpha_3, \alpha_3', 0)$. Here, T_2' is a modified task of T_2.

As shown in the figure, the system is initially in mode M_1, in which T_1 and T_2 are active and scheduled under FP,

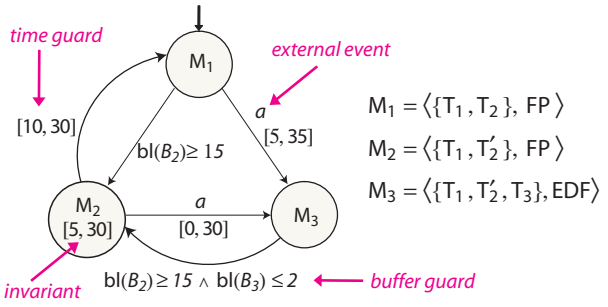

Fig. 5. A multi-mode automaton.

and in which T_1 has a higher priority than T_2. When the system is in M_1 and the input buffer B_2 contains more than 15 events (denoted by "$\mathsf{bl}(B_2) \geq 15$"), the system will move to mode M_2. In M_2, the task T_2 is changed into T_2', which has a deadline that is twice that of T_2. The unchanged task T_1 and the modified task T_2' are again scheduled under FP with the same priority order. The system will stay in M_2 for at least 10 and at most 30 time units before it moves back to M_1. However, if the signal a arrives during this time, the system moves to M_3, in which the task T_3 becomes active and will be scheduled together with T_1' and T_2 under EDF. The rest can be explained accordingly.

Mode change protocol. A mode change protocol describes the execution behavior of a multi-mode model during a transition from one mode to another, i.e., from the instant a transition is enabled until the instant all the new attributes associated with the destination mode are in effect. In this paper, the mode change behavior of a multi-mode application is captured by a mode change protocol model [21].

Multi-mode components. A multi-mode component is a pair $\langle A, P \rangle$, where A is the multi-mode automaton of the application and P is its associated mode change protocol. For example, the multi-mode component $\mathcal{C}_{1,j}$ in Fig. 2 is made of the multi-mode automaton A_1^j and the mode change protocol P_1. A composition of two multi-mode components $\langle A_1, P_1 \rangle$ and $\langle A_2, P_2 \rangle$ is a multi-mode component $\langle A, P \rangle$, where (i) A is the composition of A_1 and A_2, and (ii) P is the composition of P_1 and P_2. A composition of two multi-mode interfaces and of two mode change protocols can be computed using the same composition technique as in [20] and in [21], respectively. We require that P is always feasible for A for any component $\langle A, P \rangle$, i.e., the mode change behavior of A enforced by P guarantees schedulability and no overflow constraint (which can be verified using the feasibility analysis technique in [21]).

B. Multi-Mode Interfaces

An interface of a component encapsulates the total resource requirement of the component's tasks, as well as the bounds on the arrival patterns of the component's input and output streams. The interface of a multi-mode component $\mathcal{C}$ can be captured by a *multi-mode resource interface*, which is a finite state machine in which each state is augmented with (i) a minimum service function that is demanded by $\mathcal{C}$, (ii) an

arrival function of each input stream to the component, and (iii) an arrival function of each output stream generated by the component. In addition, the interface also contains a set of input buffers and output buffers that are connected to external components. The different service functions associated with different states of the interface represent the different resource requirements of $\mathcal{C}$ when it is in different modes (or sets of modes). Similarly, the input/output arrival functions associated with each state represent different arrival patterns of the incoming/outgoing streams to/from the component. We note that the multi-mode interfaces proposed here extend the interfaces proposed in [20] to include information about the input and output streams, which is needed for composing interfaces of components that communicate via shared buffers.

Definition 1 (Multi-mode Resource Interface)**.** *A resource interface of a multi-mode component $\mathcal{C} = \langle A, P\rangle$ with multi-mode automaton A and mode change protocol P is a finite state machine* $\text{INF}(\mathcal{C}) = (S, s_{\text{in}}, \Sigma, R, \beta, \mathcal{B}, \mathcal{B}', \mathsf{Arr}, \mathsf{Arr}')$ *where*

- *S is a finite set of states, each of which characterizes the resource requirement of one or more modes in A.*
- *$s_{\text{in}} \in S$ is the initial state.*
- *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.*
- *β 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 A to be schedulable.*
- *$\mathcal{B}$ is the set of input buffers of A that store input streams from external components.*
- *$\mathcal{B}'$ is the set of output buffers of A that store output streams to external components.*
- *Arr is the input arrival mapping, which specifies for each state $s \in S$ and each input buffer $b \in \mathcal{B}$ an arrival function $\mathsf{Arr}(s, b)$ that bounds the arrival patterns of the incoming stream to b when the system is at s.*
- *Arr' is the output arrival mapping, which specifies for each state $s \in S$ and each output buffer $b' \in \mathcal{B}'$ an arrival function $\mathsf{Arr}'(s, b')$ that bounds the arrival patterns of the outgoing stream to b' when the system is at s.*

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

C. Interface Generation

A resource interface $\text{INF}(\mathcal{C})$ of a multi-mode component $\mathcal{C} = \langle A, P\rangle$ can be computed by first computing the behavioral automaton of A with respect to protocol P using the technique in [21] and applying the interface generation for the the behavioral automaton as outlined in [20]. The only addition here is the computation of the input and output arrival mappings.

We note that by following the technique in [20], each state s of the interface corresponds to a set of modes in $\mathcal{C}$, denoted by M_s. For any given $b \in \mathcal{B}$, let T be the task in $\mathcal{C}$ whose input buffer is b. Then, for all state s of the interface, $\mathsf{Arr}(s, b)$ is the envelope of all input arrival functions of T at the modes in M_s. That is, $\mathsf{Arr}(s, b) = (\alpha^u_{\max}, \alpha^l_{\min})$, where $\alpha^u_{\max}$ is the maximum of all upper input arrival functions $\alpha^u_{T,m}$ and $\alpha^l_{\min}$ is the minimum of all lower input arrival functions $\alpha^l_{T,m}$ of T at mode m for all $m \in M_s$.

Similarly, $b' \in \mathcal{B}$, let T' be the task in $\mathcal{C}$ whose output buffer is b'. Then, $\mathsf{Arr}(s, b')$ is the envelope of all output arrival functions of T' at the modes in M_s. The output arrival function of a task T at a mode in $\mathcal{C}$ can be computed from its input arrival function by exploring the behavioral automaton of A with respect to P in the same manner as done in [18]. The output function can be represented symbolically based on the variable input function, if the input function is unknown.

D. Interface Composition

Let $\text{INF}_1 = (S_1, s_{\text{in}1}, \Sigma_1, \mathrm{R}_1, \beta_1, \mathcal{B}_1, \mathcal{B}'_1, \mathsf{Arr}_1, \mathsf{Arr}'_1)$ and $\text{INF}_2 = (S_2, s_{\text{in}2}, \Sigma_2, \mathrm{R}_2, \beta_2, \mathcal{B}_2, \mathcal{B}'_2, \mathsf{Arr}_2, \mathsf{Arr}'_2)$ be two multi-mode interfaces.

The hierarchical composition of INF_1 and INF_2 that share the same resource under a scheduling policy SC is the interface $\text{INF}_{\mathsf{SC}} = (S, s_{\text{in}}, \Sigma, \mathrm{R}, \beta, \mathcal{B}, \mathcal{B}', \mathsf{Arr}, \mathsf{Arr}')$ where

- $S \subseteq S_1 \times S_2$ is the set of reachable states (with respect to R).
- $s_{\text{in}} = (s_{\text{in}1}, s_{\text{in}2})$ is the initial state of INF.
- $\Sigma = \Sigma_1 \cup \Sigma_2$ is the set of service change signals.
- $\mathrm{R} \subseteq S \times \Sigma \times \mathsf{INT} \times S$ is the transition relation, which is defined to be the same as in the interface composition in [20].
- β is the service function associated with the states in S, defined by: For all $s = (s_1, s_2) \in S$,

$$\beta(s) = \begin{cases} \beta_1(s_1) + \beta_2(s_2), & \text{if SC is EDF} \\ \mathsf{Serv}\big(\beta_1(s_1), \beta_2(s_2)\big), & \text{if SC is FP} \end{cases}$$

  where $\mathsf{Serv}(f, g) = f(\Delta - \lambda) + g(\Delta - \lambda)$ where $\lambda = \sup\{\epsilon \mid f(\Delta - \epsilon) = f(\Delta)\}$.
- $\mathcal{B} = (\mathcal{B}_1 \cup \mathcal{B}_2) \backslash (\mathcal{B}'_1 \cup \mathcal{B}'_2)$ is the set of input buffers.
- $\mathcal{B}' = (\mathcal{B}'_1 \cup \mathcal{B}'_2) \backslash (\mathcal{B}_1 \cup \mathcal{B}_2)$ is the set of output buffers.
- Arr is the input arrival mapping, where $\mathsf{Arr}(s, b) = \mathsf{Arr}_1(s_1, b)$ if $b \in \mathcal{B}_1$ and $\mathsf{Arr}(s, b) = \mathsf{Arr}_2(s_2, b)$ if $b \in \mathcal{B}_2$ for all $s = (s_1, s_2) \in S$.
- Arr' is the output arrival mapping, where $\mathsf{Arr}'(s, b') = \mathsf{Arr}_1(s_1, b')$ if $b' \in \mathcal{B}'_1$ and $\mathsf{Arr}(s, b) = \mathsf{Arr}_2(s_2, b')$ if $b' \in \mathcal{B}'_2$ for all $s = (s_1, s_2) \in S$.

The I/O composition of INF_1 and INF_2 that communicate via shared input/output buffers is the interface $\text{INF}_{I/O} = (S, s_{\text{in}}, \Sigma, \mathrm{R}, \beta, \mathcal{B}, \mathcal{B}', \mathsf{Arr}, \mathsf{Arr}')$, which is defined the same as INF_{SC}, except that the service function β is given by $\beta(s) = \big(\beta_1(s_1), \beta_2(s_2)\big)$ for all $s = (s_1, s_2) \in S$.

Two interfaces INF_1 and INF_2 are compatible iff for all $s = (s_1, s_2) \in S$: (i) for all $b \in \mathcal{B}'_1 \cap \mathcal{B}_2$, $\mathsf{Arr}'_1(s_1, b) \models \mathsf{Arr}_2(s_2, b)$, and (ii) for all $b \in \mathcal{B}'_2 \cap \mathcal{B}_1$, $\mathsf{Arr}'_2(s_2, b) \models \mathsf{Arr}_1(s_1, b)$. We say that $\alpha_1 \models \alpha_2$ iff $\alpha^u_1 \leq \alpha^u_2$ and $\alpha^l_1 \geq \alpha^l_2$ where $\alpha_1 = (\alpha^u_1, \alpha^l_1)$ and $\alpha_2 = (\alpha^u_2, \alpha^l_2)$. We assume that interface compositions are only defined for compatible interfaces.

V. Discussion

In this paper, we introduce an approach towards a compositional multi-modal framework for modeling, analyzing, and designing of adaptive cyber-physical systems. Since fundamental principles of multi-modality co-exist in control systems, referred to as *switched systems*, such a multi-modal approach can serve as a basis for the co-design of distributed embedded architectures and adaptive controllers for cyber-physical systems. We outline the key elements and preliminary technical results for realizing this compositional multi-modal framework. In this section, we highlight some of the open issues and future directions.

A key challenge inherent in any compositional analysis framework is how to balance the trade-off between interface's complexity and interface's accuracy, which is made more difficult in the context of multi-mode systems. The multi-mode interface techniques presented in Section IV allow for the detection of incompatibilities in communication and resource use of components in a composite system during interface composition, thereby reducing abstraction overhead. It can be observed, however, that the number of states in the interface composition of sub-interfaces in the worst-case is proportional to the product of the numbers of states of the sub-interfaces. As a result, the interface composition's complexity increases as the number of modes within components and the number of components increase. An opportunity here is to explore existing automata abstraction techniques in abstracting similar service functions (representing resource requirement) and similar states/transitions that exhibit similar timing behaviors to achieve more succinct interfaces without sacrificing accuracy.

The interface generation described in Section IV-C is based on the approach in [20], which assumes that all tasks within a multi-mode automaton are independent. As such, the algorithm needs to be modified to incorporate the possible data dependencies between different tasks of the multi-mode automaton that share input/output buffers. One approach is to first compute the output arrival function of a preceding task and substitute it as input to a dependent task. A challenge here is to ensure that the mode-change effect propagated from one task to another converges after a (small) finite number of steps.

We note also that the interface generation in [20] assumes that all input arrival functions of the component's tasks are known a priori, which may not hold in a distributed setting. For example, consider a setting as illustrated in Fig. 3 where the component $\mathcal{C}_j$ represents a communication bus that transmits output messages from component $\mathcal{C}_1$ to component $\mathcal{C}_N$. In this case, the input functions of the messages to $\mathcal{C}_j$ are often not specified but computed based on the generating component $\mathcal{C}_1$. Consequently, the interface composition either needs to follow the data flow or the composed interface has to be represented symbolically based on the variable input functions of the underlying components. In the former case, interface composition is not associative, which makes incremental analysis difficult. In the latter case, the interface inherently becomes complex and a closed-form solution for the function β associated with each state of the interface is needed (as an alternative to the current computation using an iterative procedure [20]).

Lastly, the composition analysis presented in this paper as well as all existing multi-mode results have been limited to uniprocessor processing elements and EDF/FP scheduling policies. It would be interesting yet challenging to extend the interface theories to multi-core settings and more complex communication mechanisms such as the FlexRay in automotive architectures.

References

[1] L. Abeni and G. Buttazzo. Adaptive bandwidth reservation for multimedia computing. In *Proc. of the International Conference on Real-Time Computing Systems and Applications (RTCSA)*, 1999.

[2] L. Abeni and G. Buttazzo. Hierarchical QoS management for time sensitive applications. In *Proc. of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, 2001.

[3] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. TIMES: a tool for schedulability analysis and code generation of real-time systems. In *Proc. of the International Conference on Formal Modeling and Analysis of Timed Systems*, 2003.

[4] D. Bertrand, A.-M. Déplanche, S. Faucou, and O. H. Roux. A study of the AADL mode change protocol. In *Proc. of the IEEE International Conference on the Engineering of Complex Computer Systems (ICECCS)*, 2008.

[5] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 1999.

[6] G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. *IEEE Transactions on Computers*, 51(3):289–302, 2002.

[7] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *Proc. of the 2003 Design, Automation and Test in Europe Conference and Exposition (DATE)*, 2003.

[8] S. Craciunas, C. Kirsch, H. Payer, H. Rock, and A. Sokolova. Programmable temporal isolation through variable-bandwidth servers. In *Proc. of the IEEE International Symposium on Industrial Embedded Systems (SIES)*, 2009.

[9] A. B. de Oliveira, E. Camponogara, and G. Lima. Dynamic reconguration in reservation-based scheduling: An optimization approach. In *Proc. of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, 2009.

[10] P. Feiler, B. Lewis, and S. Vestal. The SAE AADL standard: A basis for model-based architecture-driven embedded systems engineering. In *Workshop on Model-Driven Embedded Systems*, 2003.

[11] S. Goddard and X. Liu. A variable rate execution model. In *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 135–143, 2004.

[12] L. Huang and Q. Xu. Energy-efficient task allocation and scheduling for multi-mode MPSoCs under lifetime reliability constraint. In *Proc. of the Design, Automation Test in Europe Conference Exhibition (DATE)*, 2010.

[13] V. Kianzad and S. S. Bhattacharyya. CHARMED: A multi-objective co-synthesis framework for multi-mode embedded systems. In *Proc. of the 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, 2004.

[14] F. Maraninchi and Y. Rémond. Mode-automata: a new domain-specific construct for the development of safe critical systems. *Science of Computer Programming*, 46(3):219–254, 2003.

[15] P. Martins and A. Burns. On the meaning of modes in uniprocessor real-time systems. In *SAC*, pages 324–325, 2008.

[16] M. Mitzlaff, R. Kapitza, and W. Schröder-Preikschat. Enabling mode changes in a distributed automotive system. In *Proc. of the 1st Workshop on Critical Automotive applications: Robustness and Safety*, 2010.

[17] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 172–179, 1998.

[18] L. Phan, S. Chakraborty, and I. Lee. Timing analysis of mixed time/event-triggered multi-mode systems. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 2009.

[19] L. Phan, S. Chakraborty, and P. Thiagarajan. A multi-mode real-time calculus. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, 2008.

[20] L. Phan, I. Lee, and O. Sokolsky. Compositional analysis of multi-mode systems. In *Proc. of the Euromicro Conference on Real-Time Systems (ECRTS)*, 2010.

[21] L. Phan, I. Lee, and O. Sokolsky. A formal semantic framework for multi-mode systems. In *Proc. of the IEEE Real-Time Technology and Applications Symposium (RTAS)*, 2011.

[22] J. Real and A. Crespo. Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Systems*, 26(2):161–197, 2004.

[23] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. Cosynthesis of energy-efficient multimode embedded systems with consideration of mode-execution probabilities. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(2):153–169, 2005.

[24] Y. Shin, D. Kim, and K. Choi. Schedulability-driven performance analysis of multiple mode embedded real-time systems. In *Proc. of the 37th Design Automation Conference (DAC)*, 2000.

[25] N. Stoimenov, S. Perathoner, and L. Thiele. Reliable mode changes in real-time systems with fixed priority or EDF scheduling. In *Proc. of the 2009 Design, Automation and Test in Europe Conference and Exposition (DATE)*, 2009.

[26] J.-P. Talpin, C. Brunette, T. Gautier, and A. Gamatié. Polychronous mode automata. In *Proc. of the International Conference on Embedded Software (EMSOFT)*, pages 83–92, 2006.

[27] K. W. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *Proc. of the IEEE Real-Time Systems Symposium*, 1992.

[28] M. G. Valls, A. Alonso, and J. A. de la Puente. Mode change protocols for predictable contract-based resource management in embedded multimedia systems. In *Proc. of the International Conference on Embedded Software and Systems (EMSOFT)*, 2009.