4-2011

A Semantic Framework for Mode Change Protocols

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

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

Oleg Sokolsky
University of Pennsylvania, sokolsky@cis.upenn.edu

A Semantic Framework for Mode Change Protocols

Abstract
We present a unified framework for the specification and analysis of mode-change protocols used in multi-mode realtime systems. We propose a highly expressive formalism, called MCP, to model the system behavior during mode transitions, and show how various existing mode change protocols can be described as MCPs. The explicit representation of the MCP model provides a means to analyze the system state during a mode transition as well as during an intra-mode execution. We introduce the concept of feasibility with respect to the MCP model, and give a decidable method for checking the feasibility of a MCP for a given multi-mode system. The formalization of mode change behaviors using the MCP model allows a range of mode change protocols to be modeled, evaluated, and optimized to the specific operations and performance requirements of the system. Besides feasibility analysis, it is also possible to analyze other system behaviors (e.g., delay between modes, buffer backlog) using automata verification techniques. Our framework can also be used to describe mode change semantics of multi-mode systems whose modes/transitions have different criticality levels, or of systems composed of multiple multi-mode components that require different mode change protocols.

Disciplines
Computer Engineering | Engineering

Comments
17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Chicago, April 2011.

A Semantic Framework for Mode Change Protocols

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*— We present a unified framework for the specification and analysis of mode-change protocols used in multi-mode real-time systems. We propose a highly expressive formalism, called MCP, to model the system behavior during mode transitions, and show how various existing mode change protocols can be described as MCPs. The explicit representation of the MCP model provides a means to analyze the system state during a mode transition as well as during an intra-mode execution. We introduce the concept of feasibility with respect to the MCP model, and give a decidable method for checking the feasibility of a MCP for a given multi-mode system. The formalization of mode change behaviors using the MCP model allows a range of mode change protocols to be modeled, evaluated, and optimized to the specific operations and performance requirements of the system. Besides feasibility analysis, it is also possible to analyze other system behaviors (e.g., delay between modes, buffer backlog) using automata verification techniques. Our framework can also be used to describe mode change semantics of multi-mode systems whose modes/transitions have different criticality levels, or of systems composed of multiple multi-mode components that require different mode change protocols.**

I. Introduction

A wide spectrum of real-time embedded systems, ranging from conventional control systems to modern smart devices, operate in multiple *modes* that correspond to different mutually exclusive phases of operation and control. An aircraft, for instance, operates at one of the four modes: take-off, normal cruise, emergency, and landing. A smartphone may run in the voice-processing, video-decoding, GPS, or idle mode. Each such mode may be characterized by a different set of tasks, different data arrival rates, a different resource availability, and a different scheduling policy. Mode changes may be both time-triggered (e.g., servicing time-triggered interrupts) and event-triggered (e.g., an incoming voice call). Such a triggering condition is called a *mode change request*.

A change in the operating mode of a multi-mode system introduces a transient stage where the system needs to process both the pending events (data items) in the buffers of the current mode and new incoming events of the new mode. This co-execution of both types of events may lead to a temporal overload that causes some tasks to miss their deadlines. Simultaneously, the system may need to perform transitional activities (e.g., saving state variables to maintain functional correctness), which adds delay to the mode transition. Hence, appropriate mechanisms for handling transient stages – otherwise known as *mode change protocols* – are crucial to the overall performance of the system.

Related work: Several techniques have been proposed to extend models and timing analysis techniques from the real-time systems literature to accommodate multimodal behaviors. For example, the frameworks presented in [4], [5] allow certain tasks to intentionally change their execution periods, which is a type of mode change. The variable rate execution (VRE) [8] task model allows variable execution time and period, which may change at any time, and allows tasks to enter or leave the system at arbitrary times. Different mode change protocols have been studied (see e.g., [7], [13], [17], [18], [20]) and have been classified in [16]. Techniques developed in these papers ensure that all deadlines are met in each of the modes and during the mode transitions. They can be categorized into *synchronous*, where new mode tasks are released only when the current mode tasks have all completed their last activations, and *asynchronous*, where a combination of current and new mode tasks execute during the transition [16]. Asynchronous protocols typically involve discarding unfinished tasks of the current mode or applying an offset to the initial activations of the new mode tasks. The feasibility analysis of the overall system can then be reduced to computing the minimum offsets for the new tasks (e.g., [9], [18]).

These existing mode change protocols exhibit many restrictions, however. First, they assume that during a mode transition, pending events in the buffers of the tasks that are not active in the new mode (M_{new}) must all be finished/discarded before leaving the new mode, thus allowing no cascading pending events. In streaming multimedia systems where data loss is not desirable, these protocols will enforce the pending encoded video data to be completed in M_{new}. Since video decoding tasks are computationally expensive, this enforcement will effectively lead to (i) a very large offset for the tasks in M_{new} (to ensure schedulability), or (ii) a long delay before the system can leave M_{new}. The former (latter) is intolerable if M_{new} (the mode following M_{new}) contains a safety-critical task. On the other hand, delaying the output video stream is usually acceptable; hence, a better alternative for this system is to delay the processing of the pending events in M_{new} (i.e., preserve the buffer state) until the system moves back to a video-decoding mode.

Second, these protocols are based on an analysis that considers each transition in isolation and considers only the active tasks in the two modes of the transition. Thus, they ignore the timing constraints and the buffer states when a mode change request occurs, thereby assuming the worst-case behavior. Consider for instance a scenario where the system changes from a mode M to a mode M' if an input buffer B contains no more than 2 events ("$\mathsf{bl}(B) \leq 2$"). Suppose further that the maximum backlog of B when the system is at M is 100 and the task that processes the events in B is not active in M'. In this case, any analysis that does not consider the constraint "$\mathsf{bl}(B) \leq 2$" but instead assumes "$\mathsf{bl}(B) = 100$"

when the mode transition occurs will be overly pessimistic. An analysis that neglects timing constraints associated with the transitions/modes will also experience similar accuracy loss.

Third, all the proposed techniques assume a single mode change protocol for the entire system. As we will see in Section VII, this is often too restrictive for systems where different modes exhibit different criticality levels that need different handling methods. Further, these protocols also lack expressiveness because they cannot be used to describe the mode change behavior of a system that is composed of multiple subsystems, with each employing a different protocol. Currently, none of the existing frameworks supports the combination of different transition-based mode change protocols and composition of different mode change protocols.

Finally, existing feasibility analysis techniques are protocol-dependent and assume simplistic multi-mode models. These protocols are also often given in a semi-formal description style. The lack of formal semantics and expressiveness restricts their utility and makes their evaluation challenging.

Semantics of multi-mode systems has also been explored for the mode-automata model used in dataflow synchronous languages (e.g., Lustre programs) [11], [19]. However, their focus is on the functional correctness of the programs instead of timing and resource aspects. [3] presented a mode change semantics for AADL models, but this semantics is specific to the AADL mode change protocol, which is restrictive as it requires all threads to be synchronized when moving to a new mode. Timed automata extended with tasks [2] can also be used for checking schedulability of systems in which tasks may change during run time. However, these models assume a constant processor availability and do not consider the effects of mode changes. They too do not provide a means to handle system overload during mode transitions. [12] discussed the general meaning of modes and mode changes in uniprocessor systems, which align with our notions of modes and mode changes; however, no concrete semantics for mode change protocols was given.

Our contributions: In this paper, we propose a semantic framework for the formal modeling and analysis of multi-mode protocols that overcomes the above drawbacks of existing techniques. We present a model for mode change protocols (denoted in this work as MCP) that is highly expressive and capable of capturing various realistic mode change behaviors. Each MCP model is a DAG-structured finite state automaton where each node captures the actions that must be performed by the system in an intermediate state during a mode transition. Each edge specifies a buffer update and a timing/buffer condition between two intermediate transitional states. We show how the MCP model can be used to describe several existing mode change protocols, as well as protocols that cannot be expressed by existing techniques such as (i) preempting a task whose deadline is not stringent but data loss is undesirable (e.g., in multimedia applications); (ii) discarding a non-critical task whose stale data (previous data values) is no longer useful in the current system state (e.g., weather information data in the emergency state of automotive applications), (iii) restarting a partially executed task to minimize transition delay required in saving state variables (e.g., when moving to an emergency mode) while allowing the task to be executed at some point later, (iv) delaying new task activations if the existing tasks in the current mode are more critical, and (v) performing a mixture of the above depending on the current state of the buffers.

The MCP model is designed for general mixed time- and event-triggered multi-mode systems that process complex event/data streams (captured by arrival functions [6]) and general resources (captured by service functions [6]). Given a multi-mode system, modeled as a multi-mode automaton, and its associated MCPs, we describe the precise semantics of the system during an intra-mode and an inter-mode execution. With this semantics, we can now reason about the system state (e.g., the buffer state, the maximum mode change delay) during both normal execution at a mode and a transient mode transition stage. We define the notion of feasibility with respect to the MCP model and give a decidable method for checking if a (set of) mode change protocol(s) is feasible for a given multi-mode automaton. We then illustrate the utility of our framework using an adaptive cruise control system.

Organization of the paper: The next section introduces the basic concepts and the multi-mode automata model for multi-mode systems. Section III describes the behaviors of different elements of a system during a mode transition, and presents the MCP model that formalizes such mode change behaviors. Section IV presents a technique for modeling existing mode change protocols using MCP. The formal semantics of MCP and of a multi-mode automaton associated with an MCP is given in Section V, followed by a feasibility analysis technique in Section VI. Finally, we present in Section VII an illustrating example that showcases some of the benefits of our semantic framework before concluding the paper.

II. System Description and Basic Models

Tasks and input events. The system consists of a finite set of tasks, each of which processes an input event stream. Each task has a finite input FIFO buffer that stores the incoming events from the stream. Whenever an event arrives at the buffer of a task, an instance (a job) 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 $T = (B, E, D, \alpha, \pi)$, where B, E, D, α and π are the input buffer, the worst-case execution demand, the relative deadline, the arrival function of the task, and the priority, respectively. The priority of a task is used only when it is scheduled with other tasks in the system under a Fixed Priority (FP) scheduling policy; it is set to 0, otherwise. Note that, the smaller π is, the higher priority the task has. Fig. 1 shows a task and its parameters.

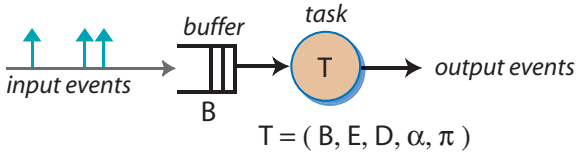


Fig. 1. A task and its parameters.

The *arrival function* α of T consists of an upper arrival function $\alpha^u(\Delta)$ and a lower arrival function $\alpha^l(\Delta)$, which specify the maximum and minimum number of events that can arrive at the input buffer B over any time interval of length Δ, respectively, for all $\Delta \in \mathbb{N}$ [6]. This arrival function α captures all the acceptable arrival patterns of the input events of T, which are also the release patterns of the jobs of T. Each pattern is abstracted as a sequence of non-negative integers $A = k_1 k_2 \cdots k_m$, where k_i is the number of events (jobs) that arrive (are released) over the unit interval $(i-1, i]$, such that

$$\forall\, 1 \le i \le j \le m:\ \ \alpha^l(j-i+1) \ \le \ \sum_{h=i}^{j} k_h \ \le \ \alpha^u(j-i+1).$$

In this case, we say A is bounded by α, written as $A \models \alpha$. One can verify that $A \models \alpha$ iff $A' \models \alpha$ where $A' = k_m k_{m-1} \cdots k_1$.

The processing requirement of an event in the buffer of T is captured by the job that processes the event, given by $J = (e, d)$, where e is the remaining execution time (RET), and d is the elapse time to deadline (ETD), i.e., the duration from the current time to the job's absolute deadline. A new input event of T is characterized by $J = (E, D)$. In other words, one can view the buffer of T as a job waiting queue that contains unfinished jobs of T, which will be executed (sequentially in their release orders) when T is active. We assume that E, D, $\alpha^u(\Delta)$, $\alpha^l(\Delta)$, and π are finite non-negative integer numbers for all $\Delta \in \mathbb{N}$. Further, B has a finite capacity, denoted by $\mathsf{size}(B)$.

Resource. The resource availability of a processing element (PE) is modeled by a *service function* $\beta = (\beta^u, \beta^l)$, where $\beta^u(\Delta)$ and $\beta^l(\Delta)$ specify the maximum and minimum number of execution units (e.g., processor cycles) that can be provided by the PE over any time interval of length Δ. Service patterns are defined in the same manner as arrival patterns. In this paper, all jobs in a multi-mode system are executed (sequentially) on a single PE.

Multi-mode automata. The behavior of a multi-mode system is modeled by a multi-mode automaton (MMA), which is a finite state machine whose states represent operating modes and transitions represent mode changes. Each state of the automaton specifies the set of tasks that are active, the scheduling policy, and the available resource when the system is executing at the corresponding mode. The guard associated with a transition specifies a mode change request (MCR). The MMA model resembles the one in [15], except that, it has a resource supply associated with the states and its mode change semantics can be given by any general protocol modeled by the MCP model (introduced in the coming section).

Let INT be the set of intervals $[k, k']$ with $0 \le k \le k'$ and $k, k' \in \mathbb{N}$. By abuse of notation, we use $[k, \infty]$ to denote the right open interval $[k, +\infty)$. We denote by $\mathsf{bl}(B)$ the backlog of B. The model is formally defined as follows.

An MMA is a tuple $\mathcal{A} = (\mathcal{T}, \mathcal{B}, \mathcal{M}, M_{\mathsf{in}}, \mathsf{Inv}, \Phi, Act, \mathrm{R})$ where

- $\mathcal{T}$ is a finite set of tasks of the system.
- $\mathcal{B}$ is a finite set of input buffers of the system.
- $\mathcal{M}$ is a finite set of modes.
- $M_{\mathsf{in}} \in \mathcal{M}$ is the initial mode.
- $\mathsf{Inv} : \mathcal{M} \to \mathsf{INT}$ 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.
- Φ is the set of constraints of the form $\mathsf{bl}(B) \# c$ where $\# \in \{<, >, \le, \ge\}$ and $B \in \mathcal{B}$.
- Act is a set of signals that trigger the mode changes.
- $\mathrm{R} \subseteq \mathcal{M} \times Act \times \Phi \times \mathsf{INT} \times \mathcal{M}$ is a transition relation.

Each mode $M \in \mathcal{M}$ has the form $\langle \tau, \beta, \mathsf{SC} \rangle$ where $\tau \subseteq \mathcal{T}$ is the set of active tasks, β is the service function of the available resource, and SC is the scheduling policy at M. The buffer of an active task in M is also said to be active in M.

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 a signal that triggers the transition, φ is a guard on the backlogs of the buffers, and I is the interval (relative to the instant the system enters M) during which the transition can be enabled. MMA transitions are non-deterministic. Further, if an outgoing transition from a mode is enabled, the system will take the transition as soon as it satisfies the delay imposed by the mode change protocol associated with the MMA.

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

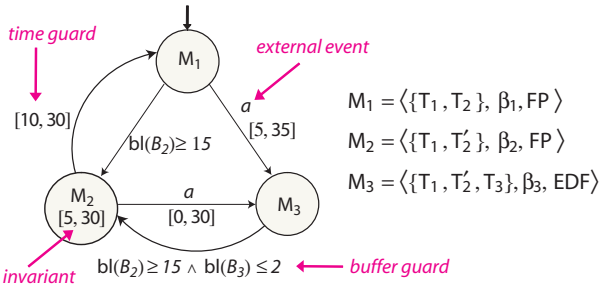


Fig. 2. A multi-mode automaton.

As shown in the figure, the system is initially at mode M_1 where T_1 and T_2 are active. The processor initially offers a service function β_1, which is shared between T_1 and T_2 under FP with T_1 having higher priority than T_2. While the system is at M_1, if the input buffer B_2 contains more than 15 events (denoted by "$\mathsf{bl}(B_2) \ge 15$"), the system will move to mode M_2. At M_2, the task T_2 is modified to become T_2' which has a deadline doubled that of T_2. At the same time, the service function is increased to β_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. During this duration, however, if the signal a arrives, the system moves to M_3 where the processor offers a service function β_3. At M_3, the task T_3 becomes active and it will be scheduled together with T_1' and T_2 under EDF. The rest can be explained accordingly.

III. Mode Change Protocol (MCP) Model

A mode change protocol describes the execution behavior of a multi-mode system 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. Different transitions of a system may require different protocols. For example, one can use a protocol that delays the arrival of the input events of a new task when an aircraft transits from the take-off mode to the cruise mode; however, this protocol is undesirable for a transition from the cruise mode to the emergency mode, which requires the emergency jobs to be released and executed as soon as possible. In general, the desirable mode change behaviors vary depending on the characteristics of the system. We identify below a spectrum of mode change behaviors that will be formalized by the MCP model.

A. System behaviors enforced by mode change protocols

Pending events in the buffers: There are generally three ways to handle the pending events in an active buffer in the origin mode: (a) all pending events remain unchanged (i.e., the buffer state is preserved); (b) a subset of the pending events are discarded; and (c) a subset of the pending events are transfered (migrated) to another buffer. The subset of pending events in (b) and (c) is determined based on the designer's objective, which often falls into the following categories for some $k \geq 1$:

- *Oldest*. The set of k oldest events (arrived earliest).
- *Newest*. The set of k newest events (arrived latest).
- *Highest priority*. The set of k events with highest priorities with respect to (w.r.t.) a scheduling policy.
- *Lowest priority*. The set of k events with lowest priorities w.r.t. a scheduling policy.
- *Random*. The set of k randomly chosen events.

Besides, the execution demands and deadlines of these pending events can be either preserved or assigned to the new parameters of the associated tasks in the destination mode. Thus, their timing parameters upon a mode change depend on the mode at which they arrive, in the former case, and on the mode at which they are processed, in the latter case.

The partially processed events: These events are either partially processed by a preempted job or currently being processed by an executing job. These events can be (i) continued to be processed until completion, (ii) saved and continued at some point later, (iii) discarded, or (iv) reset (their timing state) and re-queued in their input buffers (e.g., in systems that allow job rescheduling).

Active tasks during the transition interval: The set of tasks that are active during a mode transition is often a subset of the active tasks in the origin and destination mode, possibly with modified timing parameters.

Scheduling policy: The scheduling policy used during a mode transition is either the scheduling policy of the origin mode or of the destination mode.

Triggering conditions of the above actions: The conditions during which the above behaviors are enforced can be specified as a constraint on a timing delay, on the RET of a partially processed event, or on the backlog of an input buffer. These must be specified until the destination mode is successfully entered.

Mode change deadline: The behavior of a mode change protocol specified above must guarantee that the transition completes within a bounded amount of time called *mode change deadline*. A protocol is usually considered to be better than another one if it satisfies a smaller mode change deadline.

In the following section, we present the theory of MCP model, which formalizes the above mode change behaviors.

B. The mode change protocol (MCP) model

An MCP is a DAG-structured finite automaton, where each run of the automaton explicitly captures the behavior imposed on the system during a mode transition. Each node along a run of the MCP specifies the specific tasks that are executed at an intermediate system state during the mode transition. For example, to force the pending events to be processed during a mode transition, we include the tasks associated with these events into the active task set of the MCP nodes, while modifying the events' arrival functions to allow/disallow new event arrivals. The guard associated with an edge between two nodes of an MCP gives the condition upon which the system moves from one intermediate state (source node) to the next intermediate state (destination node). The buffer update along an edge specifies a specific way of handling the pending events and the partially executed events (as outlined earlier). For instance, resetting the RET of a partially processed event to zero will discard the event, whereas resetting the RET to its initial value will restart the event. Similarly, resetting a buffer content to an empty buffer is equivalent to discarding all the pending and partially executed events in the buffer. Such buffer updates are done via buffer valuations, which are described below.

Buffer valuation. By abuse of notation, we denote by variable B the content of the physical buffer B. Each value of B corresponds to a state of the buffer. A buffer valuation is a function V which assigns to each buffer B a vector of size $\mathsf{size}(B)+1$, where $\mathsf{size}(B)$ is the capacity of the buffer.[1] The i^{th} element of $V(B)$, i.e., $V(B)[i]$, contains the timing information of the i^{th} oldest event (ev_i) currently in the buffer. It is given as a job $J_i = (e_i, d_i)$ that is released when ev_i arrives (recall that e_i is the RET and d_i is the ETD). An empty slot in B is characterized by an empty job $J = (0,0)$. Note that a real job has $(0,0)$ as its parameters only when it finishes executing, in which case it can be removed from the buffer (thereby leaving an empty slot in the buffer). The backlog of a buffer B under a valuation V is the number of non-empty jobs in $V(B)$, i.e.,

$$\mathsf{bl}(B) \stackrel{\mathsf{def}}{=} |V(B)| = \sum \{1 \mid 1 \leq i \leq \mathsf{size}(B)+1 \wedge J_i \neq (0,0)\}$$

where $V(B) = [J_1, \ldots, J_{\mathsf{size}(B)+1}]$. We say V satisfies a buffer constraint "$\mathsf{bl}(B) \# c$", written as $V \models (\mathsf{bl}(B) \# c)$, iff $|V(B)| \# c$.

Definition 1 (MCP Model)**.** *A mode change protocol $\mathcal{P}$ is a DAG-structured finite automaton $\mathcal{P} = (\mathcal{M}_\mathcal{P}, \mathcal{E}, M_{\mathsf{orig}}, M_{\mathsf{dest}})$ where $\mathcal{M}_\mathcal{P}$ is the set of state (nodes), $\mathcal{E}$ is the set of transitions*

[1] The additional slot will be used to check buffer overflows.

(edges), M_{orig} is the initial state (unique root) and M_{dest} is the final state (unique sink). Each node v of $\mathcal{P}$ has the same syntax and semantics as that of an MMA mode. Each edge $\rho \in \mathcal{E}$ from v to v' is characterized by $\rho = \langle v, \delta, \varphi, Z, v' \rangle$ where:

- $\delta \in \mathsf{INT}$ *is the interval during which* ρ *can be taken.*
- φ *is the guard associated with* ρ*, which is specified as a conjunction of atomic forms* $\mathsf{bl}(B)\#c$ *and* $e_J\#c$ *where* $\# \in \{<, >, \leq, \geq\}$*, B is an active buffer in v or in v', and $J = (e_J, d_J)$ is a job in some active buffer in v or in v'.*
- Z *is the reset valuation of the buffers, i.e., $Z(B)$ gives the new state of B when the system enters v' if B appears in Z, and $Z(B)$ gives the current state of B otherwise.*

All edges in $\mathcal{P}$ are instantaneous. Further, for each $v \in \mathcal{M}_{\mathcal{P}}$, its service function and its scheduling policy must be the same as that of M_{orig} or that of M_{dest}. Additionally, for every task T active in v, there is a task T' active in M_{orig} or in M_{dest} with the same input buffer that has a larger upper arrival function and a larger worst case execution demand than T does. The last condition is to ensure that the protocol does not introduce extra workload during a mode transition.

Composition of MCP models. Consider a system that is composed of two communicating (via common signals) MMA models $\mathcal{A}_1$ and $\mathcal{A}_2$, where each $\mathcal{A}_i$ uses an MCP $\mathcal{P}_i = (\mathcal{M}_i, \mathcal{E}_i, M^i_{\mathsf{orig}}, M^i_{\mathsf{dest}})$. The mode change semantics of the system can be described by a protocol $\mathcal{P}$ that is the composition of $\mathcal{P}_1$ and $\mathcal{P}_2$, given by $\mathcal{P} = (\mathcal{M}_{\mathcal{P}}, \mathcal{E}, M_{\mathsf{orig}}, M_{\mathsf{dest}})$ where $\mathcal{M}_{\mathcal{P}} = \mathcal{M}_1 \times \mathcal{M}_2$, $\mathcal{E} \subseteq \mathcal{E}_1 \times \mathcal{E}_2$, $M_{\mathsf{orig}} = (M^1_{\mathsf{orig}}, M^2_{\mathsf{orig}})$ and $M_{\mathsf{dest}} = (M^1_{\mathsf{dest}}, M^2_{\mathsf{dest}})$. The transition relation $\mathcal{E}$ can be computed in the same manner as the composition of two *MMA* described in [15]. It can be easily verified that the MCP model is closed under composition.

As demonstrated in the next section, the MCP model formalizes the common mode change protocols in literature. In addition, it can also describe more general protocols beyond the existing ones. For instance, protocols that allow behaviors such as cascading pending events, discarding some events while delaying or executing others, dynamically adjusting the execution priorities of the jobs based on the buffer state. With MCP, we can express mode-dependent and transition-dependent protocols for systems that distinguish between different criticality levels of modes/transitions by simply associating with each multi-mode transition an MCP that is best suited for the transition. Additionally, one can model and analyze the mode change semantics of a system comprising multiple multi-mode components that employ different mode change protocols via MCP composition.

Instance of MCP with respect to an MMA. Observe that MCP is defined in a generic manner, i.e., independent of the multi-mode automaton. The exact system behaviors that are enforced by an MCP depends on the specific transition in the MMA that employs the MCP. Suppose $\sigma = (M, a, \varphi, I, M')$ is a transition of an MMA $\mathcal{A}$ and $\mathcal{P} = (\mathcal{M}_{\mathcal{P}}, \mathcal{E}, M_{\mathsf{orig}}, M_{\mathsf{dest}})$ is an MCP that is associated with σ. The protocol obtained by instantiating M_{orig} to M and M_{dest} to M' is called an *instance* of $\mathcal{P}$ w.r.t. σ, written as $\mathcal{P}_\sigma$. Thus, the behavior of $\mathcal{A}$ during the transitional period when $\mathcal{A}$ moves from M to M' via σ is exactly the behavior of $\mathcal{P}_\sigma$. This concept will be clear in the next section where we demonstrate examples of MCP that model common mode change protocols and their behaviors when applied to a specific transition of an MMA. The precise semantics of MCP w.r.t. to an MMA will be detailed in Section V.

IV. Modeling Common Mode Change Protocols

We first state some notations that are necessary to model common mode change protocols. The example transition below will be used to illustrate various concepts throughout this section.

Example 1 (A running example)**.** *Consider a multi-mode transition σ from $M = (\tau, \beta, FP)$ to $M' = (\tau', \beta', EDF)$ with $\tau = \{T_1, T_2, T_3\}$ and $\tau' = \{T'_1, T_3, T_4\}$, where $T_1 = (B_1, 2, 5, \alpha_1, 1)$, $T'_1 = (B_1, 2, 6, \alpha'_1, 1)$, $T_2 = (B_2, 3, 7, \alpha_2, 2)$, $T_3 = (B_3, 4, 20, \alpha_3, 3)$ and $T_4 = (B_4, 3, 10, \alpha_4, 0)$.*

Task classification. Let $M_{\mathsf{orig}} = \langle \tau_{\mathsf{orig}}, \beta_{\mathsf{orig}}, \mathsf{SC}_{\mathsf{orig}} \rangle$ and $M_{\mathsf{dest}} = \langle \tau_{\mathsf{dest}}, \beta_{\mathsf{dest}}, \mathsf{SC}_{\mathsf{dest}} \rangle$ be the root node and sink node of an MCP. Denote B_T and $B_{T'}$ as the buffers of T and T', respectively. The tasks that are active in these nodes can be classified into five disjoint sets as follows.

- $\tau_{\mathsf{AA}} = \tau_{\mathsf{orig}} \cap \tau_{\mathsf{dest}}$: the set of tasks that are active in both M_{orig} and M_{dest}.
- $\tau_{\mathsf{AC}} = \{T \in \tau_{\mathsf{orig}} \mid \exists T' \in \tau_{\mathsf{dest}} : T' \neq T \wedge B_{T'} = B_T\}$: the set of tasks that are active in M_{orig}, which will be modified when the system moves to M_{dest}.
- $\tau_{\mathsf{CA}} = \{T' \in \tau_{\mathsf{dest}} \mid \exists T \in \tau_{\mathsf{orig}} : T' \neq T \wedge B_{T'} = B_T\}$: the set of tasks that are active in M_{dest}, which are modifications of some tasks that were active in M_{orig}.
- $\tau_{\mathsf{AI}} = \{T \in \tau_{\mathsf{orig}} \mid \forall T' \in \tau_{\mathsf{dest}} : B_{T'} \neq B_T\}$: the set of tasks that are active in M_{orig} and inactive in M_{dest}.
- $\tau_{\mathsf{IA}} = \{T' \in \tau_{\mathsf{dest}} \mid \forall T \in \tau_{\mathsf{orig}} : B_{T'} \neq B_T\}$: the set of tasks that are active in M_{dest} and inactive in M_{orig}.

Thus, $\tau_{\mathsf{orig}} = \tau_{\mathsf{AA}} \cup \tau_{\mathsf{AC}} \cup \tau_{\mathsf{AI}}$ and $\tau_{\mathsf{dest}} = \tau_{\mathsf{AA}} \cup \tau_{\mathsf{CA}} \cup \tau_{\mathsf{IA}}$. The tasks in τ_{AA} (resp. $\tau_{\mathsf{AI}}, \tau_{\mathsf{IA}}$) are also known as *unchanged* (resp. *old*, *new*) tasks. The tasks in τ_{AC} and τ_{CA} are called *changed* tasks. Note that each task T' in τ_{CA} is the modification of a task T in τ_{AC} when the system moves to M_{dest} (i.e., both T, T' share the same buffer).

Example 2. *Suppose $\tau_{\mathsf{orig}} = \tau$ and $\tau_{\mathsf{dest}} = \tau'$ where τ and τ' are defined in Example 1. Observe that T'_1 and T_1 share the same buffer B_1 but have different timing parameters. Thus, $\tau_{\mathsf{AA}} = \{T_3\}$, $\tau_{\mathsf{AC}} = \{T_1\}$, $\tau_{\mathsf{CA}} = \{T'_1\}$, $\tau_{\mathsf{AI}} = \{T_2\}$, and $\tau_{\mathsf{IA}} = \{T_4\}$.*

Notations. Let τ be a set of tasks. Then, $\mathcal{B}_\tau$ denotes the set of input buffers of the tasks in τ. $\mathsf{bl}(\mathcal{B}_\tau)\#c$ is the conjunction of all $\mathsf{bl}(B)\#c$ for $B \in \mathcal{B}_\tau$. Thus, a valuation V satisfies the buffer constraint $\mathsf{bl}(\mathcal{B}_\tau)\#c$ iff it satisfies $\mathsf{bl}(B)\#c$ for all $B \in B_\tau$. Further, $\Gamma(\tau)$ is the set of tasks obtained from τ after setting the arrival functions of each task in τ to the zero function (i.e., for all $T \in \Gamma(\tau)$, $\alpha^u_T(\Delta) = \alpha^l_T(\Delta) = 0$ for all $\Delta \geq 0$) while keeping other parameters intact.

Example 3. *Consider the transition in Example 1. Then, $\mathcal{B}_\tau = \{B_1, B_2, B_3\}$ and $\mathcal{B}_{\tau'} = \{B_1, B_3, B_4\}$. Hence, $\mathsf{bl}(\mathcal{B}_\tau)\#c = (\mathsf{bl}(B_1)\#c) \wedge (\mathsf{bl}(B_2)\#c) \wedge (\mathsf{bl}(B_3)\#c)$. Besides, $\Gamma(\tau) =$*

$\{T_1^*, T_2^*, T_3^*\}$ *with* $T_1^* = (B_1, 2, 5, \alpha_0, 1)$, $T_2^* = (B_2, 3, 7, \alpha_0, 2)$, $T_3^* = (B_3, 4, 20, \alpha_0, 3)$, *where* $\alpha_0^u(t) = \alpha_0^l(t) = 0$ *for all* $t \geq 0$.

Using the above notations, we illustrate next how various mode change protocols described in [16] can be modeled by MCP. The reader is referred to [16] for their naming convention. As these protocols assume the same service function and the same scheduling policy for all modes, in our models the service function and the scheduling policy will be updated to that of the destination mode upon the first activation of a new task in the destination mode.

A. Synchronous Idle Time Protocol

The arrival of a mode change request (MCR) does not affect the normal activation of tasks in the original mode until an idle instant (no CPU load) occurs. In other words, the mode change can only happen when all the input buffers of the tasks τ_{orig} are empty. This is achieved by having a constraint on the backlogs of these buffers, as depicted in Fig. 3. Fig. 4 shows an instance of the protocol w.r.t. the transition σ defined in Example 1.

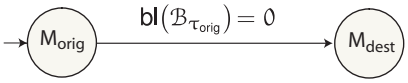


Fig. 3. Synchronous Idle Time Protocol.

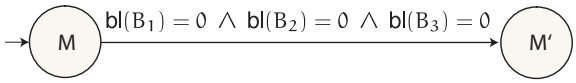


Fig. 4. An instance of the protocol in Fig. 3 with respect to σ (cf. Example 1).

B. Maximum Period Single Offset Protocol

When an MCR occurs, unchanged tasks (τ_{AA}) are unaffected and pending jobs in the origin mode are completed as normal. Old tasks and changed tasks of the origin mode (τ_{IA} and τ_{AC}) are released as normal for up to $P_{\max}$ time units, where $P_{\max}$ is the period of the least frequent task in both modes. New tasks and the modified tasks in the destination mode (τ_{IA} and τ_{CA}) are only released after $P_{\max}$ time units. The protocol is shown in Fig. 5.

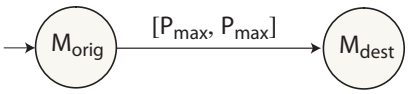


Fig. 5. Maximum Period Single Offset Protocol.

C. Synchronous Minimum Offset without Periodicity Protocol

When an MCR occurs, pending jobs in the origin mode are completed as normal. There will be no new releases of the tasks in the origin mode (τ_{orig}). Tasks in the destination mode (τ_{dest}) are delayed until all pending tasks of the origin mode complete. The MCP model of the protocol is given in Fig. 6. An instance of the protocol with respect to the transition σ is shown in Fig. 7.

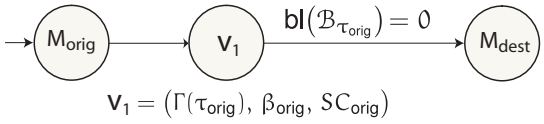


Fig. 6. Synchronous Minimum Offset without Periodicity Protocol.

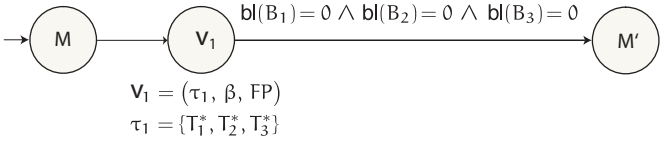


Fig. 7. An instance of the protocol in Fig. 6 with respect to σ (cf. Example 1). The value of τ_1 is given in Example 3.

D. Synchronous Minimum Offset with Periodicity Protocol

When an MCR occurs, unchanged tasks (τ_{AA}) are unaffected and pending jobs in the origin mode are completed as normal. There will be no new releases of the old tasks and changed tasks of the origin mode (τ_{AI} and τ_{AC}). New tasks and the modified tasks in the destination mode (τ_{IA} and τ_{CA}) are delayed until there is no more pending tasks of τ_{AI} and τ_{AC}. This protocol can be modeled as an MCP depicted in Fig. 8.

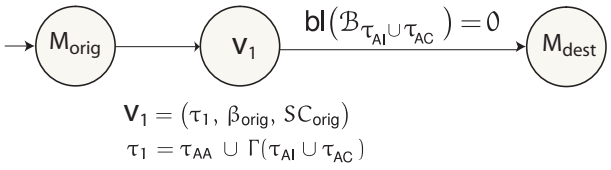


Fig. 8. Synchronous Minimum Offset with Periodicity Protocol.

Similarly, one can also model the *Asynchronous without Periodicity Protocol* and the Asynchronous with Periodicity Protocol proposed by Real and Crespo [16] using the MCP model. The details of the models are available in our technical report [14].

Semantics preservation. One can easily verify that the MCPs given above preserve the same semantics as their corresponding existing protocols. Unlike the existing ones, however, our MCP representation captures the mode change behaviors explicitly, thereby allowing the analysis of the system state during mode transitions.

V. Semantics of MCP and MMA

The semantics of MCP and MMA are defined based on the semantics of modes and transitions, which we now develop.

A. Mode and transition semantics

We assume that all arrival and service functions are specified for a finite number of time intervals. Let $N \in \mathbb{N}$ be the smallest upper-bound on the lengths of the intervals constrained by the arrival and service functions in the system.

Suppose $A = c_1 c_2 \cdots c_m$ is a finite sequence with $c_i \in \mathbb{N}$ for all $1 \leq i \leq m$ and $c \in \mathbb{N}$. We denote by $c \circ A$ the sequence $c\, c_1\, c_2 \cdots c_m$ if $m < N$, and $c\, c_1\, c_2 \cdots c_{N-1}$ otherwise, with N defined above. We further denote by $\emptyset$ the empty sequence

and $k_+ = \max\{k,0\}$ for all k. Let $\mathcal{B} = \{B_1,\ldots,B_n\}$ be the set of buffers in the system.

Definition 2 (Mode configuration)**.** *A state of a mode* $M = \langle \tau, \beta, \mathsf{SC} \rangle$ *is a configuration of the form* $S = (M, V, \vec{A}, C, t)$ *where*

- V *is a buffer valuation.*
- $\vec{A} = \{A_1,\ldots,A_n\}$ *where each* A_i *is an arrival pattern of the input events of* B_i*, if* B_i *is active in* M*, and* $A_i = \emptyset$ *otherwise.*
- C *is a service pattern of* M*, i.e.,* $C \models \beta$*.*
- $t \in \mathbb{N}$ *is the number of time units the system has been at* M*.*

Let V be a valuation of the buffers at M. The valuation of the buffers after x_i new events arrive at each B_i when the system is in M is given by $V \oplus_M x$ with $x = [x_1,\ldots,x_n]$, computed as follows. For each buffer B_i, we denote by $T_i = (B_i, E_i, D_i, \pi_i)$ the active task in M that is associated with B_i. Then, for all $1 \le j \le \mathsf{size}(B_i)+1$, $(V \oplus_M x)(B_i)[j]$ is

- $V(B_i)[j]$, if $1 \le j \le |V(B_i)|$, and
- (E_i, D_i), if $|V(B_i)| < j \le \min\{|V(B_i)| + x_i, \mathsf{size}(B_i)+1\}$.

Note that if $x_i + |V(B_i)| > \mathsf{size}(B_i)$, the buffer B_i will overflow.

Buffer updates with respect to scheduling policy. Suppose x_i is the number of events that arrive at B_i and y is the number of execution units offered by the PE over the unit time interval $[t, t+1)$, where $1 \le i \le n$. The valuation V' of the buffers at time $t+1$ when the system is at M is given by a buffer mapping UPDATE that updates the new buffer content based on the current buffer content (given by the valuation V) and the scheduling policy used in the mode, i.e.,

$$V' = \mathsf{UPDATE}(M, V, x, y, \mathsf{SC}) \stackrel{\mathsf{def}}{=} \mathsf{Schedule}(V, y, \mathsf{SC}) \oplus_M x.$$

where $x = [x_1,\ldots,x_n]$. An example of the mapping $\mathsf{Schedule}$ is shown in the following example for FP.

Example 4 (FP schedule)**.** *We assume that all released jobs in a buffer have the same fixed priority, which is the priority of the associated task that is currently active in the mode. For any* k *and* j*, denote* e_j^k *as the RET of the job* $V(B_k)[j]$ *if* B_k *is active in* M*, and* $e_j^k = 0$ *otherwise. Denote* π_k *as the priority of* T_k*. Then, for any buffer* B_i*, the total number of execution units required by the active jobs that have higher priority than the ones in* B_i *is*

$$\bar{y}_i = \sum_{k \ne i} \big\{\, e_j^k \mid \big(\pi_k < \pi_i\big) \text{ or } \big(\pi_k = \pi_i \wedge k < i\big) \big\}.$$

Thus, the number of execution units allocated to each B_i *is* $y_i = \max\{0,\, y - \bar{y}_i\}$ *if* T_i *is active in* M*, and* $y_i = 0$ *otherwise. Further, let* $n_i = \min\{\, j \mid e_1^i + \ldots + e_j^i > y_i \wedge 1 \le j \le |V(B_i)|\}$ *and* $e_i' = e_1^i + \ldots + e_{n_i}^i - y_i$*. Then, after* B_i *is scheduled,* $n_i - 1$ *oldest jobs in* B_i *are completed. Further,* e_i' *will be the RET of the* $(n_i)^{th}$ *job. Let* d_i' *be the ETD of this* $(n_i)^{th}$ *job. Due to time elapsed, all ETDs are decreased by 1. Thus,* $\mathsf{Schedule}(M, V, y, FP)$ *is the valuation* V_{FP} *defined by: For all* $1 \le i \le n$*,* $1 \le j \le m_i$*,* $V_{FP}(B_i)[j]$ *is*

- $\big(e_i',\, (d_i' - 1)_+\big)$*, if* $j = 1$*.*
- $\big(e^i_{j+n_i-1}, (d^i_{j+n_i-1} - 1)_+\big)$*, if* $2 \le j \le |V(B_k)| - n_i + 1$*.*
- $(0,0)$*, otherwise.*

if B_i *is active in* M*, and* $V_{FP}(B_i)[j] = \big(e_j^i, (d_j^i - 1)_+\big)$ *otherwise.*

Execution steps. An execution trace of an MMA can be described as a sequence of mode configurations $S_0 \to S_1 \to \cdots \to S_m$ where S_0 is the initial configuration of the initial mode and each S_j is reachable from S_{j-1}. There are two types of execution steps: intra-mode (i.e., when the system continues to stay at the current mode) and inter-mode (i.e., when the system takes a transition to a new mode).

Definition 3 (Intra-mode execution step)**.** *Consider a mode* $M = \langle \tau, \beta, \mathsf{SC} \rangle$ *and a configuration* $S = (M, V, \vec{A}, C, t)$*. A configuration* $S' = (M, V', \vec{A}', C', t+1)$ *with* $\vec{A}' = \{A_1',\ldots,A_n'\}$ *is* directly reachable *from* S *when* the system remains at M *– written as* $S\,[M, x, y\rangle\, S'$ *– iff there exist non-negative integers* $x_1,\ldots,x_n, y$ *such that*

- $V' = \mathsf{UPDATE}(V, x, y, \mathsf{SC})$*, where* $x = [x_1,\ldots,x_n]$*;*
- $A_i' = x_i \circ A$ *and* $A_i' \models \alpha$ *if there exists* $T \in \tau$ *with input buffer* B_i *and arrival function* α*, and* $A_i' = A_i = \emptyset$ *otherwise;*
- $C' = y \circ C$ *and* $C' \models \beta$*; and* $t + 1 \in \mathsf{Inv}(M)$*.*

Denote $[k]$ as the finite alphabet $\{0, 1, \ldots, k\}$. Let K_1 be the maximum number of events arriving at a buffer over a unit interval and K_2 be the maximum number of execution units offered by the PE over a unit interval, i.e., $K_1 = \max\{\alpha_T^u(1) \mid T \in \tau \wedge (\tau, \beta, \mathsf{SC}) \in \mathcal{M}\}$ and $K_2 = \max\{\beta^u(1) \mid (\tau, \beta, \mathsf{SC}) \in \mathcal{M}\}$, where α_T^u denotes the upper arrival function of T. Then, $w = (x, y)$ is a symbol in the alphabet $\Sigma = [K_1]^n \times [K_2]$. We say that the system accepts (generates) the symbol w at S if $S\,[M, w\rangle\, S'$.

We say a configuration S' is reachable from a configuration S of M if $S\,[M, w\rangle\, S'$ for some $w \in \Sigma$, or there exists a sequence of configurations $S_1,\ldots,S_k$ of M and a word $w = w_1 w_2 \ldots w_k w_{k+1}$ with $w_i \in \Sigma$ such that $S\,[M, w_1\rangle\, S_1\,[M, w_2\rangle \cdots [M, w_k\rangle\, S_k\,[M, w_{k+1}\rangle\, S'$. The word w is said to be accepted by the automaton at S.

Given a buffer valuation V and a buffer reset Z. We denote by $V|Z$ the valuation V' that is defined by: for all $B \in \mathcal{B}$, $V'(B) = Z(B)$ iff B appears in Z, and $V'(B) = V(B)$ otherwise. *Often, Z is a buffer valuation defined based on the objective of the protocol. Consider, for instance, a protocol that will discard k oldest events currently pending in a buffer B, then* $Z(B)[j] = V(B)[k + j]$ *for all* $1 \le j \le \mathsf{size}(B) + 1 - k$ *and* $Z(B)[j] = (0,0)$ *otherwise.*

Definition 4 (Inter-mode execution step)**.** *Consider an edge* $\rho = \langle M, \delta, \varphi, Z, M' \rangle$ *from a mode* $M = \langle \tau, \beta, \mathsf{SC} \rangle$ *to a mode* $M' = \langle \tau', \beta', \mathsf{SC}' \rangle$ *in an MCP. Suppose* $S = (M, V, \vec{A}, C, t)$ *is a configuration of* M*. A configuration* $S' = (M', V', \vec{A}', C', 0)$ *is said to be reachable from* S *when* the system takes the transition ρ*, denoted by* $S\,[\rho\rangle\, S'$*, iff* $t \in \delta$*,* $V \models \varphi$*,* $V' = V|Z$*, and (i)* $\vec{A}' = \{A_1',\ldots,A_n'\}$ *where* $A_i' = A_i$ *if there exists* $T \in \tau \cap \tau'$ *with input buffer* B_i*, and* $A_i' = \emptyset$ *otherwise, and (ii)* $C' = C$ *if* $\beta = \beta'$*, and* $C' = \emptyset$ *otherwise.*

B. Semantics of MCP

Consider an MCP $\mathcal{P} = (\mathcal{M}_\mathcal{P}, \mathcal{E}, M_{\mathsf{orig}}, M_{\mathsf{dest}})$. Suppose S_0 is a starting configuration of M_{orig}. Then, an execution of $\mathcal{P}$ starting from S_0 is a sequence of configurations $S_0 \xrightarrow{w_1} S_1 \xrightarrow{w_2} \cdots \xrightarrow{w_m} S_m$ where $v_0 = M_{\mathsf{orig}}$, $v_m = M_{\mathsf{dest}}$, and for all $1 \leq i \leq m$:

- S_i is a configuration of $v_i \in \mathcal{M}_\mathcal{P}$
- $v_i = v_{i+1}$ and $S_i[v_i, w_{i+1}\rangle S_{i+1}$, or $v_i \neq v_{i+1}$, $S_i[\rho_i\rangle S_{i+1}$ where $\rho_i \in \mathcal{E}$ is a transition from v_i to v_{i+1}, and w_{i+1} is the empty symbol.

The behavior of $\mathcal{P}$ for a given starting configuration S_0 of M_{orig} is captured by the automaton $\mathrm{TS}_\mathcal{P}(S_0) = (\mathcal{S}, S_0, \rightarrow, \mathcal{F}, \Sigma)$ – called the associated behavioral automaton – where $\mathcal{S}$ is the set of states, which consists of all configurations that are reachable from S_0 via one or more nodes/transitions in $\mathcal{P}$. The initial state of $\mathrm{TS}_\mathcal{P}(S_0)$ is S_0. There is a transition $S \xrightarrow{w} S'$ from $S = (M, V, \vec{A}, C, t)$ to $S' = (M', V', \vec{A}', C', t')$ in $\mathrm{TS}_\mathcal{P}(S_0)$ iff $S[M, w\rangle S'$, or $S[\rho\rangle S'$ for some transition $\rho = (M, M') \in \mathcal{E}$ and w is the empty symbol. The set of final states $\mathcal{F}$ is the set of all configurations in $\mathcal{S}$ that contain M_{dest}.

It is easy to verify that each path of $\mathrm{TS}_\mathcal{P}(S_0)$ starting from S_0 to a final state corresponds to an execution trace of $\mathcal{P}$. A configuration S is said to be reachable from S_0 on a word $w \in \Sigma^*$ with respect to $\mathcal{P}$, denoted by $S_0[\mathcal{P}, w\rangle S$, iff $S \in \mathcal{F}$ and there is a path $S_0 \xrightarrow{w_1} S_1 \xrightarrow{w_2} \cdots \xrightarrow{w_k} S_k = S$ in $\mathrm{TS}_\mathcal{P}(S_0)$ where $w = w_1 w_2 \ldots w_k$.

C. Semantics of MMA with respect to MCP

The behavior of MMA can now be described based on its associated MCP. Consider an MMA $\mathcal{A} = (\mathcal{T}, \mathcal{B}, \mathcal{M}, M_{\mathsf{in}}, \mathsf{Inv}, \Phi, Act, \mathrm{R})$. We present here the case where $\mathcal{A}$ employs a single MCP $\mathcal{P}$ for all its transitions. The results for the case where each transition σ of $\mathcal{A}$ is associated with a different MCP P can be obtained by simply substituting $\mathcal{P}_\sigma$ with P_σ where applicable.

The behavior of $\mathcal{A}$ with respect to $\mathcal{P}$ is given by a finite automaton $\mathrm{TS}_{\mathcal{A},\mathcal{P}}$ whose states are configurations of the modes in $\mathcal{M}$. The initial state of $\mathrm{TS}_{\mathcal{A},\mathcal{P}}$ is the initial configuration of M_{in} where all buffers and arrival/service patterns are empty, i.e., $S_{\mathsf{in}} = (M_{\mathsf{in}}, V_{\mathsf{in}}, \vec{A}_{\mathsf{in}}, C_{\mathsf{in}}, 0)$ where:

- For each $B \in \mathcal{B}$, $V_{\mathsf{in}}(B)$ is a vector of $\mathsf{size}(B)+1$ empty jobs $J_0 = (0,0)$ (i.e., with execution time and deadline being zeros).
- $\vec{A}_{\mathsf{in}} = \{A_1^0, \ldots, A_n^0\}$ where $A_i^0 = \emptyset$ for all $1 \leq i \leq n$.
- $C_{\mathsf{in}} = \emptyset$.

There is a transition from configuration $S = (M, V, \vec{A}, C, t)$ to configuration $S' = (M', V', \vec{A}', C', t')$ in $\Rightarrow$ on a word $w \in \Sigma^*$, written $S \overset{w}{\Rightarrow} S'$ if $S[M, w\rangle S'$ or there exists a transition $\sigma = (M, a, \varphi, I, M') \in \mathrm{R}$ such that $V \models \varphi$, $t \in I$ and $S[\mathcal{P}_\sigma, w\rangle S'$.

VI. Feasibility Analysis of MCP

As seen in the previous section, the behavior of an MMA $\mathcal{A}$ varies with respect to its associated MCPs. The choice of the MCPs depends on the nature of the multi-mode application; however, the chosen MCPs must guarantee that all tasks meet their deadlines and the buffers do not overflow. In this case, we say the MCPs are feasible for $\mathcal{A}$.

Definition 5 (Feasible configurations)**.** *A configuration $S = (M, V, \vec{A}, C, t)$ is violated iff there exists a buffer $B \in \mathcal{B}$ with $V(B) = [J_1, \ldots, J_{\mathsf{size}(B)+1}]$ such that $J_{\mathsf{size}(B)+1} \neq (0,0)$ or $J_i = (e_i, d_i)$ and $e_i > d_i = 0$ for some $1 \leq i \leq \mathsf{size}(B)$. The first condition indicates an overflow at the buffer B whereas the second condition indicates a job J_i missing its deadline. A configuration S is feasible if it is not violated.*

Definition 6 (Feasibility of MCP)**.** *An MCP $\mathcal{P}$ is said to be feasible for an MMA $\mathcal{A}$ iff (i) all configurations of $TS_{\mathcal{A},\mathcal{P}}$ are feasible, and (ii) for all transitions $S \Rightarrow S'$ in $TS_{\mathcal{A},\mathcal{P}}$ such that $S[\mathcal{P}_\sigma\rangle S'$ for some transition σ in $\mathcal{A}$, the associated behavioral automaton $TS_{\mathcal{P}_\sigma}(S)$ contains only feasible configurations.*

Observe that the above concept of feasibility of mode change protocols is developed based on the semantics of MMA. Most existing mode change protocols, however, do not explicitly define a formal model for the multi-mode systems. They assume instead that the system comprises multiple modes where it can stay in a mode arbitrarily long, mode change requests may arrive arbitrarily outside transitional intervals, and there is no cascade of mode change effects. As a result, the feasibility condition of a mode change protocol can be restricted to guaranteeing schedulability and no buffer overflows during the transitional period from an old mode to a new mode for any given pair of modes. Such restriction can reduce the complexity of the feasibility analysis.

Feasibility analysis of MCP. Given an MCP $\mathcal{P}$ and an MMA $\mathcal{A}$, one can verify if $\mathcal{P}$ is feasible for $\mathcal{A}$ by exploring the behavioral automaton $\mathrm{TS}_{\mathcal{A},\mathcal{P}}$ and checking that the conditions specified in Definition 6 are satisfied.

Recall that $\Sigma = [K_1]^n \times [K_2]$ (cf. Section V-A). Observe that each of $\mathrm{TS}_{\mathcal{A},\mathcal{P}}$ and $\mathrm{TS}_{\mathcal{P}_\sigma}(S)$ is a potentially infinite state automaton accepting (generating) sequences over the finite alphabet Σ. The decidability of the feasibility analysis follows from the fact that the associated behavioral automaton $\mathrm{TS}_{\mathcal{A},\mathcal{P}}$ ($\mathrm{TS}_{\mathcal{P}_\sigma}(S)$) can be quotiented into a *finite* state automaton which accepts the same set of finite sequences (i.e., arrival and service patterns). Given below are the main details.

Finite quotient of $\mathrm{TS}_{\mathcal{A},\mathcal{P}}$. Let U be the maximum of all the integers (different from ∞) that appear in the mode invariants and transition timing guards of $\mathcal{A}$ and $\mathcal{P}$. For instance, in the MMA in Fig. 2 and the MCP in Fig. **??** (Section **??**), $U = \max\{35, Y\}$.

Two configurations $S = (M, V, \vec{A}, C, t)$ and $S' = (M', V', \vec{A}', C', t')$ are *equivalent*, denoted as $S \approx S'$, iff (i) $M = M'$, $V = V'$, $\vec{A} = \vec{A}'$ and (ii) $t = t'$, or $t > U$ and $t' > U$. Clearly, $\approx$ partitions the infinite set of states of $\mathrm{TS}_{\mathcal{A},\mathcal{P}}$ into a finite number of equivalence classes. The number of these classes is bounded by $|\mathcal{M}| \times B_{\max}^n \times K_1^{Nn} \times K_2^N \times (U+1)$, where $B_{\max}$ is the maximum of all buffer sizes and N is the maximum length of the time intervals constrained by the arrival and service functions. The following lemma extends the classical Myhill-Nerod theorem [10] on equivalent configurations from the automata theory to the MCP model.

Lemma VI.1. *Configurations belonging to an equivalence class exhibit the same behavior in the following sense. Suppose* $S \approx S'$ *and* $S \overset{w}{\Rightarrow} S_1$ *with* $w \in \Sigma^*$, $S = (M,V,\vec{A},C,t)$, $S' = (M,V,\vec{A},C,t')$ *and* $S_1 = (M_1,V_1,\vec{A_1},C_1,t_1)$. *Then there is* $S'_1 = (M_1,V_1,\vec{A_1},C_1,t'_1)$ *such that* $S' \overset{w}{\Rightarrow} S'_1$ *and* $S_1 \approx S'_1$.

Proof: Recall that $S \overset{w}{\Rightarrow} S_1$ if $M = M_1$ and $S[M,w\rangle S_1$ with $w \in \Sigma^*$, or $M \neq M_1$ and there is a transition $\sigma = (M,a,\varphi,I,M_1) \in \mathrm{R}$ such that $V \models \varphi, t \in I$ and $S[\mathcal{P}_\sigma,w\rangle S_1$.

Case 1. Consider the first case and let $t'_1 = t' + 1$. Since $S[M,w\rangle S_1$, $t_1 = t+1 \in \mathsf{Inv}(M)$. In addition, $S \approx S'$ implies that $t = t'$ or $t > U$ and $t' > U$. If $t = t'$ then $t'_1 = t_1$ and $t'_1 \in \mathsf{Inv}(M)$, which in turn imply that $S_1 \approx S'_1$ and $S' \overset{w}{\Rightarrow} S'_1$. Otherwise, suppose $t > U$ and $t' > U$. Then, $t \in \mathsf{Inv}(M)$ implies that $\mathsf{Inv}(M) = [L,\infty]$ for some $L \leq U$. Since $t'_1 > t' > U > L$, $t'_1 \in \mathsf{Inv}(M)$. In other words, $S'[M,w\rangle S'_1$ or $S' \overset{w}{\Rightarrow} S'_1$. Furthermore, since $t_1 > t > U$ and $t'_1 > t' > U$, $S_1 \approx S'_1$.

Case 2. Next consider the case $M \neq M_1$. Then there is a transition $\sigma = (M,a,\varphi,I,M_1) \in \mathrm{R}$ such that $V \models \varphi$, $t \in I$ and $S[\mathcal{P}_\sigma,w\rangle S_1$. Thus, there exists a complete path $\eta = S \overset{w_1}{\rightarrow} \overline{S}_1 \overset{w_2}{\rightarrow} \overline{S}_2 \rightarrow \cdots \overset{w_m}{\rightarrow} \overline{S}_m = S_1$ in the associated behavioral automaton $\mathrm{TS}_{\mathcal{P}_\sigma}(S)$ of $\mathcal{P}_\sigma$, where $w = w_1 w_2 \ldots w_m$. Based on the definition of $\mathrm{TS}_{\mathcal{P}_\sigma}(S)$, we imply that $t_1 = 0$. By induction on the length m of η, we can prove that $S'[\mathcal{P}_\sigma,w\rangle S_1$. Indeed, consider the case when $m = 1$. Then, w is the empty word and $S[\rho\rangle S_1$ for some $\rho = \langle M, \delta_1, \varphi_1, Z, M_1\rangle$ in $\mathcal{P}_\sigma$ with $t \in \delta_1$. Since $S \approx S'$, either (i) $t = t'$, which implies $t' \in I$ and $t' \in \delta_1$, or (ii) $t > U$ and $t' > U$, which implies that $I = [L,\infty]$ and $\delta_1 = [L_1,\infty]$ for some $L \leq U$ and $L_1 \leq U$. In other words, $t' \in I$ and $t' \in \delta_1$. As a result, $S'[\rho\rangle S_1$ and thus, $S'[\mathcal{P}_\sigma,w\rangle S_1$. The induction step can be derived equally easily (based on the results established in this base step and in Case 1). ■

Construction of $\mathsf{REG}_{\mathcal{A},\mathcal{P}}$**:** We now define the quotiented version of $TS_{\mathcal{A}}$ to be the finite state automaton $\mathsf{REG}_{\mathcal{A},\mathcal{P}} = (\widehat{S},\Sigma,\overset{w}{\Rightarrow},\langle S_{in}\rangle)$ where:

- $\widehat{S}$ is the set of all $\langle(M,V,\vec{A},C,t)\rangle$ such that $M \in \mathcal{M}$ is a mode, V is a buffer evaluation, and $\vec{A}$ and C are finite arrival patterns of the input events and service pattern of the PE, and $t \in [U+1]$. Here, $\langle S\rangle$ is the $\approx$-equivalence class containing S. In other words, $\langle S\rangle = \{S' \mid S \approx S'\}$.
- $\langle S\rangle \overset{w}{\Rightarrow} \langle S'\rangle$ iff there exists S_1 in $\langle S\rangle$ and S'_1 in $\langle S'\rangle$ such that $S_1 \overset{w}{\Rightarrow} S'_1$, where $w \in \Sigma^*$.

Clearly, $\mathsf{REG}_{\mathcal{A},\mathcal{P}}$ can be effectively constructed using the presentation of $\mathcal{A}$ and $\mathcal{P}$. It is not difficult to show that the language of finite words accepted by $\mathsf{REG}_{\mathcal{A},\mathcal{P}}$ is exactly the words generated by $\mathrm{TS}_{\mathcal{A},\mathcal{P}}$. Using the same method, for each transition σ in $\mathcal{A}$ and each $\langle S\rangle$, we can construct a finite quotient $\mathsf{REG}_{\mathcal{P}_\sigma}(\langle S\rangle)$ of the set $\{\mathrm{TS}_{\mathcal{P}_\sigma}(S') \mid S' \in \langle S\rangle\}$.

Observe that S is feasible iff S' is feasible for all $S' \approx S$. Hence, one can analyze the feasibility of $\mathcal{P}$ when applied to $\mathcal{A}$ by checking that all the states of $\mathsf{REG}_{\mathcal{A},\mathcal{P}}$ and of $\mathsf{REG}_{\mathcal{P}_\sigma}(\langle S\rangle)$ are feasible, where σ is a transition in $\mathcal{A}$ and $\langle S\rangle \in \widehat{S}$.

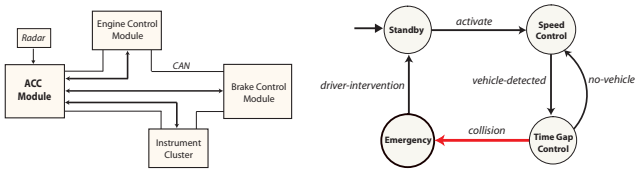
Theorem VI.2. *For any given pair of MCP* $\mathcal{P}$ *and MMA* $\mathcal{A}$, *the feasibility analysis is always decidable.*

The proof of the theorem follows directly from the finiteness of the quotient automata $\mathsf{REG}_{\mathcal{A},\mathcal{P}}$ and all $\mathsf{REG}_{\mathcal{P}_\sigma}(\langle S\rangle)$.

VII. Case Study

This section demonstrates the advantages of the MCP framework in modeling mode change behaviors via a simplified version of the Adaptive Cruise Control (ACC) system [1].

ACC is an advanced automotive feature that allows a vehicle's cruise control system to adapt the vehicle's speed to the traffic environment. An ACC vehicle has a radar attached to the front of the vehicle to detect whether slower moving vehicles are in its path. If a slower moving lead vehicle is detected, the ACC system will slow the vehicle down and control the clearance between the ACC vehicle and the lead vehicle. If the lead vehicle is no longer in the ACC vehicle's path, the ACC system will accelerate the ACC vehicle back to its set cruise control speed. During ACC operation, in case the danger of a collision is detected, the ACC system will provide a red warning light that flashes on the windshield and wait for the driver's response. It also pre-charges the brakes to prepare the vehicle for more aggressive braking to help avoid rear-end accidents. It will then move to standby mode and return the vehicle's control to the driver.



(a) ACC system architecture. (b) The multi-mode ACC Module.

Fig. 9. An Adaptive Cruise Control (ACC) System.

A typical ACC system consists of a series of interconnecting components communicating via a communication bus (e.g., Controller Area Network (CAN)) as shown in Fig. 9(a). Here, we are interested in the *ACC Module*, whose internal behavior is a multi-mode automaton depicted in Fig. 9(b).

When the *ACC Module* is turned on, it is idle at the Standby mode. Upon activation by the driver, the system enters the SpeedControl mode. In this mode, the ACC system computes the target speed based on the input vehicle speed from the *Brake Control Module*, and sends the result to the Engine Control Module. If the radar detects a lead vehicle at or within the clearance distance, the ACC Module enters the TimeGapControl mode. Here, the target speed is computed such that the set time gap between the vehicles is maintained.

Additionally, the system also computes and sends the desired acceleration/deccelaration rate to the *Brake Control Module*. Switching between SpeedControl and TimeGapControl mode is done based on whether the lead vehicle is in the clearance distance. While the system is in the TimeGapControl mode, if the danger of a collision is detected, the system enters Emergency mode where it sends driver warning messages to the *Instrument Cluster* and informs the *Brake Control*. It will then move to Standby mode and give the control back to the driver. The tasks active in each mode and their execution times and deadlines (in milliseconds), are given as follows.

- SpeedControl: Speed (5, 40), Brake (3, 15), Radar (4, 20), Weather (5, 50), Friction (5, 50).
- TimeGapControl: Speed (5, 20), Brake (3, 10), Radar (4, 20), AdjacentLane (5, 40), TimeLeft (5, 40).
- Emergency: Alarm (1, 5), Brake (2, 5), Speed (2, 5).

All tasks are periodic, with deadlines equal to periods. Their arrival functions are computed based on their periods. The tasks are scheduled under FP, where their priorities are assigned based on Rate Monotonic. The processor offers the same unit-rate service function in all the modes. To ensure safety, the following constraints are imposed on the ACC:

(C1) Tasks that are active in the Emergency mode are most critical and hence, the system enters the Emergency mode immediately if a collision is detected.

(C2) Active tasks of the TimeGapControl mode are more critical than that of the SpeedControl mode.

Given such a system, we would like to design a mode change protocol for the ACC that satisfies the above constraints. Towards this, we employ two specific protocols from two distinct classes: synchronous and asynchronous protocols. The first (P1) is the synchronous minimum offset without periodicity (Section IV-C). The second (P2) is the protocol that discards all pending events and forces the system to move immediately to the new mode. We will compare these protocols with a transition-based MCP (P3) that associates (P1) with the transition from TimeGapControl to SpeedControl, and associates (P2) with the rest of the transitions.

Table I shows the performance of the three protocols w.r.t. (i) the mode change delay when the system moves to Emergency; and (ii) the number of jobs in TimeGapControl that are missed (due to a delay in the release time) or discarded due to a mode change during normal operation. As shown in the table, the protocol (P1) violates both (C1) and (C2) constraints: it requires a long mode change delay when moving to the Emergency mode, and causes the system to miss the arrivals of jobs of the TimeGapControl mode. The protocol (P2) satisfies (C1), however, it violates (C2) as it requires the system to discard the jobs of the TimeGapControl mode. On the other hand, the proposed (P3) satisfies both constraints.

Protocol	**Delay to** Emergency	**#jobs missed**	**#jobs discarded**
P1	40 ms	11	0
P2	0	0	5
P3	0	0	0

TABLE I
PERFORMANCE OF MODE CHANGE PROTOCOLS FOR THE ACC.

The above illustrates the benefits of having different protocols for different mode transitions. In the same manner, other modules in the complete ACC system (cf. Fig. 9(a)) may also need different mode change protocols. As a result, compositions of different protocols is necessary to describe the mode change behavior of the overall system. All these requirements can be conveniently supported by MCPs.

VIII. Concluding Remarks

We have presented a unified framework for the modeling and analysis of mode change protocols in a mixed time- and event- triggered multi-mode systems. We have discussed in details the basic model for mode change protocols, its semantics when applied to the multi-mode systems, and how feasibility analysis can be done using the model. We plan to evaluate and optimize the tradeoffs among different existing mode change protocols and their combinations using our framework. Due to the state-based nature of the models, efficiency can be an issue for large systems. We would like to investigate approximation techniques that employ results from real-time calculus [6] to drive the automata abstraction and verication. Finally, we plan to build a tool that incorporates the MCP semantic framework with the compositional analysis techniques for multi-mode systems in [15] to support the component-based design of multi-mode systems.

IX. Acknowledgements

This research was supported in part by NSF CNS–0931239, NSF CNS–0930647, NSF CNS–0834524, and NSF CNS–0721541.

References

[1] Adaptive cruise control system overview. http://sunnyday.mit.edu/safety-club/workshop5/Adaptive_Cruise_Control_Sys_Overview.pdf, 2005.

[2] 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 *FORMATS*, pages 60–72, 2003.

[3] D. Bertrand, A.-M. Déplanche, S. Faucou, and O. H. Roux. A study of the aadl mode change protocol. In *ICECCS*, pages 288–293, 2008.

[4] G. C. Buttazzo, G. Lipari, and L. Abeni. Elastic task model for adaptive rate control. In *RTSS*, pages 286–295, 1999.

[5] 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.

[6] S. Chakraborty, S. Künzli, and L. Thiele. A general framework for analysing system properties in platform-based embedded system designs. In *DATE*, pages 288–293, 2003.

[7] 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.

[8] S. Goddard and X. Liu. A variable rate execution model. In *ECRTS*, pages 135–143, 2004.

[9] Q. Guangming. An earlier time for inserting and/or accelerating tasks. *Real-Time Systems*, 41(3):181–194, 2009.

[10] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2000.

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

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

[13] P. Pedro and A. Burns. Schedulability analysis for mode changes in flexible real-time systems. In *ECRTS*, pages 172–179, 1998.

[14] L. T. X. Phan, I. Lee, and O. Sokolsky. A semantic framework for mode change protocols. Technical report, University of Pennsylvania, 2011.

[15] L.T.X. Phan, I. Lee, and O. Sokolsky. Compositional analysis of multi-mode systems. In *ECRTS*, pages 197–206, 2010.

[16] 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.

[17] 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.

[18] N. Stoimenov, S. Perathoner, and L. Thiele. Reliable mode changes in real-time systems with fixed priority or edf scheduling. In *DATE*, pages 99–104, 2009.

[19] J.-P. Talpin, C. Brunette, T. Gautier, and A. Gamatié. Polychronous mode automata. In *EMSOFT*, pages 83–92, 2006.

[20] K. W. Tindell, A. Burns, and A. J. Wellings. Mode changes in priority pre-emptively scheduled systems. In *RTSS*, pages 100–109, 1992.