2018 European Control Conference (ECC)
June 12-15, 2018. Limassol, Cyprus

Exact Task Completion Time Aware Real-Time Scheduling Based on Supervisory Control Theory of Timed DES

Rajesh Devaraj, Arnab Sarkar and Santosh Biswas

***Abstract*— Real-time scheduling strategies for safety-critical applications are primarily focused towards ensuring correctness, both functional and temporal. However, in order to guarantee deadlines for critical tasks, such strategies tend to assume very conservative *Worst Case Execution Time* (WCET) estimates for tasks, thus resulting in poor resource utilization. One way of improving this situation is through the evolution of scheduling mechanisms that can reclaim resources which are provisioned for critical functionalities at design time, but remain unused at run-time. Such reclaimed resources may prove to be very useful in executing lower criticality best-effort tasks. In this work, we present a *Supervisory Control Theory of Timed Discrete Event Systems* (SCT of TDES) based formal mechanism for synthesizing non-preemptive schedulers which can recognize exact completion times of tasks at run-time and allow the processor to be immediately relinquished instead of waiting up to WCETs of tasks. This approach allows reclamation of a higher amount of resources at run-time in comparison to models which consider worst-case execution time only. Conducted experiments have shown promising results and indicate to the practical efficacy of our approach.**

I. INTRODUCTION

A system is classified as *real-time* if its notion of correctness is characterized by both logical as well as temporal constraints [1]. Applications in real-time systems are typically modeled as *recurrent* tasks. Each such task represents a piece of code which is triggered by external events in their operating environment. Each execution instance of the task is referred to as a *job*. A job's execution time is not a fixed constant, but rather varies over time depending on the input data, the processor and the computer system on which it is executed [2]. To account for such variations, a common practice is to derive upper-bounds (referred to as Worst Case Execution Time, WCET) and lower-bounds (referred to as Best Case Execution Time, BCET) estimates of task execution times. A recurrent task may be periodic, sporadic or aperiodic, based on whether consecutive jobs must be separated by a fixed inter-arrival time, a minimum inter-arrival time or arbitrary arrival times, respectively.

In recent years, researchers have shown that off-line formal approaches such as Supervisory Control Theory (SCT) of Timed Discrete Event Systems (TDES) [3] can be used to synthesize schedulers for real-time systems [4]–[12]. Chen and Wonham presented the scheduler design for *non-preemptive, periodic tasks* on uniprocessors [4]. Later, the SCT framework has been extended by Janarthanan et al. for the scheduling of *preemptive periodic tasks* [5]. Wang et al. proposed an approach for *conditionally-preemptive, real-time scheduling of periodic tasks* [9]. They have also enhanced the models presented in [4], [5] to schedule *non-preemptive, periodic tasks with multiple periods* [8]. Park and Cho developed the *preemptive scheduler for dynamically arriving sporadic tasks* [6]. Later, Park and Yang presented the preemptive scheduling scheme to allow the *co-execution of periodic and sporadic tasks* on uniprocessors with mutually exclusive accessibility of shared resources [7]. However, the scheduler synthesized in [6], [7] does not model *minimum inter-arrival times*, which is necessary for the precise characterization of sporadic tasks. Recently, this has been addressed in [10] for *non-preemptive execution*.

It may be noted that the models presented in all of the above scheduler design schemes are based on the assumption that a task will always execute up to its WCET. Thus, the schedulers synthesized using these models are unable to recognize exact task completion times and hence, cannot relinquish the processor until WCETs of the tasks are elapsed. In order to address this problem, we have developed a *TDES model for task execution* which is able to capture all possible completion times of a task between its BCET and WCET. Similarly, we have constructed TDES models for the *timing specifications* such as deadlines, periodicity etc. corresponding to each task.

Using individual TDES models for each task, we obtain their composite model. The application of standard SCT procedures [13] on this composite model results in the final scheduler which is empowered with the ability to recognize exact task completion times on-line. Our experimental studies show that the approach proposed in this work is able to reclaim significantly higher amounts of residual resources at run-time in most cases as compared to models which consider WCETs only. The *contributions* of this paper can be summarized as: *Development of execution and timing specification models* for non-preemptive, aperiodic tasks with known bounds on execution times (BCET, WCET). Later, these models have been extended to handle periodic tasks.

II. BACKGROUND THEORY

Individual system components are modeled by an automaton: $G = (Q,\ \Sigma,\ q_0,\ Q_m,\ \delta)$, where, Q is the finite set of $states$, Σ is the set of $events$, $q_0 \in Q$ is the *initial state*, $Q_m \subseteq Q$ is the set of $marked$ states, $\delta : Q \times \Sigma \longmapsto Q$ is the (partial) state *transition function*. Further, (i) $\Sigma_c \subseteq \Sigma$: the set of *controllable events*; these are the events that can be prevented by the supervisor, (ii) $\Sigma_{uc} \subseteq \Sigma$: the set of

The authors are with the Department of CSE, Indian Institute of Technology Guwahati, India. E-mail: {d.rajesh, arnabsarkar, santosh_biswas}@iitg.ernet.in. Mr. Rajesh Devaraj is partially supported through TCS research fellowship program.

uncontrollable events; these are the events that cannot be prevented by the supervisor, (iii) $\Sigma_{for} \subseteq \Sigma$: the set of *forcible events*; these are the events that can preempt a $tick$ event by the forcing action of the supervisor. The event $tick$ (t) represents the passage of one unit time of the global clock.

The *closed* behavior of TDES G is the language $L(G) = \{s \in \Sigma^* | \delta(q_0, s) \text{ is defined}\}$. The *marked* behavior of TDES G is the language $L_m(G) = \{s \in \Sigma^* | \delta(q_0, s) \in Q_m\}$. The *prefix-closure* of a language $L \subseteq \Sigma^*$ is denoted by $\overline{L}$ and $\overline{L} = \{s \in \Sigma^* : (\exists x \in \Sigma^*)\ [sx \in L]\}$. L is said to be prefix-closed if $L = \overline{L}$. G is *non-blocking* if $L_m(G)$ satisfies $\overline{L_m(G)} = L(G)$. Likewise, G is said to be *blocking* if $L(G) \neq \overline{L_m(G)}$. Let $tickcount(s)$ denote the number of $ticks$ in a string s.

Given two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ with $\Sigma = \Sigma_1 \cup \Sigma_2$ and the natural projection $P_i : \Sigma^* \rightarrow \Sigma_i^*$ defined by: (i) $P_i(\epsilon) = \epsilon$, (ii) $P_i(\sigma) = \epsilon$ if $\sigma \notin \Sigma_i$, (iii) $P_i(\sigma) = \sigma$ if $\sigma \in \Sigma_i$ and (iv) $P_i(s\sigma) = P_i(s)P_i(\sigma), s \in \Sigma^*, \sigma \in \Sigma$, the *synchronous product* of L_1 and L_2, denoted by $L_1||L_2$, is defined as $L_1||L_2 = P_1^{-1}L_1 \cap P_2^{-1}L_2$.

Formally, a *supervisory control* for G is a map $S : L(G) \rightarrow 2^{\Sigma}$. Given G and S, the resulting closed-loop system is denoted by S/G. The marked behavior of S/G is: $L_m(S/G) = L(S/G) \cap L_m(G)$. Given the system G and a specification $K \subseteq L_m(G)$, $K \neq \emptyset$, the design of supervisor requires that K must be controllable with respect to G. Let $C(K)$ denote the family of controllable sub-languages of K. $C(K)$ is always non-empty, since $\emptyset$ is controllable. $C(K)$ is closed under arbitrary set unions and has a unique largest controllable sub-language $supC(K)$ such that $supC(K) \subseteq K$. Therefore, it is possible to design a *minimally restrictive* supervisor which restricts the system behavior to the *supremal controllable sub-language* of K, denoted by $supC(K)$. If this supervisor is adjoined with G, then $L_m(S/G) = supC(K)$.

III. Proposed Scheduler Synthesis Framework

A. Aperiodic Tasks with Known Arrival Times

System Model: We consider a real-time system consisting of a set I $(= \{\tau_1, \tau_2, ..., \tau_n\})$ of n (≥ 1) non-preemptive, aperiodic tasks to be scheduled on a uniprocessor. We consider the scheduling of a single instance of an aperiodic task τ_i whose arrival time is known. Formally, τ_i is represented by a quadruple $\langle A_i, B_i, W_i, D_i \rangle$, where A_i is the *arrival time*, B_i is the *Best Case Execution Time*, W_i is the *Worst Case Execution Time* and D_i is the *relative deadline* of τ_i.

Assumptions: (1) Tasks are independent. (2) Tasks cannot suspend itself until it completes. (3) Only one task is allowed to execute on the processor at a time (i.e., *resource constraint*). (4) $B_i \leq W_i \leq D_i$ for all tasks in I.

1) Task execution model: The TDES model T_i for executing a non-preemptive task $\tau_i \langle A_i, B_i, W_i, D_i \rangle$ on a uniprocessor is shown in Figure 1 and is formally defined as:

$$T_i = (Q_i,\ \Sigma,\ \delta_i,\ q_0,\ Q_m),$$

where, $Q_i = \{0, 1, ..., 10\}$, $q_0 = 0$, $Q_m = \{10\}$, $\Sigma = \cup_{i=1}^{n} \Sigma_i \cup \{t\}$, where $\Sigma_i = \{a_i, s_i, c_i\}$, a_i: arrival of a task τ_i, s_i: start of execution of τ_i, c_i: completion of

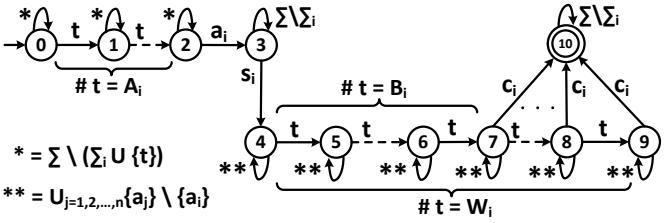


Fig. 1: Execution model T_i for task $\tau_i \langle A_i, B_i, W_i, D_i \rangle$

execution of τ_i. The events are categorized as follows: (i) $\Sigma_{uc} = \cup_{i=1}^{n} \{a_i, c_i\}$, (ii) $\Sigma_c = \cup_{i=1}^{n} \{s_i\}$, (iii) $\Sigma_{for} = \Sigma_c$. Here, a_i is modeled as *uncontrollable* event since the arrival events are induced by the environment. In addition, c_i is also modeled as *uncontrollable* since the task τ_i may complete its execution anywhere in between B_i and W_i. The starting of τ_i's execution is under the control of supervisor (i.e., scheduler) and hence, s_i is modeled as *controllable*. These controllable events are also modeled as *forcible* events.

The initial state of T_i, i.e., *State 0* represents the state in which task τ_i resides prior to the start of the system. The self-loop $\Sigma \setminus (\Sigma_i \cup \{t\})$ contains the events such as arrival, start of execution, and completion with respect to any other task $(\tau_j \in I, j \neq i)$ in the system. Thus, $\Sigma \setminus (\Sigma_i \cup \{t\})$ restricts only the execution of τ_i and does not impose any restriction on the execution of other tasks. In order to measure the occurrence of A_i ticks subsequent to system start, *State 0* is replicated for A_i times. After the occurrence of a_i at $A_i{}^{th}$ tick, T_i reaches *State 3* from *State 2*. At *State 3*, the scheduler takes a decision whether to immediately allocate the task τ_i for execution on a processor or make it wait on the ready queue. The latter is indicated by the self-loop transition $\Sigma \setminus \Sigma_i$ at *State 3*. If the scheduler decides to start the execution of τ_i, then it will enable the event s_i to assign τ_i for execution on the processor.

According to Figure 1, if event s_i occurs at *State 3*, then T_i reaches *State 4*. Here, the self-loop $\cup_{j=1,2,...,n}\{a_j\} \setminus \{a_i\}$ ensures that only arrivals (barring that of τ_i) but not the execution of tasks other than τ_i are allowed (satisfies Assumption 3). After the elapse of B_i (respectively, W_i) ticks from *State 4*, T_i reaches *State 7* (respectively, *State 9*). To model the possibility that τ_i may complete its execution any time in between B_i and W_i ticks, the outgoing transition on completion event c_i is added at all states starting from *State 7* to *State 9*. These outgoing transitions on c_i are leading to *State 10*. After executing c_i, T_i reaches and stays at *State 10* without imposing any constraint on the other tasks currently executing on the processor.

Remark 1: *The marked behavior of* T_i: $L_m(T_i) = x_1 a_i x_2 s_i x_{i1} t x_{i2} \ldots t x_{iB_{i-1}} t(x_{iB_i} + x_{iB_i} t x_{iB_{i+1}} + \cdots + x_{iB_i} t \ldots t x_{iW_i}) c_i x_3$, where $x_1, x_2, x_3 \in (\Sigma \setminus \Sigma_i)^*$, $tickcount(x_1) = A_i$, $x_{i,j} \in (\cup_{k=1,2,...,n}\{a_k\} \setminus \{a_i\})^*$ and $j = 1, 2, \ldots, W_i$. □

Definition 1: *Deadline-meeting sequence:* A sequence $x = x_1 a_i x_2 c_i x_3 \in L_m(T_i)$, where $x_1, x_3 \in \Sigma^*$, $x_2 \in (\Sigma \setminus \{c_i\})^*$ is *deadline-meeting*, if $tickcount(x_2) \leq D_i$. Otherwise, x is a deadline-missing sequence. □

Proposition 1: $L_m(T_i)$ *contains (i) deadline-meeting, (ii) deadline-missing and (iii) resource constraint satisfying sequences for task* τ_i.

Proof: Let us consider the formal representation of $L_m(T_i)$ given in Remark 1 as $seq = x_1 a_i x_2 s_i x_4 c_i x_3$, where $x_4 = x_{i1} t x_{i2} \ldots t x_{iB_{i-1}} t(x_{iB_i} + x_{iB_i} t x_{iB_{i+1}} + \cdots + x_{iB_i} t \ldots t x_{iW_i})$. In seq, *tickcount*$(x_4) = W_i$ ($\leq D_i$, Assumption 4). However, *tickcount*$(a_i x_2 s_i x_4 c_i)$ may be greater than D_i depending on *tickcount*(x_2) (say y) since $x_2 \in (\Sigma \setminus \Sigma_i)^*$ contains $tick$ (t) in it. Hence, seq is *deadline-meeting* if $(y + W_i) \leq D_i$. Otherwise, seq is *deadline-missing*. From the transition structure of T_i (Figure 1), it may be observed that this model never allows more than one task to execute on the processor at the same time. Hence, $L_m(T_i)$ contains (i) deadline-meeting, (ii) deadline-missing and (iii) resource constraint satisfying execution sequences for τ_i. □

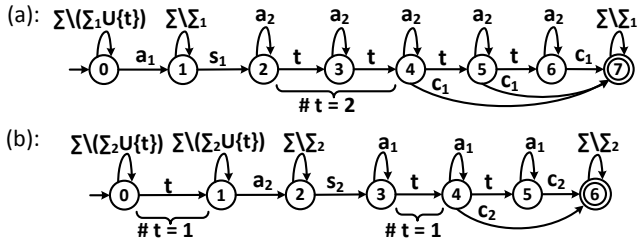


Fig. 2: (a) T_1 for τ_1, (b) T_2 for τ_2.

Example: Let us consider an example uniprocessor system consisting of two tasks $\tau_1\langle 0, 2, 4, 7\rangle$, $\tau_2\langle 1, 1, 2, 4\rangle$. The task execution models T_1 for τ_1 and T_2 for τ_2 are shown in Figures 2(a) and 2(b), respectively. □

Based on the above approach, we construct the TDES models for all the n real-time tasks in the system. Given n individual TDES task models T_1, T_2, ..., T_n, a *synchronous product* composition $T = ||_{i=1}^{n} T_i$ on the models gives us the composite model for all the tasks executing concurrently. Since $L_m(T) = \cap_{i=1}^{n} L_m(T_i)$, the marked language $L_m(T)$ *represented by the composite task execution model includes (i) deadline-meeting, (ii) deadline-missing and (iii) resource constraint satisfying execution sequences for all tasks in* I.

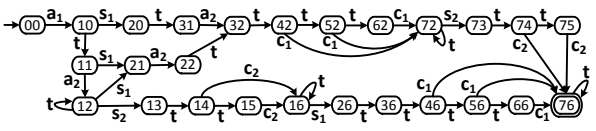


Fig. 3: The composite task execution model $T = T_1||T_2$

Example (continued): Figure 3 shows the composite model T (= $T_1||T_2$). In T, State XY (X = 0 to 7, Y = 0 to 6) represents state X of T_1 and state Y of T_2. Let us consider the sequence $seq_1 = a_1 s_1 t a_2 t t t c_1 s_2 t t c_2$ from *State* 00. In this sequence, τ_1 arrives and starts execution before τ_2. Once τ_1 completes its execution, then τ_2 starts and completes its execution. Here, τ_1 meets its deadline, i.e., $tickcount(a_1 s_1 t a_2 t t t c_1) = 4 \leq 7$. However, τ_2 misses its deadline since $tickcount(a_2 t t t c_1 s_2 t t c_2) = 5$ which is greater than its deadline 4. Hence, seq_1 is a *deadline-missing* sequence. Now, let us consider another sequence $seq_2 = a_1 t a_2 s_2 t t c_2 s_1 t t t t c_1$. Here, both τ_1 ($tickcount(a_1 t a_2 s_2 t t c_2 s_1 t t t t c_1) = 7 \leq 7$) and τ_2 ($tickcount(a_2 s_2 t t c_2) = 2 \leq 4$) meet their deadlines. Hence, seq_2 is a *deadline-meeting* sequence. □

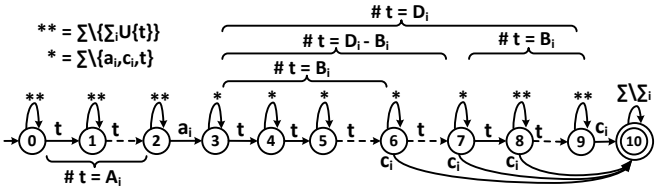


Fig. 4: Timing specification H_i for an aperiodic task τ_i.

2) A TDES model of timing specification: Once activated, a task τ_i *has to finish its execution within its relative deadline*. Such a specification is modeled using TDES H_i (shown in Figure 4). At the $A_i{}^{th}$ tick subsequent to system start (*State 0*), the arrival of τ_i is allowed at *State* 2 (i.e., a_i). After the occurrence of arrival event a_i, H_i reaches *State* 3. The self-loop $\Sigma \setminus \{a_i, c_i, t\}$ in *State 3* is used to model the fact that task τ_i is allowed to only execute event s_i associated with it, however, without imposing any restriction with respect to other tasks in the system. After the elapse of one tick event, H_i reaches *State* 4 in which the self-loop is similar to that in *State* 3. Since τ_i is not allowed to execute event c_i before the elapse of B_i ticks (because at least B_i time ticks must be incurred before signaling the completion of τ_i), states that are similar to *State* 3 and *State* 4 are instantiated B_i times starting from *State* 3. Following that, at *State* 6, task τ_i is allowed to execute c_i in addition to s_i because τ_i is allowed to complete after executing for B_i ticks subsequent to arrival.

It may be noted that if τ_i does not start its execution at least B_i ticks before its deadline, then τ_i is guaranteed to miss its deadline. Hence, s_i is allowed only up to $D_i - B_i$ ticks from the arrival of τ_i which is captured in the self-loop transitions present at states 3 to 6. When the time remaining before deadline is less than B_i ticks, s_i is disallowed which is captured by the self-loop transition $\Sigma \setminus \{\Sigma_i \cup \{t\}\}$ present at states 8 to 9. Since, the task τ_i is not allowed to start/continue its execution after D_i ticks from the arrival of its current instance, *the model* H_i *correctly captures the deadline requirement of* τ_i. Based on the above approach, H_i is constructed for all other tasks in the system. Then we perform synchronous product composition to obtain the composite deadline specification model $H = ||_{i=1}^{n} H_i$.

Remark 2: (i) $L_m(H)$ *represented by the composite timing specification model includes only deadline-meeting execution sequences for all tasks*. (ii) It may be observed that H_i does not restrict the simultaneous execution of multiple tasks on the processor. Hence, $L_m(H)$ contains the deadline-meeting sequences that may violate resource constraint. □

Example (continued): The specification models H_1 for τ_1, H_2 for τ_2 and the composite specification model H (= $H_1||H_2$) are shown in Figures 5(a), 5(b) and 5(c), respectively. As mentioned earlier, model H represents all

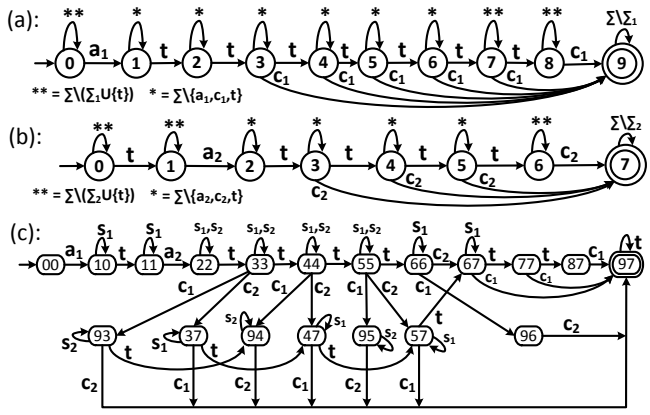


Fig. 5: (a) H_1 for τ_1, (b) H_2 for τ_2, (c) $H = H_1||H_2$.

possible deadline-meeting sequences for τ_1 and τ_2. In order to illustrate this fact, let us try to find the deadline-missing sequence seq_1 (= $a_1 s_1 t a_2 t t t c_1 s_2 t t c_2$) in the composite model H shown Figure 5(c). After proceeding through the states $\langle 00\rangle \xrightarrow{a_1} \langle 10\rangle \xrightarrow{s_1} \langle 10\rangle \xrightarrow{t} \langle 11\rangle \xrightarrow{a_2} \langle 22\rangle \xrightarrow{t} \langle 33\rangle \xrightarrow{t} \langle 44\rangle \xrightarrow{t} \langle 55\rangle \xrightarrow{c_1} \langle 95\rangle \xrightarrow{s_2} \langle 95\rangle$, the composite model H *gets blocked* due to the absence of a transition on t at State 95. Thus, $L_m(H)$ does not contain the deadline-missing sequence seq_1. The deadline-meeting sequence seq_2 (= $a_1 t a_2 s_2 t t c_2 s_1 t t t c_1$) can be retrieved by tracing the states $\langle 00\rangle \xrightarrow{a_1} \langle 10\rangle \xrightarrow{t} \langle 11\rangle \xrightarrow{a_2} \langle 22\rangle \xrightarrow{s_2} \langle 22\rangle \xrightarrow{t} \langle 33\rangle \xrightarrow{t} \langle 44\rangle \xrightarrow{c_2} \langle 47\rangle \xrightarrow{s_1} \langle 47\rangle \xrightarrow{t} \langle 57\rangle \xrightarrow{t} \langle 67\rangle \xrightarrow{t} \langle 77\rangle \xrightarrow{t} \langle 87\rangle \xrightarrow{c_1} \langle 97\rangle$. Hence, $seq_2 \in L_m(H)$. □

3) Supervisor Synthesis: In order to find all deadline-meeting sequences in $L_m(T)$, we construct $M = T||H$.

Theorem 1: *$L_m(M)$ contains only and all the deadline-meeting periodic sequences of $L_m(T)$.*

Proof: We know that $L_m(T)$ contains both deadline-meeting as well as deadline-missing sequences (Proposition 1). On the other hand, $L_m(H)$ contains only and all possible deadline-meeting sequences (Remark 2). Since $L_m(M) = L_m(T) \cap L_m(H)$, $L_m(M)$ contains *only* and *all* the deadline-meeting sequences of $L_m(T)$. It may be noted that this composition also eliminates the sequences in $L_m(H)$ that violate resource constraint. □

Although $L_m(M)$ contains only and all sequences sequences that satisfy both deadline as well as resource constraint, the sequences that violate (i) deadlines in $L_m(T)$ and (ii) resource constraint in $L_m(H)$, lead to *deadlock* states in M. In order to transform M such that it becomes both *controllable* and *non-blocking*, we apply the standard *safe state synthesis* mechanism presented in Algorithm 1 [13] to remove the minimal number of states from M such that it becomes both *controllable* and *non-blocking*.

Example (continued): Figure 6 shows the initial supervisor candidate M. Here, *State $WXYZ$* represents *State W* of T_1, *State X* of T_2, *State Y* of H_1 and *State Z* of H_2. Algorithm 1 invokes Algorithm 2 which initializes

Algorithm 1: SAFE STATE SYNTHESIS

Input: $M(= Q, \Sigma, \delta, q_0, Q_m)$
Output: The set of reachable safe states Q_s of M
Set of safe states, $Q_s \leftarrow \emptyset$;
Set of un-safe states, $Q_u \leftarrow \emptyset$;
$i \leftarrow 0$; $Q_u^i \leftarrow Q_u$;
repeat
  $i \leftarrow i + 1$;
  $Q' \leftarrow$ CO-REACHABLE (M, Q_u^{i-1});
  $Q'' \leftarrow$ NON-CO-REACHABLE (M, Q');
  $Q_u^i \leftarrow Q_u^{i-1} \cup Q''$;
until $Q_u^i = Q_u^{i-1}$;
$Q_u \leftarrow Q_u^i$;
$i \leftarrow 0$; $SQ_i \leftarrow q_0$; {FORWARD-REACHABILITY}
repeat
  $i \leftarrow i + 1$;
  $SQ_i \leftarrow (SQ_{i-1} \cup \texttt{Image}(SQ_{i-1}, \delta)) \setminus Q_u$;
until $SQ_i = SQ_{i-1}$;
return $Q_s = SQ_i$;

Algorithm 2: CO-REACHABLE

Input: M, Q_u
Output: The set of co-reachable states of M
$i \leftarrow 0$; $SQ_i \leftarrow Q_m \setminus Q_u$;
repeat
  $i \leftarrow i + 1$;
  $SQ_i \leftarrow (SQ_{i-1} \cup \texttt{PreImage}(SQ_{i-1}, \delta)) \setminus Q_u$;
until $SQ_i = SQ_{i-1}$;
return SQ_i;

Algorithm 3: NON-CO-REACHABLE

Input: M, The set of co-reachable states Q'
Output: The set of non-co-reachable states
$i \leftarrow 0$; $SQ_i \leftarrow Q \setminus Q'$;
repeat
  $i \leftarrow i + 1$;
  $Q^{uc} \leftarrow \texttt{PreImage\_UC}(SQ_{i-1}, \delta)$;
  $Q^t \leftarrow \texttt{Undef\_for}(\texttt{PreImage\_t}(SQ_{i-1}, \delta), \delta)$;
  $SQ_i \leftarrow (SQ_{i-1} \cup Q^{uc} \cup Q^t)$;
until $SQ_i = SQ_{i-1}$;
return SQ_i;

$SQ_0 = Q_m = \{7697\}$. Now, PreImage (SQ_0, δ)[1] returns {7494, 7595, 7496, 7596, 7495, 6687, 5667, 6677, 5677, 4657} and gets added to SQ_1 along with {7697}. In a similar manner, Algorithm 2 continues to iterate until $i = 11$, where a fix-point is reached. This returns the set of co-reachable states, Q' = {0000, 1010, 1111, 1222, 1322,

[1]The operator PreImage(V, δ) computes the set of states that, by one transition, can reach a state in V ($\subseteq Q$), formally defined as: PreImage$(V, \delta) = \{q \in Q | \exists q' \in V : \delta(q, \sigma) = q', \sigma \in \Sigma\}$. PreImage_UC$(V, \delta) = \{q \in Q | \exists q' \in V : \delta(q, \sigma) = q', \sigma \in \Sigma_{uc}\}$. PreImage_t$(V, \delta) = \{q \in Q | \exists q' \in V : \delta(q, t) = q'\}$. Image$(V, \delta) = \{q' \in Q | \exists q \in V : \delta(q, \sigma) = q', \sigma \in \Sigma\}$. Undef_for$(V, \delta) = \{q \in V | \forall \sigma \in \Sigma_{for} : \delta(q, \sigma)$is undefined$\}$.

1433, 1637 2637, 3647, 4657, 5667, 6677, 1544, 1647, 2647, 3657, 4667, 5677, 6687, 2111, 2222, 3233, 4244, 7294, 7394, 7495, 7596, 5255, 7295, 7395, 7496, 2010, 3111, 3222, 4233, 5244, 7293, 7393, 7494, 7595, 7697}.

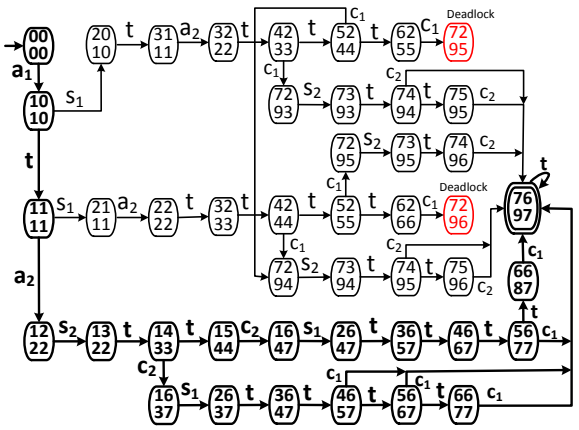


Fig. 6: $M = T||H$.

Next, Algorithm 1 invokes Algorithm 3 which first computes the initial set of non-co-reachable states, $SQ_0 = Q \setminus Q'$ = {7295, 6255, 7296, 6266}. Then, Q^{uc} is computed and its $\emptyset$ since there is no incoming transition on uncontrollable event to any state in SQ_0. However, Q^t = {5244, 5255} since both these states contain outgoing transition on t to a state that belongs to SQ_0 as well as they do not contain any outgoing transition on forcible event. Hence, SQ_1 = $SQ_0 \cup Q^t$ at the end of first iteration. By continuing further, SQ_5 = {2010, 2111, 3111, 2222, 3222, 3233, 4233, 4244, 5244, 5255, 6255, 6266, 7295, 7296}. Subsequently, Algorithm 3 reaches fix-point at $i = 6$ which implies $SQ_6 = SQ_5$ and $Q'' = SQ_6$.

Finally, Algorithm 1 invokes reachability process which starts with SQ_0 = 0000 (the initial state of M) and adds all reachable states excluding Q''. Now, `Image` (SQ_0, δ) returns 1010 which makes $SQ_1 = \{0000, 1010\}$. Since, *State* 2010 belongs to Q'', $SQ_2 = \{0000, 1010, 1111\}$. Similarly, it continues to iterate until $i = 14$, where fix-point is reached. Finally, it returns the set of safe states Q_s = {0000, 1010, 1111, 1222, 1322, 1433, 1544, 1637, 1647, 2637, 2647, 3647, 3657, 4657, 4667, 5667, 5677, 6677, 6687, 7697}. Since Q_s is non-empty, the given task set is schedulable. Next, the scheduler S is constructed using Q_s which ensures that both τ_1 and τ_2 meet their deadlines. The scheduler S is shown using thick lines in Figure 6. It can be observed that the number of fix-point iterations of Algorithms 2 and 3 is bounded by the number of states in M, and each such iteration has a complexity that is linear in the size of M. It follows that the complexity of Algorithm 1 is polynomial in the size of M.

B. Handling Arbitrary Aperiodic Task Arrivals

In order to design a scheduler to support arbitrary arrivals of task τ_i, T_i presented in Figure 1 has been modified. Here, T_i stays at *State 0* until the arrival of τ_i (i.e., a_i) by executing the events in the self-loop $\Sigma \setminus \Sigma_i$. The remaining part of T_i

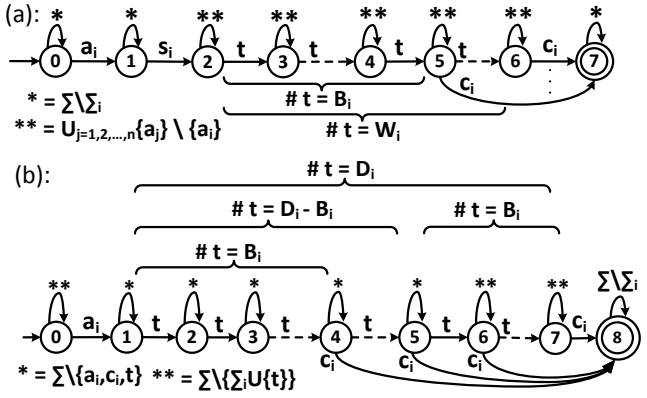


Fig. 7: (a) Task execution model T_i and (b) Timing specification model H_i for dynamically arriving aperiodic task.

(i.e., from a_i to c_i) is same as Figure 1. Similar changes have been made to H_i in Figure 4. The modified H_i for dynamically arriving aperiodic tasks is shown in Figure 7(b).

C. Handling Periodic Tasks

It may be noted that *the periodic task τ_i consists of an infinite sequence of instances that are separated by a fixed inter-arrival time P_i* [1]. Formally, a dynamically arriving periodic task τ_i is represented by a four tuple $\tau_i\langle B_i, W_i, D_i, P_i\rangle$, where B_i is the BCET, W_i is the WCET, D_i is the deadline and $P_i(\geq D_i)$ the fixed inter-arrival time.

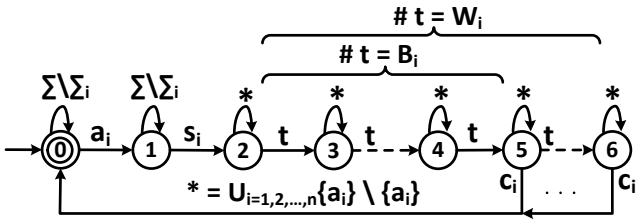


Fig. 8: T_i for periodic task $\tau_i\langle B_i, W_i, D_i, P_i\rangle$.

Presently, the model T_i shown in Figure 7(a) moves to *State 7* after the completion of a single instance of τ_i and stays there indefinitely. This must be modified to support the execution of periodic tasks. Figure 8 shows the modified execution model T_i corresponding to the dynamically arriving periodic task τ_i. With respect to T_i shown in Figure 7(a) for the execution of aperiodic job, we have introduced a loop-back to the initial state (*State 0*) from *State 5* to *State 6* on completion event c_i such that the execution of infinite sequences can be supported. In addition to this, we set *State 0* as initial as well as marked state which implies that T_i stays at *State 0* prior to the arrival of next instance of τ_i as well as after the completion of its current instance.

Figure 9 shows the timing specification model H_i of a dynamically arriving, periodic task τ_i. This model includes the set of states $\{8, 9, ..., 13\}$ in addition to the states that are already present in H_i (shown in Figure 7(b)) to capture the fixed inter-arrival time constraint. In order to model the infinite sequence of identical instances, a loop-back on

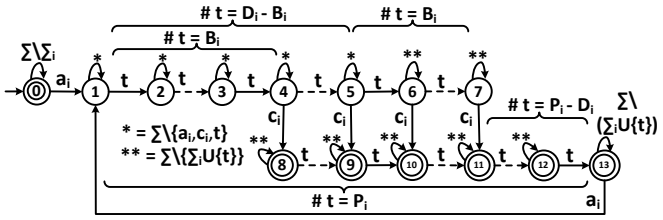
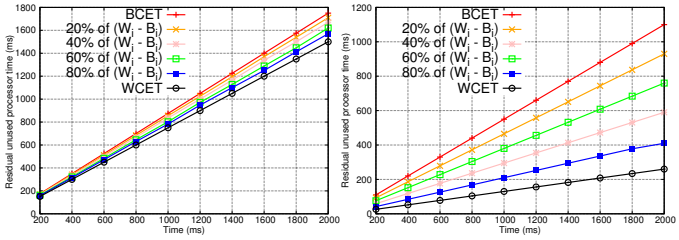


Fig. 9: H_i for periodic task $\tau_i\langle B_i, W_i, D_i, P_i\rangle$.



(a) Aperiodic tasks. (b) Periodic tasks.

Fig. 10: Idle slots reclaimed during on-line execution.

completion event c_i (from states $\{4, 5, ..., 7\}$) to *State 1* of H_i have been added via states $\{8, 9, ..., 13\}$. $L_m(H_i)$ contains the deadline-meeting, periodic sequences. Then, we can construct the composite models using the above individual models to obtain the initial supervisor candidate M. By using Algorithm 1, we can obtain the scheduler S which controls the task execution on-line.

D. Application Example

Let us consider an *Instrument Control System* (ICS) which is used for collecting and analyzing diagnostic information of the electrical and electronic components in a modern automotive system [14]. This system is modeled as a collection of five real-time tasks ($\tau_i\langle B_i, W_i, D_i\rangle$): Mode management $\tau_1\langle 10, 15, 50\rangle$, Mission data management $\tau_2\langle 5, 12, 100\rangle$, Instrument monitoring $\tau_3\langle 3, 8, 40\rangle$, Instrument configuration $\tau_4\langle 5, 10, 200\rangle$ and Instrument processing $\tau_5\langle 2, 4, 20\rangle$. Figure 10 depicts the total amount of residual unused processor time (denoted by RUT (ms)) that our proposed scheduling mechanism is able to reclaim as the schedule progresses (x-axis denotes time). Each plot in the figure is obtained for a distinct expected execution time for the five tasks considered. A plot labeled $x\%$ of $(W_i - B_i)$ refers to the scenario when the expected execution times for all tasks τ_i is given by: $B_i + x\%(W_i - B_i)$.

In Figure 10a, all tasks have been modeled as aperiodic with random inter-arrival times between consecutive jobs. From the figure, it may be observed that at any time instant subsequent to system start, RUT is inversely proportional to the expected execution time corresponding to the plot scenario. For example, at time 1400 ms, the value of RUT is 1200 for shorter expected execution times of the tasks (20% of $(W_i - B_i)$), as compared to 1100 when expected execution times are relatively higher (80% of $(W_i - B_i)$). It may be noted that all state-of-the-art TDES based scheduler synthesis mechanisms such as [4], [5], [9], consider only WCETs of the tasks. Hence, these schemes will only be able to reclaim unused processor times proportional to those in the plots labeled WCET in Figure 10 irrespective of actual task completion times. In Figure 10b, we consider all tasks to be periodic in nature and persistent throughout the schedule length. The average load on the processor in the scenario under consideration is much higher than for the aperiodic tasks considered in Figure 10a. It may be observed that although the trends corresponding to the plots in both figures are same, the RUT values for the periodic case are lower than those for aperiodic tasks.

IV. Conclusion

We have developed TDES models for task execution and deadline specification which are able to capture all possible completion times of a task between its BCET and WCET. Using these individual models, we have presented a scheduler synthesis mechanism. Through experiments, we have shown that the resulting scheduler allows reclamation of a higher amount of system resources at run-time in comparison to models which consider worst-case execution time only.

References

[1] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer, 2011, vol. 24.

[2] Wilhelm *et al.*, "The worst-case execution-time problem-overview of methods and survey of tools," *ACM TECS*, vol. 7, no. 3, p. 36, 2008.

[3] B. A. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–342, 1994.

[4] P. C. Chen and W. M. Wonham, "Real-time supervisory control of a processor for non-preemptive execution of periodic tasks," *Real-Time Systems*, vol. 23, no. 3, pp. 183–208, 2002.

[5] V. Janarthanan, P. Gohari, and A. Saffar, "Formalizing real-time scheduling using priority-based supervisory control of discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 51, no. 6, pp. 1053–1058, 2006.

[6] S.-J. Park and K.-H. Cho, "Real-time preemptive scheduling of sporadic tasks based on supervisory control of discrete event systems," *Information Sciences*, vol. 178, no. 17, pp. 3393–3401, 2008.

[7] S.-J. Park and J.-M. Yang, "Supervisory control for real-time scheduling of periodic and sporadic tasks with resource constraints," *Automatica*, vol. 45, no. 11, pp. 2597–2604, 2009.

[8] X. Wang, Z. Li, and W. M. Wonham, "Dynamic multiple-period reconfiguration of real-time scheduling based on timed DES supervisory control," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 1, pp. 101–111, 2016.

[9] ——, "Optimal Priority-Free Conditionally-Preemptive Real-Time Scheduling of Periodic Tasks Based on DES Supervisory Control," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. PP, no. 99, pp. 1–17, 2016.

[10] R. Devaraj, A. Sarkar, and S. Biswas, "Real-time scheduling of non-preemptive sporadic tasks on uniprocessor systems using Supervisory Control of timed DES," in *American Control Conference (ACC)*, pp. 3212–3217, 2017.

[11] ——, "Fault-Tolerant Preemptive Aperiodic RT Scheduling by Supervisory Control of TDES on Multiprocessors," *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 3, pp. 87:1–87:25, 2017.

[12] ——, "Fault-Tolerant Scheduling of Non-preemptive Periodic Tasks using SCT of Timed DES on Uniprocessor Systems," *IFAC-PapersOnline*, vol. 50, no. 1, pp. 9315–9320, 2017.

[13] S. Miremadi and B. Lennartson, "Symbolic on-the-fly synthesis in supervisory control theory," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 5, pp. 1705–1716, Sept 2016.

[14] R. M. Pathan, "Real-time scheduling algorithm for safety-critical systems on faulty multicore environments," *Real-Time Systems*, vol. 53, no. 1, pp. 45–81, 2017.