Real-time preemptive scheduling of sporadic tasks based on supervisory control of discrete event systems

Seong-Jin Park\textsuperscript{a}, Kwang-Hyun Cho\textsuperscript{b,*}

\textsuperscript{a}Department of Electrical and Computer Engineering, Ajou University, Suwon 443-749, Republic of Korea
\textsuperscript{b}Department of Bio and Brain Engineering and KI for the BioCentury, Korea Advanced Institute of Science and Technology (KAIST), 335 Gwahangno, Yuseong-gu, Daejeon 305-701, Republic of Korea

\textbf{Abstract}

This paper presents a preemptive scheduling scheme for real-time systems with sporadic tasks based on the supervisory control theory of discrete event systems. In particular, we present a systematic method of computing a schedulable language that includes all achievable sequences that meet the given deadlines of accepted sporadic tasks. A supervisor that achieves the schedulable language corresponds to a scheduler that can secure the deadlines of all accepted tasks. We further show that the schedulable language includes the decisions on whether a scheduler accepts or rejects a newly arrived sporadic task.

\textcopyright{} 2008 Elsevier Inc. All rights reserved.

\section{1. Introduction}

In real-time systems executing sporadic tasks, a scheduler accepts a newly arrived sporadic task if it can schedule the task to be completed in time without causing any deadlines to be missed for the previously accepted sporadic tasks. In this case, the scheduling problems emerge as follows: how to develop schedulability (or acceptance) test algorithms and how to schedule the accepted tasks. One non-optimal schedulability test algorithm\cite{10} is if the total density (or workload) of the previously accepted sporadic tasks and a newly arrived sporadic task is less than or equal to 1, then the newly arrived task is accepted and scheduled following the earliest deadline first (EDF) algorithm in deadline-driven systems. More improved algorithms have been developed based on slack computation\cite{10,13} and a synthetic utilization level\cite{1}. In particular, under the separation constraint for task arrival, an optimal schedulability test (offline) algorithm for a sporadic task system has been proposed based on EDF\cite{3}. Additionally, for arbitrarily arriving sporadic tasks, an optimal schedulability test (online) algorithm for active sporadic tasks has been proposed also based on EDF\cite{5}.

In this paper, we present a new method of designing preemptive schedulers for sporadic tasks based on a supervisory control theory of discrete event systems (DESs). The supervisory control theory\cite{15} has been used to analyze and control various DESs including HIV/AIDS\cite{9}, operating systems\cite{12}, and reactive software\cite{14}. In particular, it has been shown that the supervisory control theory can be successfully applied to the design of real-time schedulers\cite{2,6–11}. In\cite{2}, a scheduler design method based on timed automata and state feedback control was presented. A design method for the non-preemptive scheduling of periodic tasks with hard deadlines was reported in\cite{6}, and a scheduler synthesis method for periodic tasks with fixed priorities was reported in\cite{7}. The scheduling problem for multiprocessors executing periodic tasks with fixed...
release times has also been addressed in [8]. In [11], a scheduling method for the non-preemptive execution of periodic tasks in a dynamically reconfigurable device was presented. In this paper, we also consider a scheduling problem based on the supervisory control theory of DESs. However, while previous studies primarily addressed the scheduling problems of periodic tasks with fixed release times, we consider the following two scheduling problems for sporadic tasks with arbitrary release times: how to test the schedulability of the newly arrived sporadic tasks and how to schedule the accepted sporadic tasks.

The scheduler presented in this paper is a feedback controller (called a supervisor) that controls systems by enabling or disabling controllable events based on the observation of a sequence of events. For instance, when a sporadic task first arrives, the scheduler decides whether it enables or disables the acceptance event of the task based on the given deadlines. The specific goal of the scheduler is to achieve a schedulable language composed of all achievable sequences that meet the deadlines of the accepted sporadic tasks. This paper provides a systematic method of computing such a schedulable language. A scheduler can be designed in an offline manner based on the schedulable language, and the designed scheduler can control the execution of sporadic tasks in an online manner upon observation of the executed sequences. We further show that the schedulable language includes all achievable sequences that meet the deadlines of the accepted tasks, the resulting acceptance decisions based on the language always secure the deadlines of all previously accepted sporadic tasks, as well as the newly arrived sporadic tasks.

As mentioned by Chen and Wonham [6], the supervisory control approach adopted in this paper for real-time scheduling differs from the conventional real-time scheduling approaches in the following aspects. First, the supervisory control approach does not include the problems of finding the schedulability conditions and realizing the specific scheduling algorithms such as EDF. This is because the supervisory control approach systematically finds all achievable execution sequences that meet the task deadlines. Hence, the proposed method is more concise in formulating the scheduler design problem and also more efficient in designing the schedulers. Second, the conventional scheduling algorithms usually result in only partial execution sequences that meet the task deadlines. However, the schedulable language presented in this paper is always complete because it contains all achievable execution sequences that meet the task deadlines in a system. This issue of completeness has not yet been addressed in conventional real-time scheduling approaches. In this respect, the proposed approach can establish a new method of systematically optimizing the task execution. For instance, the proposed method allows the efficient selection of all possible sequences satisfying additional criteria, such as a deadline-matching property (i.e. tasks should be completed as close as possible to their deadlines).

A convincing motivation for exploiting supervisory control can be found in the following facts. First, the supervisory control theory is a formal method and, therefore, it can provide a systematic approach to a given scheduling problem for a complete solution. Thus, it is a rigorous tool for the analysis and synthesis of real-time systems that satisfy the given performance specifications. In addition, supervisory control is the feedback control of a dynamic system, so it can achieve a more predictable and adaptive behavior with a guaranteed performance for real-time systems in highly dynamic environments.

2. Supervisory control of timed DESs

2.1. A timed DES and its behavior

In this paper, we adopt the supervisory control framework of timed DESs (TDESs) presented in [4]. A TDES $G$ is represented by the following finite state automaton:

$$G = (Q, \Sigma, q_0, \delta, Q_m),$$

where $Q$ is a finite set of states, $\Sigma$ is a set of events, $q_0$ is the initial state, $\delta : Q \times \Sigma \rightarrow Q$ is the state transition (partial) function, and $Q_m \subseteq Q$ is a set of marked states representing the completion of the tasks. The event set, $\Sigma$, is composed of two subsets:

$$\Sigma = \Sigma_{act} \cup \{t\},$$

where $\cup$ means a disjoint union, $\Sigma_{act}$ is the set of activity (or logical) events, and $t$ denotes the passage of one unit of time, or one tick of the global clock. The event set, $\Sigma_{act}$, is categorized by three subsets: the controllable events set $\Sigma_c$, the uncontrollable events set $\Sigma_{uc}$, and the forcible events set $\Sigma_{for}$. The controllable events can be enabled or disabled by a supervisor, but the uncontrollable events are always enabled. On the other hand, the forcible events can preempt the event $t$ by forcing action of a supervisor, and a forcible event may be either controllable or uncontrollable.

For two event sets $\Sigma$ and $A \subseteq \Sigma$, let $\Sigma^r$ and $A^r$ denote the sets of all finite sequences (or strings) over $\Sigma$ and $A$, respectively, including the empty string $\epsilon$. Then, any subset of $\Sigma^r$ is called the language over $\Sigma$. The transition function, $\delta$, can be extended to $\Sigma^r$ by defining $\delta(q, \epsilon) := q$ and $\delta(q, s\sigma) := \delta(\delta(q, s), \sigma)$ for all $s \in \Sigma^r$ and $\sigma \in \Sigma$. For $s \in \Sigma^r$, $pr(s)$ denotes the set of all strings that are prefixes of $s$, i.e. $pr(s) := \{t \in \Sigma^r | tu = s$ for some $u \in \Sigma^r \}$. The prefix closure $pr(L)$ of a language $L \subseteq \Sigma^r$ is the set of prefixes of all strings in $L$.

The behavior of a TDES $G$ is described by the following two languages:

$$L(G) := \{s \in \Sigma^r | \delta(q_0, s) \}$$. 

$$L_m(G) := \{s \in \Sigma^r | \delta(q_0, s) \in Q_m\}.$$
where the symbol ! means ‘is defined’. \( L(G) \) includes all possible sequences occurring in the system, and \( L_m(G) \) includes the sequences of \( L(G) \) that lead to the marked states.

### 2.2. Coaccessibility and synchronous products

A state \( q \) of \( G \) is said to be coaccessible if there is a path from the state \( q \) to a marked state in the state transition diagram of \( G \). Otherwise, the state is called non-coaccessible. \( G \) is nonblocking if \( L(G) = pr(L_m(G)) \), otherwise it is blocking. Note that if \( G \) is nonblocking, then every state of \( G \) is coaccessible.

To construct an execution model of a composite task describing the concurrent behavior of the individual tasks, we employed a synchronous product operation [15]. Given two TDESs, \( G_i = (Q_i, \Sigma_i, q_{0,i}, \delta_i, Q_{m,i}) \) for \( i = 1, 2 \), the synchronous product of \( G_1 \) and \( G_2 \) is defined as

\[
G_1 \parallel G_2 := (Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, (q_{0,1}, q_{0,2}), \delta, Q_{m,1} \times Q_{m,2}),
\]

where

\[
\delta((q_1, q_2), \sigma) := \begin{cases} 
\delta_1(q_1, \sigma), \delta_2(q_2, \sigma) & \text{if } \delta_1(q_1, \sigma)! \text{ and } \delta_2(q_2, \sigma)!, \\
\text{undefined} & \text{otherwise}.
\end{cases}
\]

In the product operation, the transition of the two automata must always be synchronized for a common event in \( \Sigma_1 \cap \Sigma_2 \). In other words, an event can occur in \( G_1 \parallel G_2 \) if and only if it occurs in both \( G_1 \) and \( G_2 \). Thus, it holds that \( L(G_1 \parallel G_2) = L(G_1) \cap L(G_2) \) and \( L_m(G_1 \parallel G_2) = L_m(G_1) \cap L_m(G_2) \) [15].

### 2.3. Supervisor and controllability

A supervisor controls a system by enabling or disabling the controllable events based on the observation of a sequence of events. Formally, a supervisor, \( S \), is defined as a map \( S : L(G) \rightarrow 2^\Sigma \) such that \( \Sigma_{uc} \subseteq S(s) \) for any \( s \in L(G) \), where \( S(s) \) represents the set of events that are enabled or forced to occur after the observation of the string \( s \). A supervised (or controlled) system is then denoted by \( S/G \), and its closed-loop behavior \( L(S/G) \) is a language defined as follows: (1) \( e \in L(S/G) \); (2) if \( s \in L(S/G) \), \( \sigma \in S(s) \), and \( s\sigma \in L(G) \), then \( s\sigma \in L(S/G) \); and (3) no other strings belong to \( L(S/G) \). Given a language specification \( K \subseteq L_m(G) \), the basic supervisory control problems are to find the existence conditions of a supervisor, \( S \), satisfying \( L(S/G) = pr(K) \) and to design such a supervisor. To solve these problems, the following language property of \( K \), called controllability, is required [4].

For \( s \in \Sigma^* \), let \( \Sigma_K(s) := \{ \sigma \in \Sigma \mid s\sigma \in pr(K) \} \). Then, \( K \) is controllable with respect to (w.r.t.) \( L(G) \) if, for any \( s \in pr(K) \)

\[
\Sigma_K(s) \supseteq \begin{cases} 
\Sigma_{L(G)}(s) \cap (\Sigma_{uc} \cup \{ t \}) & \text{if } \Sigma_K(s) \cap \Sigma_{for} = \emptyset, \\
\Sigma_{L(G)}(s) \cap \Sigma_{uc} & \text{if } \Sigma_K(s) \cap \Sigma_{for} \neq \emptyset.
\end{cases}
\]

It has been known that a supervisor \( S \) satisfying \( L(S/G) = pr(K) \) exists if and only if \( K \) is controllable w.r.t. \( L(G) \) [4]. This implies that, if a given language is controllable, then all sequences in the language are achievable.

In the next section, we present a formal method to find all of the achievable sequences that meet the deadlines of the accepted sporadic tasks. Moreover, we provide a detailed algorithm to compute a controllable language composed of such sequences. Note that a supervisor that achieves the controllable language corresponds to a scheduler that secures the deadlines of all accepted tasks.

### 3. Main results

#### 3.1. A TDES model of task execution

Let us consider a TDES model of a processor executing \( n \) sporadic tasks. For \( i \in I := \{1, 2, \ldots, n\} \), the TDES model of the processor executing sporadic task \( i \) with an execution time \( E_i \) and a deadline (relative) \( D_i \) is as follows:

\[
T_i = (Q_i, \Sigma_i, q_{0,i}, \delta_i, Q_{m,i}),
\]

where \( \Sigma = \cup_{i \in I} \Sigma_i \cup \{ t \} \) and \( \Sigma_i = \{ a_i, p_i, r_i, e_i, c_i \} \). The state transition diagram of \( T_i \) is shown in Fig. 1, and the descriptions of the events are summarized in Table 1. The events are categorized as follows:

- \( \Sigma \): All events
- \( \Sigma_i \): Events associated with task \( i \)
- \( \Sigma \setminus \Sigma_i \cup \{ t \} \): Events that are not associated with task \( i \)
- \( \Sigma \setminus (\Sigma_i \cup \{ t \}) \): Events associated with task \( i \)
- \( \Sigma \setminus (\Sigma_i \cup \{ t \}) \): Events associated with task \( i \)

\[
f(t) = \begin{cases} 
f(t) & \text{if } t < E_i, \\
0 & \text{if } t = E_i, \\
\#	ext{ of } e_i - 1 & \text{if } t = E_i + 1.
\end{cases}
\]

Fig. 1. The execution model \( T_i \) for task \( i \).
\[
\Sigma_{\text{ec}} = \bigcup_{i \in I} \{ a_i \}, \quad \Sigma_c = \Sigma \setminus (\Sigma_{\text{ec}} \cup \{ t \}), \quad \Sigma_{\text{for}} = \Sigma_c.
\]

This model is an extended version of the task model presented in [7] where periodic tasks were modeled without considering the acceptance tests.

In this paper, we assume that sporadic tasks can arrive arbitrarily. This differs from the assumption in the existing literature, including [3,5], where sporadic tasks were assumed to have minimum interarrival times. Regarding the execution times and deadlines of all sporadic tasks, we assume that they are known a priori as constant values as in the existing literature. We note, however, that in some practical situations, the execution times and deadlines of the sporadic tasks may vary widely. The results presented in this paper can be extended to more general cases, e.g. the case where a sporadic task has both minimum and maximum execution times and deadlines.

Moreover, we assume that the processing of a sporadic task requires executing \( E_i \) segments, and each segment takes one unit of time, which is represented by an event \( t \) followed by an event \( e_i \) as shown in Fig. 1. We also assume that the acceptance tests for the arrived tasks do not take any additional processing time. This assumption simplifies our developments without loss of generality, and the results presented can be easily extended to non-zero processing times.

Since \( T_i \) in Fig. 1 is a model for a sporadic task, the release time of task \( i \) can be arbitrary, e.g. \( a, ta, tta, ttaa, \ldots \in L(T_i) \). Moreover, \( T_i \) in Fig. 1 describes a preemptive scheduling scheme in the sense that a task can always preempt another task, except when the execution of a current task segment is in progress. For instance, after the event \( e_i \), other events, such as \( e_j \) and \( c_j \) \((i \neq j)\), are not defined except the event \( t \). Note that the proposed scheduler never leaves the processor idle unless no task is waiting to be executed. In other words, if a sporadic task is accepted, then the \( t \) transitions are not defined unless the task is executed in terms of \( e_i \).

Let us construct a composite model for the concurrent executions of \( n \) sporadic tasks as follows:

\[
T = \bigparallel_{i \in I} T_i.
\]

This model includes all possible execution sequences of \( n \) sporadic tasks. It includes cases where a sporadic task completes its last task segment before its deadline and also cases where the task misses its deadline due to the execution of other tasks. We note that, if a segment execution event \( e_i \) of task \( i \) occurs, the next possible event is only the unit time passage event, \( t \), which causes the task \( i \) to reach its segment completion state. In other task models, only the self-loop transitions through the event \( t \) can occur. This implies that, when a task segment is executed, the processor is occupied solely by the task during the unit time.

3.2. Problem formulation for scheduler design

To formalize the scheduler design problem addressed in this paper, we define the following notions.

**Definition 1.** For \( s_1, s_3 \in \Sigma^* \) and \( s_2 \in (\Sigma \setminus \{ c_i \})^* \), let \( s = s_1 a_i p_i s_2 c_i s_3 \in L_m(T) \). The sequence \( s \in L_m(T) \) is deadline-meeting if the number of unit time passage events in the string \( s_2 \) is less than or equal to \( D_i - 1 \) for any \( a_i, p_i, c_i \) in the string \( s \). Otherwise, the sequence \( s \) is deadline-missing.

**Definition 2.** A language \( K \subseteq L_m(T) \) is schedulable if \( K \) includes only deadline-meeting sequences and is also controllable w.r.t. \( L(T) \).

We consider the following problem in this paper:

Find the largest schedulable language \( K \) in the sense that it contains all achievable deadline-meeting sequences in \( T \). If we find such a schedulable language \( K \), then we can design a scheduler (or supervisor) \( S \) such that \( L(S/T) = pr(K) \).

Let us find all deadline-meeting sequences of \( L_m(T) \). To this end, we first construct a deadline specification model for each task as illustrated in Fig. 2. The model \( H_i \) for task \( i \) includes all possible sequences that meet the deadline \( D_i \) of the task when it is accepted by the scheduler. Based on these individual models, we construct the composite deadline specification model as follows:

\[
H = \bigparallel_{i \in I} H_i.
\]

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_i )</td>
<td>Arrival (or release) of task ( i )</td>
</tr>
<tr>
<td>( p_i )</td>
<td>Acceptance of task ( i )</td>
</tr>
<tr>
<td>( r_i )</td>
<td>Rejection of task ( i )</td>
</tr>
<tr>
<td>( e_i )</td>
<td>Execution of a segment of task ( i )</td>
</tr>
<tr>
<td>( c_i )</td>
<td>Execution of the last segment of task ( i )</td>
</tr>
</tbody>
</table>
This overall specification model includes all sequences that concurrently meet the deadlines of the accepted sporadic tasks. However, we note that this composite model may be blocked, i.e. $L(H) \neq pr(L_m(H))$; thus, there may be some deadlock states from which there is no outgoing path leading to marked states. So, some accepted tasks may be led to a deadlock state within their deadlines, but there are no further sequences from those states that meet all deadlines of the tasks.

In general, the behavior of the composite deadline specification model $H$ does not belong to that of the composite task model $T$, i.e. $L(H) \not\subseteq L(T)$. Hence, it is necessary to find all sequences meeting the deadlines of the accepted tasks among the sequences of $L(T)$. For this purpose, we construct the following finite state automaton $M$:

$$M = T||H.$$ 

We note that since $H$ may be blocked, $M$ may also be blocked. In addition, since $L(M) \neq pr(L_m(M))$, some sequences in $L(M)$ may exist that lead to the deadlock states of $M$. In other words, some sequences may exist in $L(M)$ after which there is no more continuation of events leading to the marked states of $M$.

Based on the construction of $M$, we obtained the following result.

**Proposition 1.** $L_m(M)$ contains all deadline-meeting sequences of $L_m(T)$.

**Proof.** We prove this by contradiction. Suppose that a deadline-meeting sequence $s \in L_m(T)$ that satisfies $s \notin L_m(M)$ exists. According to the definition of deadline-meeting sequences, for any task $i$ with $s = s_1 a_1 p_1 s_2 c_1 s_3$ where $s_1, s_3 \in \Sigma$ and $s_2 \in (\Sigma \setminus \{c_i\})^*$, the number of unit time passage events in the string $s_2$ should be less than or equal to $D_i - 1$, which implies that $s \in L(H_i)$. Since this should be satisfied for any task $i$, it holds that $s \in \cap_{i \in \mathbb{Z}} L(H_i)$ implying that $s \in L(H)$. From $s \in L_m(T)$ and $s \in L(H)$, $s \in L_m(M)$ follows the definition of $M = T||H$. This contradicts the assumption. Therefore, we can conclude that $L_m(M)$ contains all deadline-meeting sequences of $L_m(T)$. □

### 3.3. Schedulable language and scheduler

To secure all the deadlines of the accepted tasks, a scheduler should be designed to achieve $L_m(M)$. For this, it is necessary for $L_m(M)$ to be controllable w.r.t. $L(T)$. In general, $L_m(M)$ is, however, uncontrollable w.r.t. $L(T)$. In other words, $L_m(M)$ is not an achievable language. Thus, we need to compute a controllable language encompassing $L_m(M)$. To this end, we construct a finite state automaton $M'$ from $M$ according to the following procedure. Let $\delta_M$ and $\delta_{M'}$ be the state transition functions of $M$ and $M'$, respectively. Let the other elements of $M$ and $M'$ be identical except the transition functions. For $s \in \Sigma^*$, let $le(s)$ denote the last event of the string $s$, e.g. $le(\alpha \beta \gamma) = \gamma$ for $\alpha, \beta, \gamma \in \Sigma$. Then, we construct $\delta_{M'}$ based on $\delta_M$ as follows. For every coaccessible state $q$ of $M$, perform the following:

1. Let $s \in \{a_1, \ldots, a_n\}^\ast \setminus \{\epsilon\}$.
2. For any $s' \in pr(s) \setminus \{\epsilon\}$, let $\delta_M(q, s') = q'$ be a non-coaccessible state of $M$.
3. For every non-coaccessible $q'$, let $\delta_{M'}(q', (r_i)) := q$ and $le(s') = a_i$, and eliminate the $p_i$ transition (and its subsequent transitions) from $M$.

Note that $M'$ is constructed from $M$ by adding the rejection transition $(r_i)$ from a non-coaccessible state $q'$ to a coaccessible state $q$.

**Remark 1.** Since $q$ is a coaccessible state of $M$, there exist some $u \in pr(L_m(M))$ and $v \in \Sigma^*$ such that $\delta_M(q^M_0, u) = q$ and $uv \in L_m(M)$. Then, $us' \in pr(L_m(M))$ while $us' r_i \in L_m(M')$, which leads to $us' r_i v \in L_m(M') \setminus L_m(M)$.

Let $K := L_m(M')$. Then, the main results of this paper are as follows.

**Proposition 2.** The language $K$ is schedulable w.r.t. $L(T)$. 
Proof. First, let us show that $K$ contains only deadline-meeting sequences. For a sequence $s \in K$, from $K = L_m(M') \supseteq L_m(M)$, we consider the following two cases: $s \in L_m(M)$ (Case 1) and $s \in L_m(M') \setminus L_m(M)$ (Case 2). For Case 1, according to Proposition 1, the sequence $s$ is deadline-meeting. For Case 2, according to Remark 1, it follows that $s = s_1 s_2 r_1 s_3$ for some $s_1, s_2, r_1, s_3 \in \Sigma^*$ and $s_3 \in \{a_1, \ldots, a_n\}$ satisfying $le(s_3) = a_i$ and $s_1 s_2 \in L_m(M)$. Then, since the sequence $s r_1$ does not include the unit time passage event $t$ and the sequence $s_1 s_2$ is deadline-meeting, $s$ is also deadline-meeting.

Next, let us show that $K$ is controllable w.r.t. $L(T)$. For this, we show that for any $s \in pr(K)$, $st \in L(T)$ implies $st \in pr(K)$. This is clear from the fact that the event $t$ is only defined at the initial state of $T$ or the execution of task segment $i$, i.e., $e_i$ or $c_i$. In other words, if $s \in pr(K)$ and $st \in L(T)$, then $le(s) = e_i$ or $le(s) = c_i$, or $le(s) = c_i$. Hence, $st \in pr(K)$ holds since $K$ includes only deadline-meeting sequences.

From the above, it is apparent that if $s \in pr(K)$ and $sz \notin pr(K)$, then $z \in \Sigma^*$ or $z = a_i$ (recall that $\Sigma^c = \bigcup_i \{a_i\}$). Thus, in order to show that $K$ is controllable w.r.t. $L(T)$, it is necessary to show that for any $s \in pr(K)$ and $u \in \{a_1, \ldots, a_n\}^*$, $su \in L(T)$ implies $su \in pr(K)$. It follows from Remark 1 that, if $s \in pr(L_m(M))$ and $su \in L(M) \setminus pr(L_m(M)) \subseteq L(T)$, then $sur_{1} \in pr(L_m(M')) = pr(K)$ where $le(u) = a_i$, which implies $su \in pr(K)$. Thus, it can be seen that $K$ is controllable w.r.t. $L(T)$. □

Theorem 1. The language $K$ is the largest schedulable language.

Proof. Since $K = L_m(M') \supseteq L_m(M)$ and $L_m(M)$ contains all deadline-meeting sequences of $T$, $K$ includes all deadline-meeting sequences of $T$. Moreover, according to Proposition 2, $K$ is controllable w.r.t. $L(T)$. Therefore, we conclude that $K$ is the largest schedulable language. □

A supervisor, $S$, achieving the language $K$ corresponds to a complete scheduler because it guarantees all achievable deadline-meeting sequences of the system. The scheduler can be designed as follows: for any $s \in pr(K)$

$$S(s) := \{z \in \Sigma^c | \Sigma^c \subseteq \{a_i\} \} \cup \Sigma^c,$$

which represents the set of events to be enabled when the sequence $s$ is observed. As a result of the supervision, we obtain $L(S/T) = pr(K)$. This implies that the task executions controlled by the scheduler remain within the schedulable language $K$. Note that the scheduler can be designed with the language $K$ in an offline manner, while the designed scheduler can control the task executions in an online manner. In particular, when a sporadic task arrives after the execution of a certain sequence $s$, the scheduler only checks whether the acceptance event of the task belongs to $S(s)$ or not. Thus, it provides a unified solution to the following two scheduling problems: how to test the schedulability of the newly arrived task and how to schedule it. Most conventional online schedulability test algorithms incur a significant computational burden when they make a decision of acceptance or rejection. For instance, in [5], when a sporadic task arrives, it is inserted into an ordered list of tasks based on an EDF policy. Then, the computation and analysis of the residual times of all tasks in the currently active task set are performed to determine the schedulability of the newly arrived task. The residual time of a task at a specific time instant is the remaining time interval until the task reaches its deadline. In this respect, we argue that, compared with conventional online schemes, the proposed scheme can reduce the computational burden of testing the schedulability of a newly arrived task.

However, one important drawback of the proposed scheme is the computational complexity involved when computing a schedulable language. Here, the computation is required primarily for the component TDES models. The state space size of a composite model exponentially increases with the number of tasks. For instance, let us consider a real-time system that processes 10 sporadic tasks, each modeled by a 5-state automaton. In this case, the composite task execution model $T$ has approximately $5^{10}$ states. This exponential growth of state complexity also occurs for the deadline specification models. We note, however, that the automaton $M := T \parallel H$ has considerably less complexity than the worst case of $5^{10}$ states, since $M$ includes only the sequences that meet the deadlines of the accepted tasks among the sequences of $L(T)$. Nevertheless, this problem of exponential growth in the state complexity is a challenging issue that needs further consideration to realize the proposed scheme. However, various approaches developed in the supervisory control area, such as modular synthesis [16], can be incorporated to deal with this problem. Moreover, some brute-force methods based on database technologies and computing systems can also be used to address the issue of a large number of states.

4. Examples

4.1. A TDES model of two sporadic tasks

Let us consider a real-time system processing two sporadic tasks: Task 1 and Task 2. The TDES models for executing these tasks are $T_1$ and $T_2$, respectively. The event sets are defined as follows: $\Sigma_i = \{a_i, p_i, r_i, e_i, c_i\}$ for $i = 1, 2$. Suppose that the execution times and deadlines of these two tasks are given as follows: $E_i = 2$ and $D_i = 3$ for Task 1; $E_2 = 1$ and $D_2 = 1$ for Task 2. The state transition diagrams for $T_1$ and $T_2$ are shown in Fig. 3 where the initial state is indicated by an entering arrow and the marked state is denoted by double circles. In this case, both the initial state and the marked state are identical.

Based on these models, we construct a composite task execution model $T = T_1 \parallel T_2$ as shown in Fig. 4. This composite model includes all possible sequences occurring during the concurrent execution of the two tasks that contain both deadline-meeting and deadline-missing sequences. For instance, let us consider two sequences $s_1, s_2 \in L_m(T)$ as follows:
Fig. 3. Task execution models $T_1$ and $T_2$.

$[T_1 \text{ with } E_1=2, D_1=3 ]$

$[T_2 \text{ with } E_2=1, D_2=1 ]$

Fig. 4. A composite model $T = T_1 || T_2$.

$[H_1 ]$

$[H_2 ]$

$[H = H_1 || H_2 ]$

Fig. 5. Deadline specification models.
Note that $s_1$ is deadline-meeting, but $s_2$ is deadline-missing, since the number of $t$ of the substring $a_2p_2c_1tc_2$ in the string $s_2$ is not $D_2 - 1 = 1 - 1 = 0$ but 1.

4.2. Scheduler design

In order to find all deadline-meeting sequences in $L_m(T)$, we construct deadline specification models as shown in Fig. 5. Based on these models, we construct a finite state automaton $M = T|H$ as shown in Fig. 6 where the deadline-meeting sequences are denoted by thick lines. The sequences denoted by thin lines in Fig. 6 imply that at least one of the accepted tasks in the sequences cannot complete its execution within the deadline. In Fig. 6, we also observe that all events that deviate from the deadline-meeting sequences are the arrival events, i.e. $a_i$. This means that, if a task with such a deviated arrival is accepted by a scheduler, then there is no method to schedule the tasks to meet all deadlines of both the previously accepted tasks and the newly accepted task. Hence, a rejection decision should be made for all tasks with arrival events that
deviate from the deadline-meeting sequences of \( L_m(M) \). For this purpose, from \( M \) in Fig. 6, we construct a finite state automaton \( M' \) according to the procedure described in the previous section, as shown in Fig. 7. Then, the marked language \( K = L_m(M') \) becomes a schedulable language w.r.t. \( L(T) \), and according to Theorem 1, it is the largest one.

Consequently, a scheduler \( S \) exists that satisfies \( L(S/T) = pr(K) \), and thereby the scheduler can be designed as follows: for any \( s \in pr(K) \),

\[
S(s) = \{ x \in \Sigma \setminus \Sigma_{uc} | sx \in pr(K) \} \cup \Sigma_{uc}.
\]

According to this design, when the sequence \( s = a_1p_1e_1a_2p_2 \) is observed, the decision of the scheduler is \( S(s) = \{ c_2, t \} \cup \Sigma_{uc} \). This implies that if Task 2 arrives in the middle of executing Task 1, the scheduler decides a context switching to Task 2. Since \( \Sigma_{uc} = \Sigma_{for} \) and the events deviated from the deadline-meeting sequences are all arrival events, there is no need to consider forcing actions of the scheduler in this case.

5. Conclusions

In this paper, we have presented a formal constructive method for the preemptive scheduling of sporadic tasks based on the supervisory control theory of DESs. The resulting scheduler is optimal for sporadic tasks with arbitrary arrival times. This implies that if the proposed scheduler rejects a newly arrived task, then there is no method of designing other schedulers that can meet the deadline of the rejected task.

The presented method is based on the offline analysis for computation of the largest schedulable language including all achievable sequences that meet the deadlines of the accepted tasks. Here, the largest schedulable language means a complete solution in that there is no other scheduler that can guarantee the achievement of any deadline-meeting sequence not included in the language.

Real-time systems often process both periodic and sporadic tasks that share resources. However, the periodic tasks and resource requirements are not addressed in this paper. So, a future research topic is to build an optimal and complete scheduler for such real-time systems processing of both periodic and sporadic tasks with resource requirements based on the supervisory control theory.

Acknowledgments

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2005-041-D00492), by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (M10503010001-07N030100112), and through the Nuclear Research Grant (M20708000001-07B0800-00110) and the 21C Frontier Microbial Genomics and Application Center Program.

References