Dynamic Multiple-Period Reconfiguration of Real-Time Scheduling Based on Timed DES Supervisory Control

Xi Wang, ZhiWu Li, W. M. Wonham

Version Post-print/accepted manuscript

Citation (published version) Xi Wang, Zhiwu Li, W. M. Wonham. Dynamic multiple-period reconfiguration of real-time scheduling based on timed DES supervisory control. IEEE Trans. on Industrial Informatics 12(1) February 2016, pp.101-111.

Publisher's Statement © 2016 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

How to cite TSpace items

Always cite the published version, so the author(s) will receive recognition through services that track citation counts, e.g. Scopus. If you need to cite the page number of the author manuscript from TSpace because you cannot access the published version, then cite the TSpace version in addition to the published version using the permanent URI (handle) found on the record page.

This article was made openly accessible by U of T Faculty. Please tell us how this access benefits you. Your story matters.
Abstract—Based on the supervisory control theory (SCT) of timed discrete-event systems (TDES), this study presents a dynamic reconfiguration technique for real-time scheduling of real-time systems running on uni-processors. A new formalism is developed to assign periodic tasks with multiple-periods. By implementing SCT, a real-time system (RTS) is dynamically reconfigured when its initial safe execution sequence set is empty. During the reconfiguration process, based on the multiple-periods, the supervisor proposes different safe execution sequences. Two real-world examples illustrate that the presented approach provides an increased number of safe execution sequences as compared to the earliest-deadline-first (EDF) scheduling algorithm.

Index Terms—Real-time system, timed discrete-event system, supervisory control, dynamic reconfiguration, non-preemptive scheduling.

I. INTRODUCTION

In [1], Liu and Layand define a periodic task model with a deadline equal to its period, which we refer to as the Liu-Layand (LL) model. Thereafter, Nassor and Bres propose a new task model in [2], with a deadline less than or equal to its period, which we refer to as the Nassor-Bres (NB) model. Currently, the most widely-used scheduling algorithms for hard periodic real-time systems (RTS) running on a uni-processor are fixed priority (FP) scheduling and earliest-deadline-first (EDF) scheduling algorithms [1]. Moreover, an RTS can be scheduled in a preemptive or non-preemptive mode [3], [4]. In real-time scheduling theory, these widely applied algorithms provide at most one schedulable sequence for an RTS to meet the hard deadlines. For non-preemptive scheduling of an RTS that executes the NB model tasks, Chen and Wonham [5] propose a timed discrete-event system (TDES)-based task model, which we refer to as the Chen-Wonham (CW) model, and a real-time scheduling technique. Based on supervisory control theory (SCT), all safe execution sequences are generated by the TDES supervisor, from which the user chooses a preferred sequence to schedule the RTS. The RTS is claimed to be non-schedulable if the supervisor is empty. Based on the LL model and SCT, a priority-based and preemptive real-time scheduling policy and a task model, which we refer to as the Janarthanan-Gohari-Saffar (JGS) model, are proposed by Janarthanan et al. in [6]. The work in [5] and [6] is a significant improvement over real-time scheduling. However, the authors did not reconfigure the system in case of non-schedulability.

In [7]–[10], an elastic period task model is proposed to handle the overload of an RTS by decreasing the task processor utilization. Moreover, the supremal controller found by SCT provides the RTS with all the safe execution sequences [5]. Building on the two latter studies, we present a new modeling technique to endow the real-time tasks represented by the CW and JGS models with multiple-periods. To handle the overload of an RTS, SCT is utilized to find all the possible solutions based on different periods of each task. For each solution, all the safe execution sequences are provided.

Dynamic reconfiguration in the present study consists of two steps: 1) the initial model of each task is assigned with the shortest period (the highest processor utilization), and by utilizing SCT, all the RTS’ safe execution sequences (if any) are found; 2) for the purpose of reconfiguring the RTS in case of non-schedulability, this study reconfigures the RTS’ composite task model by assigning to the tasks multiple-periods. The multiple-period provides multiple processor utilization for each task. Thereafter, a processor utilization interval for the RTS is obtained. SCT is utilized again to find all the safe execution sequences (possible reconfiguration scenarios) in the predefined processor utilization interval. If the supervisor is still empty, we claim that the RTS is non-schedulable. Two real-world examples are implemented in this study. The results illustrate that, in the dynamic reconfiguration approach, the presented method finds a set of safe execution sequences.

The rest of this paper is structured as follows. The state of the art is reviewed in Section II. Section III presents the terminology used throughout the paper. The multiple-period TDES model for RTS is defined in Section IV. Section V reports methodologies of supervisory control and reconfiguration of RTS. A real-world example is implemented in Section VI to verify the supervisory control and reconfiguration. Further relevant issues are discussed in Section VII. Conclusions are provided in Section VIII.

II. STATE OF THE ART

In a periodic RTS, a permanent overload condition occurs if the processor utilization is greater than one [11]. In this case, the RTS needs to be reconfigured. In recent years several academic and industrial studies [12]–[14] have addressed the dynamic reconfiguration of RTS. These approaches can be divided into two categories: manual, applied by users [15], and automatic, applied by intelligent control agents [16]. The most widely used overload management approaches are: elastic scheduling [7]–[10] and job skipping [17]. In real-time scheduling, effective solutions for reconfiguration based on sensitivity approach of worst-case execution times (WCET), deadlines, and periods of tasks are reviewed in [18]. These solutions are utilized to reconfigure the RTS scheduled by FP real-time scheduling. There is no reconfiguration result...
based on the sensitivity approach to reconfigure the tasks’ periods for dynamic-priority real-time scheduling [18]. Based on SCT, this study presents a new dynamic reconfiguration technique to reconfigure the RTS when they are claimed to be non-schedulable under the approaches in [5] or [6]. Unlike traditional real-time scheduling and reconfiguration via the calculation of processor utilization, processor demand [19], and on-line monitoring to provide one safe execution sequence, an off-line technique is presented: in a predefined processor utilization interval, based on SCT, all the safe execution sequences (possible reconfiguration scenarios) are found.

III. CONCEPTS AND TERMINOLOGY

A. Preliminaries on TDES

In the language-based Ramadge-Wonham (RW) framework [20], [21], a finite discrete-event system (DES) is represented by a state machine G = (Q, δ, Q0, Qm), where Q is the state set, Σ is the event set, δ: Q × Σ → Q is the (partial) state transition function, q0 is the initial state, and Qm is the marker state set satisfying Qm ⊆ Q. Let Σ+: (resp., ε) denote the set of all finite sequences over Σ (resp., empty string). We have Σ− ≡ = Σ+ ∪ {ε}. A plant and a specification are represented by G and S, respectively. In [22], by adjoining to the RW framework time bounds on the transitions, G starts from an (uninit) activity transition graph (ATG) Gact = (A, Σact, δact, a0, A0) with Σ := Σact ∪ {tick}. The elements of the activity set A are “activities”, denoted by a. Σact is partitioned into two subsets, Σact = Σspec ∪ Σrem, where Σspec (resp., Σrem) is the prospective (resp., remote) event set with finite (resp. infinite) upper time bounds [21].

By defining the timer interval for σ, represented by Tσ, to be [0, uσ] or [0, lσ], where σ ∈ Σspec and σ ∈ Σrem, respectively, the initial state is q0 := (q0, {τ ∈ σ|σ ∈ Σact}), where lσ = uσ or lσ for a prospective or remote state, respectively.

The marker state set is Qm ⊆ A0 × Π(Tσ|σ ∈ Σact). Thus a TDES is represented by G = (Q, Σ, δ, Q0, Qm).

An event σ ∈ Σact is enabled at q if δact(a, σ) is defined, written δact(a, σ)!, it is eligible if its transition δ(q, σ) is also defined, i.e., δ(q, σ)!.

The closed behavior language L(G) := {s ∈ Σ+|δ(q0, s)!}. In addition, the marked behavior of G is Lm(G) := {s ∈ L(G)|δ(q0, s) ∈ Qm}. G is non-blocking if Lm(G) satisfies Lm(G) = L(G), where Lm(G) is the (prefix) closure of L(G). In a TDES plant G, the eligible event set Elig(k) ⊆ Σ at a state k corresponding to a string s ∈ L(G) is defined by Elig(k)(s) := σ ∈ Σ|σσ ∈ L(G).

For an arbitrary language K ⊆ L(G), let s ∈ L(G), Elig(k)(s) := {σ ∈ Σ|σσ ∈ K}. The set of all controllable sublanguages of K is denoted by C(K); this family is nonempty (the empty set belongs) and is closed under arbitrary set unions. Hence, a unique supremal (i.e., largest) element exists, and is denoted by supC(K). Considering a specification language E ⊆ Σ+, there exists an optimal monolithic supervisor S. Its closed behavior is L(S) = Lm(S), where Lm(S) is the marked behavior represented by Lm(S) = supC(E ∩ Lm(G)) ⊆ Lm(G).

B. Synchronous product

Suppose that we have a set of generators Gi with i ∈ n = {1, 2, . . . , n}. In accordance with [21], the behavior of Gi is represented by language Li. Synchronous product [21] is a standard way to combine several DES into a single and more complex one. Suppose that we have two languages L1 ⊆ Σ1 and L2 ⊆ Σ2 with Σ = Σ1 ∪ Σ2. The natural projection P1: Σ → Σ1 is defined by

- P1(ε) = ε,
- P1(σ) = {σ, if σ ∈ Σ1,
- P1(σ) = P2(σ), if σ ∈ Σ2.

The inverse image function of P1 is

P1−1 : Pwr(Σ1) → Pwr(Σ).

For H ⊆ Σ1, P1−1(H) := {s ∈ Σ1|P1(s) ∈ H}.

The synchronous product of L1 and L2, denoted by L1||L2, is defined as

L1||L2 := P1−1L1 ∩ P2−1L2.

C. System Model

Suppose that a periodic RTS S processes n tasks, i.e., S = (τ1, τ2, . . . , τn), i ∈ n. Assume also that this set contains at least one task with a multiplet-period, namely one having a lower and upper (non-negative integral time) bound. The execution model of such a system is a set of tasks processed in a uni-processor, in which a task τi is described by (Ri, Ci, Di, Ti) with

- release time Ri,
- WCET Ci,
- hard deadline Di, and
- multiplet-period Ti.

An RTS is a synchronous system [19] in case all the processed tasks are released at the same time, namely Ri = 0. In this research the RTS is synchronous. A deadline is hard if its violation is unacceptable. A multiplet-period is a period set containing several possible periods; the lower bound (i.e., shortest one) is represented by Tmin, and the upper bound (i.e., longest one) is represented by Tmax. Thus, we have

Ti = [Tmin, Tmax].

During the real-time scheduling process, for task τi, only one period T satisfying Tmin ≤ T ≤ Tmax is selected in each scheduling period. The processor utilization Ui of task τi is calculated by

Ui = Ci/Ti.

The total processor utilization of S is Us = n Σ Ui. An RTS S is not schedulable in case Us > 1 [11].

Task τi consists of an infinite sequence of jobs Jij = (r1i, Ci, di, τi) repeated periodically. The absolute deadline di denotes the global clock time at which the execution of Jij must be completed. Similarly, we define the absolute
**IV. TDES MODEL FOR REAL-TIME SYSTEMS**

**A. CW Model**

The CW model [5] represents a real-time periodic task $\tau_i = (R_i, C_i, D_i, T_i)$, $i \in n$, with $D_i \leq T_i$, by a TDES $G_i = (Q_i, \Sigma_i, \delta_i, q_{0i}, Q_{mi})$. As depicted in Fig. 1, the corresponding ATG part is $G_{act} = (A_i, \Sigma_{acti}, \delta_{acti}, q_{0i}, A_{mi})$ with

- $A_i = \{Y_i, I_i, W_i\}$,
- $\Sigma_{acti} = \{\gamma_i, \alpha_i, \beta_i\}$,
- $\delta_{acti} : A_{acti} \times \Sigma_{acti} \rightarrow A_{acti}$ with
  - $\delta_{acti}(Y_i, \gamma_i) = I_i$,
  - $\delta_{acti}(I_i, \alpha_i) = W_i$, and
  - $\delta_{acti}(W_i, \beta_i) = Y_i$,
- $q_{0i} = Y_i$, and
- $A_{mi} = \{Y_i\}$.

![Fig. 1: ATG of a real-time task.](image)

States $Y_i$, $I_i$, and $W_i$ represent that task $\tau_i$ is at states delay, idle, and work, respectively. The events in the alphabet $\Sigma_i$ are

- $\gamma_i$: the event that $\tau_i$ is released,
- $\alpha_i$: the execution of $\tau_i$ is started, and
- $\beta_i$: the execution of $\tau_i$ is finished.

Event $\alpha_i$ is controllable and events $\gamma_i$ and $\beta_i$ are uncontrollable. Moreover, all the events in $\Sigma_{acti}$ are forcible. Suppose that, after enabling, events $\gamma_i$, $\alpha_i$, and $\beta_i$ should wait for $t_{\gamma_i}$, $t_{\alpha_i}$, and $t_{\beta_i}$ ticks, respectively, until they are eligible to occur. Thus, $t_{\alpha_i}$ is the time at which $\tau_i$ starts its execution. Furthermore, in the CW model, $t_{\beta_i} = C_i$. A CW model has the following two features: 1) $\gamma_i$ signals that after $r_{i,1}$, $\tau_i$ will release at every $T_i$ ticks periodically; and 2) $\beta_i$ must occur before $\tau_i$ is released again. The time interval between the occurrences of events $\beta_i$ and $\gamma_i$ is the remaining time of the current period, which decreases along with the increase of $t_{\alpha_i}$. Hence, in two adjacent periods, the values of $t_{\gamma_i}$ could be different. Formally,

- $\gamma_i$ has time bounds $[0, 0]$, if $\tau_i$ releases at $r_{i,1}$,
- $\gamma_i$ has time bounds $[T_i - t_{\alpha_i} - t_{\beta_i}, T_i - t_{\alpha_i}]$, if $r_{i,j} = T_i$ releases at $r_{i,j}$,
- $\alpha_i$ has time bounds $[0, D_i - t_{\beta_i}]$, and
- $\beta_i$ has time bounds $[t_{\beta_i}, t_{\beta_i}]$.

**B. JGS Model**

Another TDES real-time task model, the JGS model proposed in [6], can be utilized to preemptively schedule periodic tasks $\tau_i$ satisfying $D_i = T_i$. The scheduling is priority-based. The general TDES models for the WCET and the period of each task are represented by the two TDES generators shown in Figs. 2 and 3, respectively, in which $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \cdots \cup \Sigma_n$, and $\Sigma' = \Sigma \cup \{l\}$. The event set $\Sigma_i$ for $\tau_i$ is composed of

- $a_i$: the arrival of task $\tau_i$,
- $c_i$: the execution of task $\tau_i$, and
- $e_i$: the execution of the last time unit of task $\tau_i$.

Event $a_i$ is uncontrollable while events $c_i$ and $e_i$ are controllable. Moreover, all the events in the alphabet $\Sigma_i$ are forcible.

![Fig. 2: JGS WCET model.](image)

![Fig. 3: JGS period model.](image)

**C. Comparison between CW and JGS Models**

Several differences between CW and JGS models are shown in Table I, in which Y and N represent “yes” and “no”, respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>$D \leq T$</th>
<th>priority</th>
<th>preemption</th>
</tr>
</thead>
<tbody>
<tr>
<td>CW</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>JGS</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

Both CW and JGS models have their advantages and disadvantages. Thus they can be utilized to model different RTS. The CW model can be utilized to model an RTS executing
a set of periodic tasks with deadlines less than or equal to their corresponding periods. However, priority-based scheduling and preemptive scheduling cannot be accommodated by the CW model. On the contrary, the JGS model can only be utilized to model an RTS executing a set of tasks with deadlines equal to their periods. Moreover, in the JGS model, priority-based scheduling and preemptive scheduling of real-time tasks are addressed. Users can choose different models to solve different real-time scheduling problems.

D. TDES Model for Multiple-Period Tasks

The elastic task model in [7]–[10] assigns a lower and an upper period bound for each task to dynamically reconfigure an RTS. At each time, the reconfiguration of each task’s period is assigned a value between the two bounds $T_{min}$ and $T_{max}$. Consequently, the processor utilization $U_t$ of an elastic periodic task has a lower bound $U_{min}$ and an upper bound $U_{max}$. Formally, we have

$$U_t = [U_{min}, U_{max}]$$

with $U_{min} = C_i/T_{max}$ and $U_{max} = C_i/T_{min}$. The system processor utilization is

$$U_S = [U_{min}, U_{max}]$$

with $U_{min} = \sum_{i=1}^{n} U_{i_{min}}$ and $U_{max} = \sum_{i=1}^{n} U_{i_{max}}$. In the interval $[U_{min}, U_{max}]$, there may exist multiple safe execution sequences (reconfiguration scenarios) that correspond to different processor utilizations. Moreover, SCT [21] is utilized to find the supremal controllable sublanguages, i.e., it is possible to provide multiple reconfiguration scenarios for each task.

Building on the elastic task model and SCT, we present a new model that provides all the possible periods for each task; the supervisor provides all the safe execution sequences (possible reconfiguration scenarios) simultaneously. Users choose any scenario to reconfigure the RTS dynamically.

A regular periodic task with a fixed period is considered as a multiple-period task $\tau_i$ with $T_{i_{min}} = T_{i_{max}}$. With a regular task, the reconfiguration of its period would affect its utilization, which is not allowed. On the other hand, SCT is utilized to provide all the possible scheduling paths based on different periods (utilizations).

1) Multiple-period CW (MCW) model:

In this study, the MCW model is depicted in Fig. 4, in which $y_0$ is the initial state, and $\{y_{min}, y_{min+1}, \ldots, y_{max-1}, y_0\}$ is the marker state set. Each marker state represents that $\tau_i$ has finished the current execution of $J_{i,j}$ and is ready for the release of $J_{i,j+1}$. State $y_0$ represents that job $J_{i,j}$ finishes its operation at $T_{i_{max}}$ or has never been invoked. States $y_{min}, y_{min+1}, y_{max-1}$, and $y_{max-1}$ represent that job $J_{i,j}$ finishes its operation at times $T_{i_{min}}, T_{i_{min+1}},$ and $T_{i_{max-1}}$, respectively.

On the occurrence of $\alpha_1$, $\tau_i$ starts the processing of the current job. After event tick occurs $C_i$ times, the execution of $\tau_i$ is completed. The next occurrence of event $\gamma_i$ drives $\tau_i$ into the next execution period. Formally,

$$\begin{align*}
\tau_i &\text{ has time bounds } [0, 0], \\
\gamma_i &\text{ has time bounds } [T_{i_{min}} - t_{\alpha_1} - t_{\beta_1} - t_{\alpha_1} - t_{\beta_1}], \\
\text{if } (\forall j > 1) &\text{, } \tau_i \text{ releases at } r_{i,j}
\end{align*}$$

Remarks:

1. Initially, a task with $T_i = T_{i_{min}}$ plays the role of the task proposed in [5]. In this case, task $\tau_i$ always stays at the highest processor utilization. If the RTS is non-schedulable, the multiple-period model with $T_i = [T_{i_{min}}, T_{i_{max}}]$ is utilized to provide all the possibilities to compress the processor utilization.

2. For example, a non-reconfigurable periodic task $\tau_1$ is defined as $\tau_1 = (0, 1, 4, [5, 5])$. The processor utilization of task $\tau_1$ is fixed to be $U_1 = 1/5$. The TTG model $G_1$ for task $\tau_1$ is depicted in Fig. 5. For the events in $\Sigma_{act1}$,

$$\begin{align*}
\gamma_1 &\text{ has time bounds } [0, 0], \\
\text{if } \tau_1 \text{ releases at } r_{1,1} &\text{, } [A - t_{\alpha_1}, 4 - t_{\alpha_1}], \\
\text{if } (\forall j > 1) &\text{, } \tau_1 \text{ releases at } r_{1,j}
\end{align*}$$

$$\begin{align*}
\alpha_1 &\text{ has time bounds } [0, 4 - t_{\beta_1}], \\
\beta_1 &\text{ has time bounds } [1, 1].
\end{align*}$$
3. The remaining time between $\beta_i$ and $\gamma_i$ equals 0 if 1) $D_i = T_i$ and 2) $\alpha_i$ occurs at time $D_i - t_{\beta_i}$. As a result, the occurrence of $\beta_i$ may lead the TDES model to state $y_0$ directly. For example, suppose that we have two other tasks $\tau_2 = (0, 2, 6, [4, 6])$ and $\tau_3 = (0, 2, 5, [3, 5])$. The corresponding TTG models $G_2$ and $G_3$ are illustrated in Figs. 6 and 7, respectively, in which events $\beta_2$ and $\beta_3$ lead the TDES model to the initial states directly. All the possible processor utilizations of $\tau_2$ are $2/4$, $2/5$, and $2/6$; and the possible processor utilizations of $\tau_3$ are $2/3$, $2/4$, and $2/5$.

![Fig. 6: Multiple-period TDES $G_2$.](image1)

![Fig. 7: Multiple-period TDES $G_3$.](image2)

4. The time bounds $[T_{\min} - t_{\alpha_i} - t_{\beta_i}, T_{\max} - t_{\alpha_i} - t_{\beta_i}]$ for event $\gamma_i$ for $r_{i,j}$ with $j \geq 1$ are dynamic, which decreases along with the increase of $t_{\alpha_i}$.

2) Multiple-period JGS (MJGS) model:

In order to assign a multiple-period to a JGS model, we need to define marker states. Consequently, the initial states are revised, i.e., they are also assigned to be marker states. The new models for WCET and multiple-period are depicted in Figs. 8 and 9, respectively. In Fig. 9, we have that $y_0$ is the initial state; and $\{T_{\min}, y_{\min} + 1, \cdots, y_{\min} + 1, y_0\}$ is the marker state set. Each marker state represents that $\tau_i$ has finished the current execution of $J_{i,j}$ and is ready for the release of $J_{i,j+1}$. State $y_0$ represents that job $J_{i,1}$ finishes its operation at $T_{\max}$ or has never been invoked. States $y_{\min}, y_{\min} + 1,$ and $y_{\max} - 1$ represent that job $J_{i,j}$ finishes its operation at times $T_{\min}, T_{\min} + 1$ and $T_{\max} - 1$, respectively. The transition rule

$$(q \in \{T_{\min}, y_{\min} + 1, \cdots, y_{\min} + 1, y_0\}) \Rightarrow \delta(q, \alpha_i) = 0$$

represents the new arrival of task $\tau_i$.

![Fig. 8: Revised WCET of JGS model.](image3)

![Fig. 9: Multiple-period of a JGS model.](image4)

Example.

Suppose that we have two tasks $\tau_a = (0, 1, [2, 3], [2, 3])$ and $\tau_b = (0, 2, [3, 3], [3, 3])$. The WCET of $\tau_a$ and $\tau_b$ are illustrated in Figs. 10 and 11, respectively. The multiple-periods of $\tau_a$ and $\tau_b$ are shown in Figs. 12 and 13, respectively. The processor utilization of $\tau_a$ could be $1/2$ or $1/3$. The processor utilization of $\tau_b$ is $2/3$.

![Fig. 10: WCET of $\tau_a$.](image5)

![Fig. 11: WCET of $\tau_b$.](image6)

E. Task Creation and Editing in TTCT

The TDES synthesis procedure TTCT is a software package to create an RTS as the composite model [5] of multiple-period TDES models and to execute further operations. All the operations and the generated files are recorded in an annotated file MAKEIT.TXT. The procedures utilized in this study are summarized in Appendix 1. The edit procedure is utilized to convert a multiple-period periodic task to a task with a fixed-period or vice-versa. In this study, a task with a superscript “I" (resp. “U") represents that it possesses the lower (resp. upper) period bound; the corresponding task name in TTCT is prefixed by an L (resp. U).

1http://www.control.utoronto.ca/DES
The composite model of an RTS is generated by the synchronous product of all the tasks [5], [21].

1) MCW RTS generation:
Suppose that tasks \( \tau_1 \) and \( \tau_2 \) are running in RTS \( \Sigma^0 \). We generate \( \Sigma^0 \) by the following TTCT operations (all the sync operations in the original MAKEIT file were reported with the message “Blocked_events = None”, eliminated here for readability):

2) MJGS RTS generation:
The MJGS model for an RTS \( \Sigma^0 \) executing \( \tau_1 \) and \( \tau_2 \) is represented by a generator LJ that is generated as follows.

V. Supervisory Control of Dynamic Reconfigurable Multiple-Period RTS
The event controllability and the supervisory control in this study follow the principles proposed in [5], [6], and [21].

A. General Specification for MCW Model
In this present paper, instead of utilizing the method proposed in [5] to dynamically revise the specification for the tasks running in the uni-processor, a general specification S with

<table>
<thead>
<tr>
<th>Task</th>
<th>TDES</th>
<th>Type</th>
<th>Period</th>
<th>( R )</th>
<th>( C )</th>
<th>( D )</th>
<th>( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_a )</td>
<td>( G_u )</td>
<td>M</td>
<td>PA</td>
<td>0</td>
<td>1</td>
<td>[2, 3]</td>
<td>[2, 3]</td>
</tr>
<tr>
<td>( \tau_a )</td>
<td>( G_u )</td>
<td>F</td>
<td>LPA</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( \tau_b )</td>
<td>( G_b )</td>
<td>F</td>
<td>PB</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

### TABLE III: Parameters of MJGS Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>TDES</th>
<th>Type</th>
<th>Period</th>
<th>( R )</th>
<th>( C )</th>
<th>( D )</th>
<th>( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_a )</td>
<td>( G_u )</td>
<td>M</td>
<td>PA</td>
<td>0</td>
<td>1</td>
<td>[2, 3]</td>
<td>[2, 3]</td>
</tr>
<tr>
<td>( \tau_a )</td>
<td>( G_u )</td>
<td>F</td>
<td>LPA</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( \tau_b )</td>
<td>( G_b )</td>
<td>F</td>
<td>PB</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

SYS0 = sync (TASK1, TASK2) (425, 644)
where “(425, 644)” denotes that \( \Sigma^0 \), represented by SYS0, has 425 states and 644 transitions. Suppose that another RTS \( \Sigma^1 \), represented by SYS1, contains \( \tau_1 \), \( \tau_2 \), and \( \tau_3 \). It is generated based on \( \Sigma^0 \) as follows.

SYS1 = sync (SYS0, TASK3) (8500, 16367)
The composite task model of traditional periodic RTS is generated by the technique proposed in [5]. In this study, by choosing the periodic tasks with the lower (resp. upper) bound of periods, we generate \( \Sigma^0 \) (LSYS0), \( \Sigma^1 \) (LSYS1), and \( \Sigma^0 \) (USYS1) as follows. They are the counterparts of \( \Sigma^0 \) and \( \Sigma^1 \) with fixed-periods.

LSYS0 = sync (TASK1, LTASK2) (255, 364)
LSYS1 = sync (LSYS0, LTASK3) (2550, 4475)
USYS1 = sync (TASK1, UTASK2) (425, 610)
USYS1 = sync (USYS1, UTASK3) (1750, 3064)

Finally, the five generated MCW RTS are listed in Table IV; they will be utilized in the supervisory control and evaluation of the closed behavior of the controlled RTS.

<table>
<thead>
<tr>
<th>System</th>
<th>Type</th>
<th>TTCT</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Sigma^0 )</td>
<td>M</td>
<td>SYS0</td>
<td>( \tau_1 ), ( \tau_2 )</td>
</tr>
<tr>
<td>( \Sigma^1 )</td>
<td>M</td>
<td>SYS1</td>
<td>( \tau_1 ), ( \tau_2 ), ( \tau_3 )</td>
</tr>
<tr>
<td>( \Sigma^0 )</td>
<td>F</td>
<td>LYS0</td>
<td>( \tau_1 ), ( \tau_2 )</td>
</tr>
<tr>
<td>( \Sigma^1 )</td>
<td>F</td>
<td>LYS1</td>
<td>( \tau_1 ), ( \tau_2 ), ( \tau_3 )</td>
</tr>
<tr>
<td>( \Sigma^0 )</td>
<td>F</td>
<td>USYS1</td>
<td>( \tau_1 ), ( \tau_2 ), ( \tau_3 )</td>
</tr>
</tbody>
</table>

### TABLE IV: Parameters of MCW Systems
\[ L(S) = L(S_1)||L(S_2)||\cdots||L(S_n) \]

is defined with event set \( \Sigma = \bigcup_{1 \leq i \leq n} \Sigma_i \), the union of the event sets of all the potential tasks which may be called by the processor. Let \( L(S) = E \subseteq \Sigma^* \). Moreover, \( L_m(G) \subseteq \Sigma^* \) is always satisfied. Hence, by Theorem 1 (Theorem 3.5.2 in [21]), \( K \) can be found by the procedure \texttt{supcon} (see Appendix 1).

**Theorem 1**: [21] Let \( E \subseteq \Sigma^* \) and let \( K = \sup \mathcal{C}(E \cap L_m(G)) \). If \( K \neq \emptyset \) then there exists a marking nonblocking supervisory control (MNSC) for \( G \) such that \( L_m(V/G) = K \).

In order to utilize SCT to schedule the RTS non-preemptively, the specifications are defined to ensure that after the occurrence of \( \alpha_i \), no other event \( \alpha_j \) with \( j \neq i \) can occur to preempt it. Hence, the TDES model of specification \( S_i \) for task \( \tau_i \) is illustrated in Fig. 14, in which \( \alpha_j \) and \( \beta_j \) with \( j \neq i \) represent events \( \alpha \) and \( \beta \) for any other task, respectively.

The symbol * represents the other events in \( \Sigma \). As listed in Appendix 3, the specifications for \( G_1 \), \( G_2 \), and \( G_3 \) are created by TTCT. Thereafter, by utilizing \texttt{sync}, the general specification \( S \) presented in Fig. 15 is generated.

**C. Dynamic Reconfiguration of RTS**

For both CW and JGS models, the reconfiguration process is illustrated in Fig. 17, which in this study is extended into a two-step approach. In the next section we will illustrate the supervisory control and reconfiguration of two real-world examples.

\[ U_{\min} \leq U_S \leq U_{\max}. \]

All possible safe execution sequences are found, resulting in a decrease of processor utilization. Users should take the responsibility to provide the tolerable lowest processor utilization \( U_{\min} \). Consequently, any safe execution sequence in the supervisor can be selected as a guide to schedule the RTS by dynamically reconfiguring the period of each task. If the supervisor is still empty, we claim that the system is non-schedulable.

**VI. EXAMPLES**

The reconfigurations in this paper are based on the revised versions of the two examples studied in [5] and [6], respectively.
A. Dynamic Reconfiguration of MCW Model

In this study, as illustrated in Fig. 18, the example of a motor network studied in [5] is revised and considered as a reconfigurable RTS. Suppose that three electric motors are controlled by a uni-processor. As depicted in Fig. 18, their deadlines and the periods are represented by D and T, respectively. At each time only a subset of these motors is called by the processor. Their parameters coincide with those of the tasks shown in Table II as

- Motor 1: $\tau_1$
- Motor 2: $\tau_2$
- Motor 3: $\tau_3$

Suppose that the motor network has two work plans, coinciding with the defined RTS $S^0$ and $S^1$:

- Plan 1. use only Motors 1 and 2; and
- Plan 2. use all three motors.

![Motor Network Example](image)

Fig. 18: A motor network example.

1) Supervisory Control of $S^0_0$

Take $S^0_0$ (LSYS0) as an example. All the safe execution sequences are calculated by the procedure supcon, i.e.,

**LSUPER0 = supcon** (LSYS0, SPEC) (153, 190).

Since LSUPERO is not empty, $S^0_0$ is schedulable at processor utilization $U_{\text{max}} = 1/5 + 2/4 = 0.7$. The safe execution sequence set in LSUPERO is represented by a TDES with 153 states and 190 transitions. By projecting out all events but $\alpha_i$, i.e.,

**PJLSUPER0 = project** (LSUPER0, Image [11, 21]) (12, 15)

![Scheduling Map of $S^0_0$](image)

Fig. 19: Scheduling map of $S^0_0$.

we obtain the scheduling map illustrated in Fig. 19, which contains 12 states and 15 transitions. PJLSUPER0 provides eight safe execution sequences to schedule the RTS with processor utilization being 0.7:

1. $\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7\alpha_8$
2. $\alpha_1\alpha_2\alpha_3\alpha_4\alpha_5\alpha_6\alpha_7\alpha_8$

For comparison, the EDF scheduling result of $S^0_0$ by Cheddar [23] is displayed in Fig. 20, which coincides with Sequence (1.) above within PJLSUPER0. Sequence (8.), depicted in Fig. 21, can never be generated by EDF. By comparing the two sequences in Figs. 20 and 21, in case $\tau_2$ (with the earliest deadline) cannot arrive on time at $t = 4$, then according to the multiple sequences users can choose another available sequence shown in Fig. 19 to schedule task $\tau_1$ first. Thus, recalculating the scheduling sequences is unnecessary. However, there is no EDF sequence to schedule task $\tau_2$ first. If $\tau_2$ cannot arrive on time, the EDF scheduling cannot execute $S^0_0$ successfully. The supervisory control technique provides a greater number of safe execution sequences as compared to EDF scheduling. Intuitively, because $L(G_2^1) \subset L(G_2)$ and $L_m(G_2^1) \subset L_m(G_2)$, the safe execution sequences in $S^0_0$ should be a proper subset of the safe execution sequences of $S_0$. This is proved as follows.

![Scheduling Map of $S^0_0$ in Cheddar](image)

Fig. 20: Scheduling map of $S^0_0$ in Cheddar.

![Scheduling of Sequence (8.)](image)

Fig. 21: Scheduling of Sequence (8.).

By calling procedure supcon, all the safe execution sequences of the multiple-period version RTS $S^0$ are obtained. By using the procedure complement, we obtain the set of the behaviors prohibited by SUPER0, which is contained in CSUPER0. By computing the meet of CSUPER0 and LSUPER0, if the trim [21] version of meet is empty, this represents that the reachable and coreachable sequences within LSUPERO are not in CSUPER0. Hence, LSUPER0 is a proper subset of SUPER0. The corresponding TTCT operations are

**SUPER0 = supcon** (SYS0, SPEC) (263, 362)

**CSUPER0 = complement** (SUPER0, [$\{}$] (264, 1848)

**TEST = meet** (CSUPER0, LSUPER0) (156, 195)

**TEST = trim** (TEST) (0, 0)

The scheduling map for $S^0_0$ is more complex than that for $S^0_0$, which has 50 states and 86 transitions, i.e.,

**PJSUPER0 = project** (SUPER0, Image [11, 21]) (50, 86)

Evidently, even though the supervisor for $S^0_0$ excludes some safe execution sequences of $S^0$, the scheduling map still provides more choices than the EDF scheduling algorithm.
2) Dynamic Reconfiguration of $S^1_u$:

The set of safe execution sequences of $S^1_u$ (LSYS1) is found by the procedure $\text{supcon}$ is empty, i.e., $\text{LSUPER1} = \text{supcon}$ (LSYS1, SPEC) (0, 0).

According to the CW model, $S^1$ is non-schedulable at processor utilization $U_{\text{max}} = 1/5 + 2/4 + 2/3 > 1$. Thus, we need to reconfigure the system to be the multi-period model $S^1_1\ldots_2$ and utilize SCT again to find the safe execution sequences by $\text{SUPER1} = \text{supcon}$ (SYS1, SPEC) (2180, 3681).

This represents that $\text{supcon}$ finds all the possible safe execution sequences between the processor utilization $U_{\text{min}} = 1/5 + 2/6 + 2/5 < 1$ and $U_{\text{max}} = 1/5 + 2/4 + 2/3 > 1$.

The system is finally schedulable since $\text{SUPER1}$ is nonempty. In order to find the scheduling map after the reconfiguration, we need to call $\text{project}$. However, TTCT fails to output the result of projecting onto events $\alpha_i$. The reason is that the dynamic reconfiguration of the periods (event $\gamma_i$) violates the observer property discussed in [21]. However, we choose the following method to view a part of the scheduling map of the reconfigured RTS $S^1_1$:

Step 1:

We choose $S^1_u$ as a subset of the composite task model of $S^1$, based on which we find the safe execution sequence set, containing 417 states and 574 transitions. The scheduling map is calculated by projecting the safe execution sequences onto events $\alpha_1$, $\alpha_2$, and $\alpha_3$; it contains 37 states and 54 transitions, as seen in Fig. 22. The corresponding TTCT operations are given as follows.

$\text{USUPER1} = \text{supcon}$ (USYS1, SPEC) (417, 574)

$\text{PJUSUPER1} = \text{project}$ (USUPER1, Image [11, 21, 31]) (37, 54)

Fig. 22: Scheduling map of $S^1_u$.

Step 2:

We can verify that $S^1_u$ is a proper subset of $S^1$ via the following TTCT procedures:

$\text{CSUPER1} = \text{complement}$ (SUPER1, [ ]) (2181, 21810)

$\text{TEST} = \text{meet}$ (CSUPER1, USUPER1) (417, 574)

$\text{TEST} = \text{trim}$ (TEST) (0, 0)

Finally, we claim that, after the reconfiguration, the scheduling map of $S^1$ is at least as complex as that presented in Fig. 22. More precisely, SUPER1 (resp., USUPER1) contains 2180 (resp., 417) states and 3681 (resp., 574) transitions. Intuitively, the scheduling map of SUPER1 should be more complex than that depicted in Fig. 22, in which the periods are dynamically reconfigured. The EDF scheduling of $S^1_u$ by Cheddar is illustrated in Fig. 23. It can find only one schedulable sequence. Moreover, no sequence for the multi-period RTS can be found by EDF scheduling in Cheddar.

Fig. 23: Scheduling map of $S^1_u$ in Cheddar.

3) Comparison with the CW model:

In LSUPER0 (CW model), every scheduling sequence is based on the fixed period of each task. The processor utilization of each task is fixed permanently. For example, we randomly choose a sequence $\gamma_1\alpha_1\gamma_2\beta_1\alpha_2\gamma_3\beta_2\gamma_4 \ldots$. By projecting out $\gamma_1$, $\alpha_1$, and $\beta_1$, we obtain $\gamma_2\beta_1\alpha_2\gamma_3\beta_2\gamma_4 \ldots$. We have $T_2 = 4$ and $U_2 = 2/4$.

In SUPER0 (MCW model), we randomly choose two sequences as follows:

1. $\gamma_1\alpha_1\gamma_2\beta_1\alpha_2\gamma_3\beta_2\gamma_4 \ldots$
2. $\gamma_1\alpha_1\gamma_2\beta_1\alpha_2\gamma_3\beta_2\alpha_3\gamma_4 \ldots$

By projecting out $\gamma_1$, $\alpha_1$, and $\beta_1$ in Sequence (1.), we obtain $\gamma_2\alpha_2\gamma_3\beta_2\gamma_4 \ldots$. By projecting out $\gamma_1$, $\alpha_1$, and $\beta_1$ in Sequence (2.), we obtain $\gamma_2\alpha_2\gamma_3\beta_2\alpha_3\gamma_4 \ldots$ Evidently, $T_2 = 5$ and $T_2 = 4$ are in two adjacent periods. In the second period of the execution of $\tau_2$, its processor utilization is changed from 2/5 to 2/4 to speed up the scheduling process. This means that, according to the processor utilization interval predefined by the users, the processor utilization of the RTS is dynamically changed at run-time.

By comparing Sequences (1.) and (2.), we see that after the occurrence of substring $\gamma_1\alpha_1\gamma_2\beta_1$, the controller provides at least two subsequences in Sequences (1.) and (2.) to schedule $\tau_2$. However, neither the CW scheduling nor the EDF scheduling can provide such scheduling plans.

B. Dynamic Reconfiguration of MIGS model

1) Dynamic Reconfiguration of $S^1_u$:

Consider a water vessel system [6] represented by two tasks as listed in Table III. The first task $\tau_a$ is assigned with a multiple-period [2, 3]. The RTS cannot be scheduled according to the initial plan with processor utilization $U_{\text{max}} = 1/2 + 2/3 \geq 1$, i.e.,

$LJ = \text{trim}$ (LJ) (0, 0)

The full scheduling map of $S^1_u$ is empty, which means that the system represented by the JGS model is non-schedulable. In order to reconfigure $S^1_u$, $u_n$ is revised to be $\tau_a$ with a multiple-period. Then we obtain the full schedule map shown in Fig. 24 by

$J = \text{trim}$ (J) (15, 29)

$J = \text{project}$ (J, Null [11]) (13, 16)

During the reconfiguration, $T_n = 3$ is selected automatically.

The processor utilization is $U_{\text{min}} = 1/3 + 2/3 = 1$ and thus the RTS is schedulable. According to [6], by assigning higher
priorities for task \( \tau_a \) and \( \tau_b \), we obtain the safe execution sequences shown in Figs. 25 and 26, respectively.

![Fig. 24: The full scheduling map.](image)

If we only use the non-preemptive specification, without considering the priority of each task, to calculate the controller, i.e.,

\[
\text{SUPER} = \text{supcon} (J, PRB) \quad (13, 15)
\]

we obtain the scheduling map as shown in Fig. 27.

![Fig. 25: Task \( \tau_a \) with a higher priority.](image)

![Fig. 26: Task \( \tau_b \) with a higher priority.](image)

![Fig. 27: Non-preemptive scheduling.](image)

2) Comparison with the JGS model:

Suppose that the initial period of \( \tau_a \) is \( T_a = 3 \). The system is schedulable based on the technique proposed in [6]. However, the real time scheduling in [6] is priority-based. Thereafter, the non-preemptive scheduling can be based only on the scheduling map shown in Fig. 25 or 26. In that case the scheduling shown in Fig. 27 can never be found. Obviously, the real-time scheduling in this paper is more general.

VII. DISCUSSION

A. Computational Complexity

The real-time scheduling and reconfiguration are based on the computation of the supremal controllable sublanguage with respect to a finite TDES. According to [5] and [24], the computation of the supremal controllable sublanguage with respect to a finite TDES can be completed in polynomial time. The computational complexity of the supremal sublanguage of a specification is \( O(m^2n^2) \) where \( m \) and \( n \) are the size of the final state set of the plant \( G \) and the specification \( S \), respectively.

In this work, the increase of a period is obtained by explicitly adding the tick event. If an RTS executes a large set of periodic tasks that are assigned with large periods, then the number of states and transitions will be increased significantly. Similar to [5], this remains a challenge in the "scaling up" of the proposed method for the reconfiguration based on SCT. Two approaches may be explored to deal with this difficulty. One approach, namely modular synthesis [25], may be applied to reduce computational overload by synthesizing a set of modular supervisors which can achieve the same result as a centralized supervisor does. Another approach is to use supervisory control of the timed version of state-tree-structures [26] to manage the state explosion problem in the calculation of the supervisors.

B. Comparison with Other Reconfiguration Methods

Job skipping can be utilized by an RTS to execute "occasionally skippable" tasks, such as video reception, telecommunications, packet communication, and aircraft control [17]. However, industrial production lines should avoid job skipping since it will increase the manufacturing cost. As another approach, the elastic scheduling model [7]–[10] can be utilized to guarantee that no deadline is missed during the manufacturing process in industrial applications [27]. However, both reconfiguration approaches can only provide a single sequence on-line. Moreover, even though all the deadlines are satisfied by the elastic scheduling model, the processor utilization of some tasks is decreased.

For industrial production lines or manufacturing processes, the technique presented in this study reconfigures an RTS that executes a set of tasks with the same task scale studied in [5]–[10]. We suggest that users redefine an acceptable processor utilization interval for each task. If no safe execution sequence can be found at the highest processor utilization, SCT is utilized to provide all the possible safe execution sequences off-line. Moreover, even though all the deadlines are satisfied by the elastic scheduling model, the processor utilization of some tasks is decreased.

Both job skipping and the elastic task model need to calculate the processor demand. However, the method provided in this study does not need to calculate the processor demand. The comparison is shown in Table V.

<table>
<thead>
<tr>
<th>method</th>
<th>on-line</th>
<th>single sequence</th>
<th>processor demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>job skipping</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>elastic task</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>this work</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
VIII. CONCLUSION

This study presents a formal constructive method for real-time periodic tasks with multiple-periods via a TDES model. The lower and upper bounds of the period of such a model are predefined by users for the purpose of dynamic reconfiguration. The formal SCT of TDES can be considered as a rigorous analysis and synthesis tool to dynamically reconfigure the non-preemptive scheduling of hard RTS. Suppose that in every scheduling plan only a subset of tasks of an RTS is called by the processor. Instead of dynamically updating the specification for the tasks running in the uni-processor, a general specification is presented, which guarantees that all the potential tasks called by the processor can be scheduled non-preemptively. In case an RTS is claimed by [5] or [6] to be non-schedulable, the presented two-step dynamic reconfiguration approach can be utilized to find all the safe execution sequences (possible reconfiguration scenarios) of each task in the RTS. These sequences provide more choices than the EDF scheduling algorithm. The processor and the real-time tasks are general models for real-world hard RTS. Similar to [5] and [6], the multiple-period model can be utilized to describe the behavior of a manual assembly process or a robotic pick-and-place operation that is executed by a processor that could be a water vessel system, computer numerical control (CNC) machine, a robot, or an assembly-line worker. This leads to the possibility that the off-line reconfiguration method can be implemented in practical contexts based on reconfigurable real-time scheduling. In future work, we will focus on the dynamic reconfiguration of RTS processing asynchronous tasks and sporadic tasks. The real-time scheduling can be preemptive or non-preemptive.

REFERENCES


Appendix 1.

TTG synthesis procedures in TTCT [21], in which TDES are represented by TDS:

TDS = create (TDS) is a new discrete-event system (TDS).
Option 1 allows fast user input via a sequence of prompts, resulting in direct creation of a .TDS file. Option 2 allows the user to create a text (.ATS) file with any ASCII text editor; this file can be converted to a .TDS file using the TCT procedure FD.
TDS2 = allexvents (TDS1) is a marked one-state TDS self-looped with all the events of TDS1.
TDS2 = complement (TDS1, [AUXILIARY-EVENTS]) is a generator of the marked language complementary to the marked language of TDS1, with respect to the extended alphabet comprising the event labels of TDS1 plus those in the auxiliary-event list. The closed behavior of TDS2 is all strings over the extended alphabet.
TDS2 = edit (TDS1) is obtained from TDS1 by user-selected modifications.

TDS2 = minstate (TDS1) is a minimal state TDS that generates the same closed and marked languages as TDS1, and the same string mapping induced by vocalization (if any). TDS2 is reachable, but not coreachable unless TDS1 is coreachable.

TDS2 = project (TDS1, [NULL/IMAGE EVENTS]) is a generator of the projected closed and marked languages of TDS1, under the natural projection specified by the listed Null or Image events. In decentralized control, TDS2 could be an observer’s local model of TDS1.

TDS2 = trim (TDS1) is the trim (reachable and coreachable) substructure of TDS1.

TDS2 = meet (TDS1, TDS2) is the meet (reachable cartesian product) of TDS1 and TDS2. TDS2 need not be coreachable.

TDS3 = supcon (TDS1, TDS2) is a trim generator for the supremal controllable sublanguage of the marked legal language generated by TDS2 with respect to the plant TDS1.

TDS3 provides a proper supervisor for TDS1.

TDS3 = sync (TDS1, TDS2) is the (reachable) synchronous product of TDS1 and TDS2.

Appendix 2.

Corresponding TTCT MAKEIT file for all the created tasks:
TASK1 = create (TASK1, [mark 0], [tran [0, 10, 1], [1, 0, 0], [1, 0, 6], [1, 1, 2], [2, 0, 3], [3, 12, 4], [4, 0, 8], [5, 0, 9], [5, 11, 6], [6, 0, 7], [7, 12, 8], [8, 0, 12], [9, 0, 13], [9, 11, 10], [10, 0, 11], [11, 12, 12], [12, 0, 16], [13, 11, 14], [14, 0, 15], [15, 12, 16], [16, 0, 0], [forcible 10, 11, 12]) (17, 20)

TASK2 = create (TASK2, [mark 0, 15, 20], [tran [0, 20, 1], [1, 0, 6], [1, 21, 2], [2, 0, 3], [3, 0, 4], [4, 22, 5], [5, 0, 10], [6, 0, 11], [6, 21, 7], [7, 0, 8], [8, 0, 9], [9, 22, 10], [10, 0, 15], [11, 0, 16], [11, 21, 12], [12, 0, 13], [13, 0, 14], [14, 22, 15], [15, 0, 20], [16, 21, 12], [16, 0, 21], [17, 0, 18], [18, 0, 19], [19, 22, 20], [20, 0, 0], [20, 20, 1], [21, 21, 22], [22, 0, 23], [23, 0, 24], [24, 22, 0], [forcible 20, 21, 22]) (25, 32)

LTASK2 = edit (LTASK2, [trans [-11, 0, 16], [-15, 0, 20]]) (25, 29)

LTASK2 = trim (LTASK2) (16, 18)

LTASK2 = minstate (LTASK2) (15, 17)

UTASK2 = edit (UTASK2, [mark [-15], [-20]], [trans [-15, 20, 1], [-20, 20, 1]]) (25, 29)

LTASK3 = create (LTASK3, [mark 0], [tran [0, 30, 1], [1, 0, 6], [1, 31, 2], [2, 0, 3], [3, 0, 4], [4, 32, 5], [5, 0, 6], [6, 31, 7], [7, 0, 8], [8, 0, 9], [9, 32, 0], [forcible 30, 31, 32]) (10, 11)

TASK3 = edit (LTASK3, [mark [+10], [+15]], [trans [+5, 0, 10], [+6, 0, 11], [+9, 32, 10], [+10, 0, 15], [+10, 30, 1], [+11, 0, 16], [+11, 31, 12], [+12, 0, 13], [+13, 0, 14], [+14, 32, 15], [+15, 0, 0], [+15, 30, 1], [+16, 31, 17], [+17, 0, 18], [+18, 0, 19], [+19, 32, 0], [+5, 0, 0], [+9, 32, 0], [20, 25])

UTASK3 = edit (UTASK3, [mark [-10], [-15]], [trans [-10, 30, 1], [-15, 30, 1]]) (20, 23)

TASKA = create (TASKA, [mark 0], [tran [0, 10, 1], [1, 13, 2], [2, 0, 0], [forcible 1 3]) (3, 3)

TASKA = selfloop (TASKA, [0, 20, 21, 23], [new forcible 21, 23]) (3, 15)

TASKA = edit (TASKA, [trans [-2, 0, 2], [-2, 20, 2], [-2, 21, 2], [-2, 23, 2]]) (3, 11)

TASKB = create (TASKB, [mark 0], [tran [0, 20, 1], [1, 21, 2], [2, 0, 3], [3, 23, 4], [4, 0, 0]], [forcible 21, 23]) (5, 5)

TASKB = selfloop (TASKB, [0, 10, 11, 13], [new forcible 11, 13]) (5, 25)

TASKB = edit (TASKB, [trans [-2, 0, 2], [-2, 10, 2], [-2, 11, 2], [-2, 13, 2], [-4, 0, 4], [-4, 10, 4], [-4, 11, 4], [-4, 13, 4]) (5, 17)

SYS = sync (TASKA, TASKB) (13, 34)

ALLSYS = allies (SYS) (1, 7)

PA = create (PA, [mark 0, 3], [tran [0, 10, 1], [1, 0, 2], [2, 0, 3], [3, 0, 0], [3, 10, 1]]) (4, 5)

PA = sync (PA, ALLSYS) (4, 25)

LP A = create (LP A, [mark 0], [tran [0, 10, 1], [1, 0, 2], [2, 0, 0]]) (3, 3)

LP A = sync (LP A, ALLSYS) (3, 18)

PB = create (PB, [mark 0], [tran [0, 20, 1], [1, 0, 2], [2, 0, 3], [3, 0, 0]]) (4, 4)

PB = sync (PB, ALLSYS) (4, 24)

Appendix 3.

The generated files for the specifications are recorded, where the SPEC with 4 states and 22 transitions is the general one.

ALLSYS1 = allies (LSYS1) (1, 10)

SPEC1 = create (SPEC1, [mark 0], [tran [0, 11, 1], [0, 21, 0], [0, 22, 0], [0, 31, 0], [0, 32, 0], [1, 12, 0], [forcible 11, 12, 21, 22, 31, 32]) (2, 6)

SPEC1 = sync (SPEC1, ALLSYS1) (2, 14)

SPEC2 = create (SPEC2, [mark 0], [tran [0, 11, 0], [0, 12, 0], [0, 21, 0], [0, 31, 0], [0, 32, 0], [1, 22, 0], [forcible 11, 12, 21, 22, 31, 32]) (2, 6)

SPEC3 = create (SPEC3, [mark 0], [tran [0, 11, 0], [0, 12, 0], [0, 21, 0], [0, 31, 0], [0, 32, 0], [1, 22, 0], [forcible 11, 12, 21, 22, 31, 32]) (2, 6)

SPEC = sync (SPEC1, SPEC2) (3, 18)

SPEC = sync (SPEC, SPEC3) (4, 22)

PRB = create (PRB, [mark 0], [tran [0, 0, 0], [0, 10, 0], [0, 11, 0], [0, 13, 0], [0, 20, 0], [0, 21, 0], [0, 23, 0], [1, 0, 1], [1, 10, 1], [1, 20, 1], [1, 21, 1], [1, 23, 0], [forcible 11, 13, 23, 1]) (2, 12)