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.

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

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, Nasser 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 Nasser-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 $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$, where Q is the state set, Σ is the event set, $\delta: Q \times \Sigma \rightarrow Q$ is the (partial) state transition function, q_0 is the initial state, and Q_m is the marker state set satisfying $Q_m \subseteq Q$. Let Σ^+ (resp., ϵ) denote the set of all finite sequences over Σ (resp., empty string). We have $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$. A plant and a specification are represented by $\mathbf{G}$ and $\mathbf{S}$, respectively. In [22], by adjoining to the RW framework time bounds on the transitions, $\mathbf{G}$ starts from an (untimed) activity transition graph (ATG) $\mathbf{G}_{act} = (A, \Sigma_{act}, \delta_{act}, a_0, A_m)$ with $\Sigma := \Sigma_{act} \dot{\cup} \{tick\}$. The elements of the activity set A are "activities", denoted by a . Σ_{act} is partitioned into two subsets, $\Sigma_{act} = \Sigma_{spe} \dot{\cup} \Sigma_{rem}$, where Σ_{spe} (resp. Σ_{rem}) is the prospective (resp. remote) event set with finite (resp. infinite) upper time bounds [21]. By defining the timer interval for σ , represented by T_σ , to be $[0, u_\sigma]$ or $[0, l_\sigma]$ for $\sigma \in \Sigma_{spe}$ and $\sigma \in \Sigma_{rem}$, respectively, the initial state is $q_0 := (a_0, \{t_{\sigma 0} | \sigma \in \Sigma_{act}\})$, where $t_{\sigma 0}$ equals u_σ or l_σ for a prospective or remote state, respectively. The marker state set is $Q_m \subseteq A_m \times \prod \{T_\sigma | \sigma \in \Sigma_{act}\}$. Thus a TDES is represented by $G = (Q, \Sigma, \delta, q_0, Q_m)$. An event $\sigma \in \Sigma_{act}$ is enabled at q if $\delta_{act}(a, \sigma)$ is defined, written $\delta_{act}(a, \sigma)!$; it is eligible if its transition $\delta(q, \sigma)$ is also defined, i.e., $\delta(q, \sigma)!$. The closed behavior of $\mathbf{G}$ is the language $L(\mathbf{G}) := \{s \in \Sigma^* | \delta(q_0, s)!\}$. In addition, the marked behavior of $\mathbf{G}$ is $L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) | \delta(q_0, s) \in Q_m\}$. $\mathbf{G}$ is non-blocking if $L_m(\mathbf{G})$ satisfies $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$, where $\overline{L_m(\mathbf{G})}$ is the (prefix) closure of $L(\mathbf{G})$. In a TDES plant $\mathbf{G}$, the eligible event set $Elig_{\mathbf{G}}(s) \subseteq \Sigma$ at a state q corresponding to a string $s \in L(\mathbf{G})$ is defined by $Elig_{\mathbf{G}}(s) := \{\sigma \in \Sigma | s\sigma \in L(\mathbf{G})\}$. For an arbitrary language $K \subseteq L(\mathbf{G})$, let $s \in \overline{K}$, $Elig_K(s) := \{\sigma \in \Sigma | s\sigma \in \overline{K}\}$. The set of all controllable sublanguages of K is denoted by $\mathcal{C}(K)$; 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 $\sup \mathcal{C}(K)$. Considering a specification language $E \subseteq \Sigma^*$, there exists an optimal monolithic supervisor $\mathbf{S}$. Its closed behavior is $L(\mathbf{S}) = \overline{L_m(\mathbf{S})}$, where $L_m(\mathbf{S})$ is the marked behavior represented by $L_m(\mathbf{S}) = \sup \mathcal{C}(E \cap L_m(\mathbf{G})) \subseteq L_m(\mathbf{G})$.

B. Synchronous product

Suppose that we have a set of generators $\mathbf{G}_i$ with $i \in \mathbf{n} = \{1, 2, \dots, n\}$. In accordance with [21], the behavior of $\mathbf{G}_i$ is represented by language L_i . Synchronous product [21] is a standard way to combine several DES into a single and more complex one. Suppose that we have two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ with $\Sigma = \Sigma_1 \cup \Sigma_2$. The natural projection $P_i: \Sigma^* \rightarrow \Sigma_i^*$ is defined by

- $P_i(\epsilon) = \epsilon$,
- $P_i(\sigma) = \begin{cases} \epsilon, & \text{if } \sigma \notin \Sigma_i \\ \sigma, & \text{if } \sigma \in \Sigma_i \end{cases}$,
- $P_i(s\sigma) = P_i(s)P_i(\sigma)$, $s \in \Sigma^*$, $\sigma \in \Sigma$.

The inverse image function of P_i is

$$P_i^{-1}: Pwr(\Sigma_i^*) \rightarrow Pwr(\Sigma^*).$$

For $H \subseteq \Sigma_i^*$,

$$P_i^{-1}(H) := \{s \in \Sigma^* | P_i(s) \in H\}.$$

The synchronous product of L_1 and L_2 , denoted by $L_1 || L_2$, is defined as

$$L_1 || L_2 := P_1^{-1}L_1 \cap P_2^{-1}L_2.$$

C. System Model

Suppose that a periodic RTS $\mathbb{S}$ processes n tasks, i.e., $\mathbb{S} = \{\tau_1, \tau_2, \dots, \tau_n\}$, $i \in \mathbf{n}$. Assume also that this set contains at least one task with a multiple-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 τ_i is described by $\tau_i = (R_i, C_i, D_i, T_i)$ with

- release time R_i ,
- WCET C_i ,
- hard deadline D_i , and
- multiple-period T_i .

An RTS is a synchronous system [19] in case all the processed tasks are released at the same time, namely $R_i = 0$. In this research the RTS is synchronous. A deadline is hard if its violation is unacceptable. A multiple-period is a period set containing several possible periods: the lower bound (i.e., shortest one) is represented by $T_{i_{min}}$, and the upper bound (i.e., longest one) is represented by $T_{i_{max}}$. Thus, we have $T_i = [T_{i_{min}}, T_{i_{max}}]$.

During the real-time scheduling process, for task τ_i , only one period T satisfying $T_{i_{min}} \leq T \leq T_{i_{max}}$ is selected in each scheduling period. The processor utilization U_i of task τ_i is calculated by

$$U_i = C_i/T.$$

The total processor utilization of $\mathbb{S}$ is $U_{\mathbb{S}} = \sum_{i=1}^n U_i$. An RTS $\mathbb{S}$ is not schedulable in case $U_{\mathbb{S}} > 1$ [11].

Task τ_i consists of an infinite sequence of jobs $J_{i,j} = (r_{i,j}, C_i, d_{i,j}, p_{i,j})$ repeated periodically. The absolute deadline $d_{i,j}$ denotes the global clock time at which the execution of $J_{i,j}$ must be completed. Similarly, we define the absolute

release time (resp. period) $r_{i,j}$ (resp. $p_{i,j}$) to mean the global clock time at which τ_i must be released (resp. start the next period). The subscript “ i, j ” of $J_{i,j}$ represents the j -th execution of task τ_i . For each j , $J_{i,j}$ requests the processor at global clock time $r_{i,j}$. Moreover, the execution of $J_{i,j}$ takes C_i ticks, which must be completed no later than $d_{i,j}$. The absolute deadline $d_{i,j}$ occurs no later than the absolute period $p_{i,j}$. The EDF scheduling algorithm [1] assigns the priority of each job based on the absolute deadlines: the earlier the deadline, the higher is the job’s priority. The EDF scheduling algorithm can be utilized to schedule RTS. At each time unit, the job with the highest priority enters the processor. If the execution of a job is allowed to be preempted by other jobs before its execution finishes, the scheduling is preemptive; otherwise, it is non-preemptive.

IV. TDES MODEL FOR REAL-TIME SYSTEMS

A. CW Model

The CW model [5] represents a real-time periodic task $\tau_i = (R_i, C_i, D_i, T_i)$, $i \in \mathbf{n}$, with $D_i \leq T_i$, by a TDES $\mathbf{G}_i = (Q_i, \Sigma_i, \delta_{i0}, q_{0i}, Q_{mi})$. As depicted in Fig. 1, the corresponding ATG part is $\mathbf{G}_{act} = (A_i, \Sigma_{acti}, \delta_{acti}, a_{0i}, A_{mi})$ 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
 - $\delta_{acti}(W_i, \beta_i) = Y_i$.
- $a_{0i} = Y_i$, and
- $A_{mi} = \{Y_i\}$.

Fig. 1: ATG of a real-time task.

States Y_i , I_i , and W_i represent that task τ_i is at states *delay*, *idle*, and *work*, respectively. The events in the alphabet Σ_i are

- γ_i : the event that τ_i is released,
- α_i : the execution of τ_i is started, and
- β_i : the execution of τ_i is finished.

Event α_i is controllable and events γ_i and β_i are uncontrollable. Moreover, all the events in Σ_{acti} are forcible. Suppose that, after enabling, events γ_i , α_i , and β_i should wait for t_{γ_i} , t_{α_i} , and t_{β_i} ticks, respectively, until they are eligible to occur. Thus, t_{α_i} is the time at which τ_i starts its execution. Furthermore, in the CW model, $t_{\beta_i} = C_i$. A CW model has the following two features: 1) γ_i signals that after $r_{i,1}$, τ_i will release at every T_i ticks periodically; and 2) β_i must occur before τ_i is released again. The time interval between

the occurrences of events β_i and γ_i is the remaining time of the current period, which decreases along with the increase of t_{α_i} . Hence, in two adjacent periods, the values of t_{γ_i} could be different. Formally,

- γ_i has time bounds $\begin{cases} [0, 0], & \text{if } \tau_i \text{ releases at } r_{1,1} \\ [T_i - t_{\alpha_i} - t_{\beta_i}, T_i - t_{\alpha_i} - t_{\beta_i}], & \text{if } (\forall j > 1) \tau_i \text{ releases at } r_{i,j} \end{cases}$,
- α_i has time bounds $[0, D_i - t_{\beta_i}]$, and
- β_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 τ_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 $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \dots \cup \Sigma_n$, and $\Sigma^t = \Sigma \cup \{t\}$. The event set Σ_i for τ_i is composed of

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

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

Fig. 2: JGS WCET model.

Fig. 3: JGS period model.

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 I: CW model v.s. JGS model

Model	$D \leq T$	priority	preemption
CW	Y	N	N
JGS	N	Y	Y

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

336 3. The remaining time between β_i and γ_i equals 0 if 1)
 337 $D_i = T_i$ and 2) α_i occurs at time $D_i - t_\beta$. As a result,
 338 the occurrence of β_i may lead the TDES model to state
 339 y_0 directly. For example, suppose that we have two other
 340 tasks $\tau_2 = (0, 2, 6, [4, 6])$ and $\tau_3 = (0, 2, 5, [3, 5])$. The
 341 corresponding TTG models G_2 and G_3 are illustrated in Figs.
 342 6 and 7, respectively, in which events β_2 and β_3 lead the
 343 TDES model to the **initial** states directly. All the possible
 344 processor utilizations of τ_2 are $2/4$, $2/5$, and $2/6$; and the
 345 possible processor utilizations of τ_3 are $2/3$, $2/4$, and $2/5$.

Fig. 6: Multiple-period TDES G_2 .

Fig. 7: Multiple-period TDES G_3 .

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

349 2) Multiple-period JGS (MJGS) model:

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

$$(q \in \{T_{i_{min}}, y_{i_{min}+1}, \dots, y_{i_{min}-1}, y_0\}) \Rightarrow \delta(q, a_i) = 0$$

represents the new arrival of task τ_i .

Fig. 8: **Revised** WCET of JGS model.

Fig. 9: Multiple-period of a JGS model.

Example.

365 Suppose that we have two tasks $\tau_a = (0, 1, [2, 3], [2, 3])$ and
 366 $\tau_b = (0, 2, [3, 3], [3, 3])$. The WCET of τ_a and τ_b are illustrated
 367 in Figs. 10 and 11, respectively. The **multiple-periods** of τ_a
 368 and τ_b are shown in Figs. 12 and 13, respectively. The processor
 369 utilization of τ_a could be $1/2$ or $1/3$. The processor utilization
 370 of τ_b is $2/3$.
 371

Fig. 10: WCET of τ_a .

Fig. 11: WCET of τ_b .

E. Task Creation and Editing in TTCT

372 The TDES synthesis procedure TTCT¹ is a software pack-
 373 age to create an RTS as the **composite model** [5] of **multiple-**
 374 **period** TDES models and **to execute** further operations. All the
 375 operations and the generated files are recorded in an annotated
 376 file **MAKEIT.TXT**. The procedures utilized in this study are
 377 summarized in Appendix 1. The **edit** procedure is utilized to
 378 convert a multiple-period periodic task to a task with a fixed-
 379 period or vice-versa. In this study, a task with a superscript
 380 “l” (resp. “u”) represents that it possesses the lower (resp.
 381 upper) period **bound**; the corresponding task name in TTCT
 382 is prefixed by an L (resp. U).
 383

¹ <http://www.control.utoronto.ca/DES>

Fig. 12: Multiple-period of τ_a . Fig. 13: Multiple-period of τ_b .

1) Task creation for MCW model:

The TTCT operations for the creation and editing of tasks τ_1 , τ_2 , and τ_3 , represented by $\mathbf{G}_1$, $\mathbf{G}_2$, and $\mathbf{G}_3$, respectively, are reported in Appendix 2. In addition, the parameters of these created tasks are presented in Table II, in which the types “M” and “F” in Table II denote that the corresponding tasks possess multiple-periods or fixed-periods (as in a CW model), respectively. In the TDES models, events γ_i , α_i , β_i , and $tick$ are represented by $i0$, $i1$, $i2$, and 0 , respectively. Evidently, we have $L(\mathbf{G}_2^l) \subseteq L(\mathbf{G}_2)$, $L(\mathbf{G}_2^u) \subseteq L(\mathbf{G}_2)$, $L(\mathbf{G}_3^l) \subseteq L(\mathbf{G}_3)$, $L(\mathbf{G}_3^u) \subseteq L(\mathbf{G}_3)$; and $L_m(\mathbf{G}_2^l) \subseteq L_m(\mathbf{G}_2)$, $L_m(\mathbf{G}_2^u) \subseteq L_m(\mathbf{G}_2)$, $L_m(\mathbf{G}_3^l) \subseteq L_m(\mathbf{G}_3)$, $L_m(\mathbf{G}_3^u) \subseteq L_m(\mathbf{G}_3)$.

TABLE II: Parameters of MCW Tasks

task	TDES	type	TTCT	R	C	D	T
τ_1	$\mathbf{G}_1$	M	TASK1	0	1	4	5
τ_2	$\mathbf{G}_2$	M	TASK2	0	2	6	[4, 6]
τ_2^l	$\mathbf{G}_2^l$	F	LTASK2	0	2	4	4
τ_2^u	$\mathbf{G}_2^u$	F	UTASK2	0	2	6	6
τ_3^l	$\mathbf{G}_3^l$	F	LTASK3	0	2	3	3
τ_3	$\mathbf{G}_3$	M	TASK3	0	2	5	[3, 5]
τ_3^u	$\mathbf{G}_3^u$	F	UTASK3	0	2	5	5

2) Task creation for MJGS model:

The TTCT operations for the creation and editing of tasks τ_a and τ_b , represented by $\mathbf{G}_a$ and $\mathbf{G}_b$ respectively, are reported in Appendix 2. WCET of tasks τ_a , τ_a^l , and τ_b are named TASKA, LTASKA, and TASKB, respectively. Their other parameters are recorded in Table III. In the TDES models, events a_i , e_i , c_i , and $tick$ are represented by $i0$, $i1$, $i3$, and 0 , respectively.

F. TDES RTS Model

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 τ_1 and τ_2 are running in RTS $\mathbb{S}^0$. We generate $\mathbb{S}^0$ by the following TTCT procedures (all the **sync** operations in the original **MAKEIT** file were reported with the message “Blocked_events = None”, eliminated here for readability):

TABLE III: Parameters of MJGS Tasks

task	TDES	type	period	R	C	D	T
τ_a	$\mathbf{G}_a$	M	PA	0	1	[2, 3]	[2, 3]
τ_a^l	$\mathbf{G}_a^l$	F	LPA	0	1	2	2
τ_b	$\mathbf{G}_b$	F	PB	0	2	3	3

SYS0 = sync (TASK1, TASK2) (425, 644)

where “(425, 644)” denotes that $\mathbb{S}^0$, represented by SYS0, has 425 states and 644 transitions. Suppose that another RTS $\mathbb{S}^1$, represented by SYS1, contains τ_1 , τ_2 , and τ_3 . It is generated based on $\mathbb{S}^0$ 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 $\mathbb{S}_l^0$ (LSYS0), $\mathbb{S}_l^1$ (LSYS1), and $\mathbb{S}_u^1$ (USYS1) as follows. They are the counterparts of $\mathbb{S}^0$ and $\mathbb{S}^1$ 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 IV: Parameters of MCW Systems

system	type	TTCT	tasks
$\mathbb{S}^0$	M	SYS0	τ_1, τ_2
$\mathbb{S}^1$	M	SYS1	τ_1, τ_2, τ_3
$\mathbb{S}_l^0$	F	LSYS0	τ_1, τ_2^l
$\mathbb{S}_l^1$	F	LSYS1	$\tau_1, \tau_2^l, \tau_3^l$
$\mathbb{S}_u^1$	F	USYS1	$\tau_1, \tau_2^u, \tau_3^u$

2) MJGS RTS generation:

The MJGS model for an RTS $\mathbb{S}_l^J$ executing τ_a^l and τ_b is represented by a generator LJ that is generated as follows.

LP = sync (LPA, PB) (12, 61)

LJ = sync (SYS, LP) (89, 155)

The MJGS model $\mathbb{S}^J$ for an RTS executing τ_a and τ_b , represented by J, is generated in a similar way, i.e.,

P = sync (PA, PB) (16, 85)

J = sync (SYS, P) (124, 228)

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

$$L(S) = L(S_1) || L(S_2) || \dots || L(S_n)$$

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 **supcon** (see Appendix 1).

Theorem 1: [21] Let $E \subseteq \Sigma^*$ and let $K = \text{supC}(E \cap L_m(G))$. If $K \neq \emptyset$ 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 α_i , no other event α_j with $j \neq i$ can occur to preempt it. Hence, the TDES model of specification S_i for task τ_i is illustrated in Fig. 14, in which α_j and β_j with $j \neq i$ represent events α and β for any other task, respectively. The symbol $*$ represents the other events in Σ . As listed in Appendix 3, the specifications for G_1 , G_2 , and G_3 are created by TTCT. Thereafter, by utilizing **sync**, the general specification S presented in Fig. 15 is generated.

Fig. 14: Specification for a real-time task.

Fig. 15: General specification.

B. Specification for MJGS Model

The initial preemptive specification for a JGS model is shown in Fig. 3 in [6]. In the present paper, it is revised to possess an initial and marker state, which is the specification of the MJGS model. As listed in Appendix 3, the preemptive specification S_b for G_b , represented by PRB, is created by TTCT. S_b is depicted in Fig. 16 with $*$ = $\{t, a_a, b_a\}$.

Fig. 16: MJGS model specification.

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.

Fig. 17: Procedures for real-time scheduling.

Suppose that in every scheduling plan only a subset of tasks executed by an RTS enters the uni-processor for execution. Initially the tasks are running in the periodic version with lower bound $T_{i_{min}}$. Thus, the initial processor utilization of initial system S_0 is $U_{max} = \sum_{i=1}^n C_i / T_{i_{min}}$. In case that S_0 is non-schedulable, i.e., no safe execution sequence can be found by supervisory control, S_0 should be reconfigured dynamically at run-time. All the tasks are replaced by the corresponding multiple-period TDES model with $T_i = [T_{i_{min}}, T_{i_{max}}]$, which is followed by supervisory control again to find all the safe execution sequences (possible reconfiguration scenarios). For any composite task model and the general specification, we find (using Theorem 1) all the safe execution sequences by **supcon**. For any task assigned with multiple-periods, its exact processor utilization lies between $U_{i_{min}}$ and $U_{i_{max}}$. Consequently, the processor utilization of the reconfigured RTS S lies between U_{min} and U_{max} , i.e.,

$$U_{min} \leq U_S \leq U_{max}.$$

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.

2) Dynamic Reconfiguration of $\mathbb{S}_l^1$:

The set of safe execution sequences of $\mathbb{S}_l^1$ (**LSYS1**) found by the procedure **supcon** is empty, i.e.,

LSUPER1 = **supcon** (LSYS1, SPEC) (0, 0)

According to the CW model, $\mathbb{S}_l^1$ is non-schedulable at processor utilization $U_{max} = 1/5 + 2/4 + 2/3 > 1$. Thus, we need to reconfigure the system to be the multiple-period model $\mathbb{S}^1$ (**SYS1**) and utilize SCT again to find the safe execution sequences by

SUPER1 = **supcon** (SYS1, SPEC) (2180, 3681)

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

The system is finally schedulable since **SUPER1** is nonempty. In order to find the scheduling map after the reconfiguration, we need to call **project**. However, TTCT fails to output the result of projecting onto events α_i . The reason is that the dynamic reconfiguration of the periods (event γ_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 $\mathbb{S}^1$:

Step 1:

We choose $\mathbb{S}_u^1$ as a subset of the composite task model of $\mathbb{S}^1$, 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 α_1 , α_2 , and α_3 ; it contains 37 states and 54 transitions, as seen in Fig. 22. The corresponding TTCT operations are given as follows.

USUPER1 = **supcon** (USYS1, SPEC) (417, 574)

PIUSUPER1 = **project** (USUPER1, Image [11, 21, 31]) (37, 54)

Fig. 22: Scheduling map of $\mathbb{S}_u^1$.

Step 2:

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

CSUPER1 = **complement** (SUPER1, []) (2181, 21810)

TEST = **meet** (CSUPER1, USUPER1) (417, 574)

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

Finally, we claim that, after the reconfiguration, the scheduling map of $\mathbb{S}^1$ 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 $\mathbb{S}_u^1$

by Cheddar is illustrated in Fig. 23. It can find only one schedulable sequence. Moreover, no sequence for the multiple-period RTS can be found by EDF scheduling in Cheddar.

Fig. 23: Scheduling map of $\mathbb{S}_u^1$ 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 $\gamma_1 \alpha_1 \gamma_2 t \beta_2 t \alpha_2 t t \beta_2 \gamma_2 \dots$. By projecting out γ_1 , α_1 , and β_1 , we obtain $\gamma_2 t \beta_2 t \alpha_2 t t \beta_2 \gamma_2 \dots$. We have $T_2 = 4$ and $U_2 = 2/4$.

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

1. $\gamma_1 \alpha_1 \gamma_2 t \beta_1 \alpha_2 t t \beta_2 t t \gamma_2 \dots$

2. $\gamma_1 \alpha_1 \gamma_2 t \beta_1 t t \alpha_2 t \gamma_1 t \beta_2 \gamma_2 \alpha_2 t t \beta_2 \alpha_1 t \beta_1 t \gamma_2 \dots$

By projecting out γ_1 , α_1 , and β_1 in Sequence (1.), we obtain $\gamma_2 t \alpha_2 t t \beta_2 t t \gamma_2 \dots$. Obviously, we have $T_2 = 5$. The processor utilization of τ_2 is $2/5$.

By projecting out γ_1 , α_1 , and β_1 in Sequence (2.), we obtain $\gamma_2 t t t \alpha_2 t t \beta_2 \gamma_2 \alpha_2 t t \beta_2 t t \gamma_2 \dots$. Evidently, $T_2 = 5$ and $T_2 = 4$ are in two adjacent periods. In the second period of the execution of τ_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 $\gamma_1 \alpha_1 \gamma_2 t \beta_1$, the controller provides at least two subsequences in Sequences (1.) and (2.) to schedule τ_2 . However, neither the CW scheduling nor the EDF scheduling can provide such scheduling plans.

B. Dynamic Reconfiguration of MJGS model

1) Dynamic Reconfiguration of $\mathbb{S}_l^J$:

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

LJ = **trim** (LJ) (0, 0)

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

J = **trim** (J) (15, 29)

J = **project** (J, Null [11]) (13, 16)

During the reconfiguration, $T_a = 3$ is selected automatically. The processor utilization is $U_{min} = 1/3 + 2/3 = 1$ and thus the RTS is schedulable. According to [6], by assigning higher

680 priorities for task τ_a and τ_b , we obtain the safe execution
 681 sequences shown in Figs. 25 and 26, respectively.

Fig. 24: The full scheduling map.

Fig. 25: Task τ_a with a higher priority.

Fig. 26: Task τ_b with a higher priority.

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

685 $\text{SUPER} = \text{supcon}(\mathbf{J}, \text{PRB})(13, 15)$

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

Fig. 27: Non-preemptive scheduling.

687 2) Comparison with the JGS model:

688 Suppose that the initial period of τ_a is $T_a = 3$. The
 689 system is schedulable based on the technique proposed in
 690 [6]. However, the real time scheduling in [6] is priority-based.
 691 Thereafter, the non-preemptive scheduling can be based only
 692 on the scheduling map shown in Fig. 25 or 26. In that case the
 693 scheduling shown in Fig. 27 can never be found. Obviously,
 694 the real-time scheduling in this paper is more general.

695 VII. DISCUSSION

696 A. Computational Complexity

697 The real-time scheduling and reconfiguration are based on
 698 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 $\mathbf{G}$ and the specification $\mathbf{S}$,
 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 “occa-
 sionally skippable” tasks, such as video reception, telecom-
 munications, 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 manufac-
 turing 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 predefine 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
 by off-line supervisory control.

Both job skipping and the elastic task model need to calcu-
 late 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 V: Comparison with other reconfiguration methods

method	on-line	single sequence	processor demand
job skipping	Y	Y	Y
elastic task	Y	Y	Y
this work	N	N	N

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

- [1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real time environment," *J. Assoc. Comput. Mach.*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
- [2] E. Nasser, and G. Bres, "Hard real-time sporadic task scheduling for fixed priority schedulers", in: *Proc. intern. workshop on responsive syst.*, pp. 44-47, 1991.
- [3] G. Buttazzo, M. Bertogna, and G. Yao, "Limited preemptive scheduling for real-time systems. A survey," *IEEE Trans. Ind. Inform.*, vol. 9, no. 1, pp.3-15, 2013.
- [4] L. Lo Bello, E. Bini, G. Patti, "Priority-driven swapping-based scheduling of aperiodic real-time messages over etherCAT networks," *IEEE Trans. Ind. Inform.*, vol. PP, no. 99, pp.1-11.
- [5] P. C. Y. Chen and W. M. Wonham, "Real-time supervisory control of a processor for non-preemptive execution of periodic tasks," *Real-Time Syst.*, vol. 23, pp. 183-208, 2002.
- [6] V. Janarthanan, P. Gohari, and A. Saffar, "Formalizing real-time scheduling using priority-based supervisory control of discrete-event systems", *IEEE Trans. Autom. Cont.*, vol. 51, no. 6, pp. 1053-1058, 2006.
- [7] G. Buttazzo, G. Lipari, and L. Abeni, "Elastic task model for adaptive rate control," in: *Proc. 19th Real-Time Syst. Symp.*, 1998, pp. 286-295.
- [8] G. Buttazzo and L. Abeni, "Adaptive workload management through elastic scheduling", *Real-Time Syst.*, vol. 23, no. 1-2, pp. 7-24, 2002.
- [9] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic scheduling for flexible workload management", *IEEE Trans. Comput.*, vol. 51, no. 3, pp. 289-302, 2002.
- [10] M. Marinoni and G. Buttazzo, "Elastic DVS management in processors with discrete voltage/frequency modes," *IEEE Trans. Ind. Inform.*, vol. 3, no. 1, pp. 51-62, 2007.
- [11] L. Sha, T. Abdelzaker, K. E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-time systems*, vol. 28, no. 2, pp. 101–155, 2004.
- [12] J. Cecilio and P. Furtado, "Architecture for uniform (re)configuration and processing over embedded sensor and actuator networks," *IEEE Trans. Ind. Inf.*, vol. 10, no. 1, pp. 53-60, 2014.
- [13] M. Garcia-Valls, I. R. López, and L. F. Villar, "iLAND: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems," *IEEE Trans. Ind. Inf.*, vol. 9, no. 1, pp. 228-236, 2013.
- [14] L. Li, S. Li, and S. Zhao, "QoS-aware scheduling of services-oriented internet of things," *IEEE Trans. Ind. Inf.*, vol. 10, no. 2, pp. 1497-1505, 2014.
- [15] M. N. Rooker, C. Sünder, T. Strasser, A. Zoitl, O. Hummer, and G. Ebenhofer, "Zero downtime reconfiguration of distributed automation systems: The ϵ CEDAC approach," in: *Proc. Int. Conf. Indust. Appl. Holonic Multi-Agent Syst.*, Regensburg, Sept. 2007, pp. 326-337.
- [16] P. Vrba and V. Marik, "Capabilities of dynamic reconfiguration of multiagent-based industrial control systems", *IEEE Trans. Syst. Man Cybern., Part A: Syst. Humans*, vol. 40, no. 2, pp. 213-223, 2010.
- [17] G. Koren and D. Shasha, "Skip-over: Algorithms and complexity for overloaded systems that allow skips", *Real-Time Systems Symposium*, in: *Proc. 19th Real-Time Syst. Symp.*, pp. 110-117, 1995.
- [18] L. George, and P. Courbin, "Reconfiguration of uniprocessor sporadic real-time systems: the sensitivity approach," *book chapter in IGI-Global Knowledge on Reconfigurable Embedded Control Systems: Applications for Flexibility and Agility*, Hershey, PA, USA: IGI Global, 2011, pp. 167-189.
- [19] S. Baruah, R. Howell, and L. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor," *Real-Time Syst.*, vol. 2, pp. 301-324, 1990.
- [20] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Contr. Optim.*, vol. 25, no. 1, pp. 206-230, 1987.
- [21] W. M. Wonham, *Supervisory control of discrete-event systems*, Department of Electrical and Computer Engineering, University of Toronto, 2014. Available at <http://www.control.utoronto.ca/DES>.
- [22] B. Brandin and W. M. Wonham, "Supervisory control of timed discrete event systems," *IEEE Trans. Autom. Cont.*, vol. 39, no. 2, pp.329-342, 1994.
- [23] F. Singhoff, J. Legrand, L. Nana, and L. Marce, "Cheddar: A flexible real time scheduling framework," in *Proc. Int. ACM SIGAda Conf.*, pp. 1-8, 2004.
- [24] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no.1, pp. 81-98, 1989.
- [25] W. M. Wonham and P. J. Ramadge, "Modular supervisory control of discrete event systems," *Math. of Cont. Signal Syst.*, vol. 1, no. 1, pp. 13-30, 1988.
- [26] C. Ma and W. M. Wonham, *Nonblocking Supervisory Control of State Tree Structures*, Lecture Notes in Control and Information Sciences (LNCIS) 317, Springer, 2005.
- [27] A. Girbea, C. Cuci, S. Nechifor, and F. Sisak, "Design and implementation of a service-oriented architecture for the optimization of industrial applications," *IEEE Trans. Ind. Inform.*, vol. 10, no. 1, pp. 185-196, 2014.

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 = allevents (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.

878 **TDS2 = edit** (TDS1) is obtained from TDS1 by user-selected
 879 modifications.
 880 **TDS2 = minstate** (TDS1) is a minimal state TDS that
 881 generates the same closed and marked languages as TDS1,
 882 and the same string mapping induced by vocalization (if
 883 any). TDS2 is reachable, but not coreachable unless TDS1
 884 is coreachable.
 885 **TDS2 = project** (TDS1, [NULL/IMAGE EVENTS]) is a
 886 generator of the projected closed and marked languages of
 887 TDS1, under the natural projection specified by the listed Null
 888 or Image events. In decentralized control, TDS2 could be an
 889 observer's local model of TDS1.
 890 **TDS2 = trim** (TDS1) is the trim (reachable and coreachable)
 891 substructure of TDS1.
 892 **TDS3 = meet** (TDS1, TDS2) is the meet (reachable cartesian
 893 product) of TDS1 and TDS2. TDS3 need not be coreachable.
 894 **TDS3 = supcon** (TDS1, TDS2) is a trim generator for
 895 the supremal controllable sublanguage of the marked legal
 896 language generated by TDS2 with respect to the plant TDS1.
 897 TDS3 provides a proper supervisor for TDS1.
 898 **TDS3 = sync** (TDS1, TDS2) is the (reachable) synchronous
 899 product of TDS1 and TDS2.

900 Appendix 2.

901 Corresponding TTCT **MAKEIT** file for all the created
 902 tasks:

903 **TASK1 = create** (TASK1, [mark 0], [tran [0, 10, 1], [1, 0, 5],
 904 [1, 11, 2], [2, 0, 3], [3, 12, 4], [4, 0, 8], [5, 0, 9], [5, 11, 6],
 905 [6, 0, 7], [7, 12, 8], [8, 0, 12], [9, 0, 13], [9, 11, 10], [10, 0,
 906 11], [11, 12, 12], [12, 0, 16], [13, 11, 14], [14, 0, 15], [15,
 907 12, 16], [16, 0, 0], [forcible 10, 11, 12]) (17, 20)
 908 **TASK2 = create** (TASK2, [mark 0, 15, 20], [tran [0, 20, 1],
 909 [1, 0, 6], [1, 21, 2], [2, 0, 3], [3, 0, 4], [4, 22, 5], [5, 0, 10],
 910 [6, 0, 11], [6, 21, 7], [7, 0, 8], [8, 0, 9], [9, 22, 10], [10, 0,
 911 15], [11, 0, 16], [11, 21, 12], [12, 0, 13], [13, 0, 14], [14, 22,
 912 15], [15, 0, 20], [15, 20, 1], [16, 0, 21], [16, 21, 17], [17, 0,
 913 18], [18, 0, 19], [19, 22, 20], [20, 0, 0], [20, 20, 1], [21, 21,
 914 22], [22, 0, 23], [23, 0, 24], [24, 22, 0]], [forcible 20, 21, 22])
 915 (25, 32)
 916 **LTASK2 = edit** (TASK2, [trans -[11, 0, 16], -[15, 0, 20]]) (25,
 917 29)
 918 **LTASK2 = trim** (LTASK2) (16, 18)
 919 **LTASK2 = minstate** (LTASK2) (15, 17)
 920 **UTASK2 = edit** (TASK2, [mark -[15], -[20]], [trans -[15, 20,
 921 1], -[20, 20, 1]]) (25, 29)
 922 **LTASK3 = create** (LTASK3, [mark 0], [tran [0, 30, 1], [1, 0,
 923 6], [1, 31, 2], [2, 0, 3], [3, 0, 4], [4, 32, 5], [5, 0, 0], [6, 31,
 924 7], [7, 0, 8], [8, 0, 9], [9, 32, 0]], [forcible 30, 31, 32]) (10,
 925 11)
 926 **TASK3 = edit** (LTASK3, [mark +[10], +[15]], [trans +[5, 0,
 927 10], +[6, 0, 11], +[9, 32, 10], +[10, 0, 15], +[10, 30, 1], +[11,
 928 0, 16], +[11, 31, 12], +[12, 0, 13], +[13, 0, 14], +[14, 32, 15],
 929 +[15, 0, 0], +[15, 30, 1], +[16, 31, 17], +[17, 0, 18], +[18, 0,
 930 19], +[19, 32, 0], -[5, 0, 0], -[9, 32, 0]]) (20, 25)
 931 **UTASK3 = edit** (TASK3, [mark -[10], -[15]], [trans -[10, 30,
 932 1], -[15, 30, 1]]) (20, 23)
 933 **TASKA = create** (TASKA, [mark 0], [tran [0, 10, 1], [1, 13,
 934 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 = allevents (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)
LPA = create (LPA, [mark 0], [tran [0, 10, 1], [1, 0, 2], [2,
 0, 0]]) (3, 3)
LPA = sync (LPA, 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)

957 Appendix 3.

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

960 **ALLSYS1 = allevents** (LSYS1) (1, 10)
 961 **SPEC1 = create** (SPEC1, [mark 0], [tran [0, 11, 1], [0, 21,
 962 0], [0, 22, 0], [0, 31, 0], [0, 32, 0], [1, 12, 0]], [forcible 11,
 963 12, 21, 22, 31, 32]) (2, 6)
 964 **SPEC1 = sync** (SPEC1, ALLSYS1) (2, 14)
 965 **SPEC2 = create** (SPEC2, [mark 0], [tran [0, 11, 0], [0, 12,
 966 0], [0, 21, 1], [0, 31, 0], [0, 32, 0], [1, 22, 0]], [forcible 11,
 967 12, 21, 22, 31, 32]) (2, 6)
 968 **SPEC3 = create** (SPEC3, [mark 0], [tran [0, 11, 0], [0, 12,
 969 0], [0, 21, 0], [0, 22, 0], [0, 31, 1], [1, 32, 0]], [forcible 11,
 970 12, 21, 22, 31, 32]) (2, 6)
 971 **SPEC = sync** (SPEC1, SPEC2) (3, 18)
 972 **SPEC = sync** (SPEC, SPEC3) (4, 22)
 973 **PRB = create** (PRB, [mark 0], [tran [0, 0, 0], [0, 10, 0], [0,
 974 11, 0], [0, 13, 0], [0, 20, 0], [0, 21, 1], [0, 23, 0], [1, 0, 1],
 975 [1, 10, 1], [1, 20, 1], [1, 21, 1], [1, 23, 0]], [forcible 11, 13,
 976 21, 23]) (2, 12)