KYBERNETIKA — VOLUME *48* (2012), NUMBER 6, PAGES 1114–1135

SUPERVISORY CONTROLLER DESIGN FOR TIMED-PLACE PETRI NETS

Aydin Aybar and Altuğ İftar

Supervisory controller design to avoid deadlock in discrete-event systems modeled by timed-place Petri nets (TPPNs) is considered. The recently introduced approach of place-stretching is utilized for this purpose. In this approach, given an original TPPN (OPN), a new TPPN, called the *place-stretched Petri net* (PSPN), is obtained. The PSPN has the property that its marking vector is sufficient to represent its state. By using this property, a supervisory controller design approach for TPPNs to avoid deadlock is proposed in the present work. An algorithm to determine the set of all the states of the PSPN which lead to deadlock is presented. Using this set, a controller for the PSPN is defined. Using this controller, a controller for the OPN is then obtained. Assuming that the given Petri net is bounded, the proposed approach always finds a controller in finite time whenever there exists one. Furthermore, this controller, when exists, is maximally permissive.

Keywords: discrete-event systems, supervisory control, Petri nets, time-delay, deadlock

Classification: 93C65, 93A30, 93B03

1. INTRODUCTION

Discrete-event systems (DES), whose dynamics can best be described by occurrence of certain events, have been the subject of extensive research for the past few decades [11, 25]. One of the most common modelling methods for DES is Petri nets [17, 19, 21]. Historically, Petri nets were first introduced without the notion of time. However, it was then realized that time may play an important role in some DES. Thus, so-called timed Petri nets (TPNs) were introduced to model such systems [10, 14, 24, 26]. In different models of TPNs, time can be associated with either transitions, places, or arcs [24]. Such TPNs are respectively called timed-transition, timed-place, and timed-arc Petri nets. In this work, we consider timed-place Petri nets (TPPNs). In a TPPN, a *time-delay* (also called *holding duration*) occurs between the time a token enters a place and the time it can enable any downstream transition.

An important property for DES is *deadlock*. Deadlock is said to occur in a discrete-event system modelled by a Petri net if the system reaches to a state at which no event can occur, i. e., no transition of the Petri net can fire. Since it is undesirable to reach to such a state, a supervisory controller is in general needed to avoid this situation. Many works have been devoted to design supervisory controllers to avoid deadlock in systems modelled by Petri nets (e. g., [3, 6, 12, 13, 16, 18, 22, 23]). To design and implement such a supervisory controller, however, the knowledge of the state of the system is, in general, required. For an untimed Petri net, *the marking vector*, which gives the number of tokens in each place, suffices to represent the state of the system. For a TPN, however, marking vector alone is insufficient. For a TPPN, for example, the *time left to the availability* of each token in each place is also required. It is said that a token in a place is *available*, if it can be used to enable any downstream transition. This property makes it very difficult to represent the state of a TPN for any purpose. An approach, called *stretching* (which is later called *transition-stretching*), was first introduced in [4] to overcome this difficulty for timed-transition Petri nets (TTPNs). This approach was then used to design supervisory controllers for TTPNs to avoid deadlock in [5]. A similar approach, called *place-stretching*, was recently introduced in [7] for TPPNs. In this approach, given an original TPPN (OPN), a new TPPN, called the *place-stretched Petri net* (PSPN), is obtained. The PSPN has the property that its marking vector is sufficient to represent its state. By using this property, in the present work we propose a supervisory controller design approach for TPPNs to avoid deadlock. In this approach, given an OPN, its PSPN is obtained first. A controller for the PSPN is then designed. Finally, using the controller for the PSPN, a controller for the OPN is obtained.

Assuming that the given Petri net is bounded (i. e., the number of tokens in each place remains bounded), the proposed approach always finds a controller whenever there exists one. Furthermore, the determined controller is *maximally permissive*, i. e., the reachability set of the controlled Petri net under this controller includes the reachability set of the controlled Petri net under any other controller which also avoids deadlock. It is important to obtain maximally permissive controllers in order to allow maximal operation of DES [15, 20].

It should be noted that a TPPN can be converted into a TTPN and vice versa. Therefore, it is in principle possible to first convert a TPPN into a TTPN and use the approach given in [5] to design a controller. In a Petri net the transitions denote the events and places denote resources [25]. Therefore, it is most natural to associate time with a transition when the corresponding event takes some time to occur and to associate time with a place when some time has to elapse for the availability of the corresponding resource. Once a Petri net model with holding durations on the places (i. e., a TPPN) is obtained, to convert this model into a TTPN it is required to remove the holding durations from all the places and instead associate an equivalent time-delay with each upstream (alternatively downstream) transition. This method would work well if each transition has only one downstream (upstream) place. When this assumption fails, it would be required to repeat certain transitions in order to obtain an equivalent TTPN model. This would however, complicate the model (by increasing the number of transitions) and also would result in a model which does not have a one-to-one correspondence with the actual system (an event in the original system now corresponds to more than one transition). Therefore, rather than converting the original TPPN model into a TTPN model and using the approach of [5], it is easier and more natural to work on the original model and use the approach presented in the present work. It should also be noted that in some DES both the events and the resources may have certain times associated with them. In such a case, it is most natural to model this system with a TPN where both the transitions and places are timed. Then a hybrid stretching approach, where both the transitions and the places are *stretched* can be used. In this case, a hybrid design approach, where the approach of [5] and the approach given in here is used together, can be used to design a controller to avoid deadlock.

2. TIMED-PLACE PETRI NETS

A TPPN is defined as a tuple $G = (P, T_c, T_u, N, O, S_0, \mathcal{D})$. Here, P is the set of *places*, T_c is the set of *controllable transitions*, T_u is the set of *uncontrollable transitions*, $N : P \times T \to \mathcal{N}$ is the *input matrix* which specifies the weights of the arcs directed from places to transitions, where $T := T_c \cup T_u$ is the set of *transitions* and $\mathcal{N}$ is the set of nonnegative integers, $O : P \times T \to \mathcal{N}$ is the *output matrix* which specifies the weights of the arcs directed from transitions to places, $S_0 := S(0)$ is the *initial state*, and $\mathcal{D} : P \to \mathcal{N}_+$ specifies the *holding duration* of a place $p \in P$, where $\mathcal{N}_+$ is the set of positive integers. The difference between a controllable transition $t \in T_c$ and an uncontrollable transition $t \in T_u$ is that a controllable transition can be disabled by a supervisory controller at any time, but an uncontrollable transition can not be disabled.

In this work, it is assumed that the time can be discretized using an appropriate sampling period such that each holding duration is an integer multiple of that period. That is, we assume that all holding durations can be satisfactorily represented by a rational value. Then, the sampling period can simply be chosen as the inverse of the least common denominator of all holding durations (or as the greatest common divisor if all holding durations are integers). Thus, the time variable is represented by an integer k and, without loss of generality, the initial time is let to be $k = 0$. It should be noted that in some cases this approach may necessitate a rather small sampling period, which in turn would result in long holding durations. In our approach of stretching (to be explained below), this would mean that a rather large number of new places and transitions must be introduced to define the stretched Petri net. It is known that the size of the reachability set of a Petri net (thus, also the complexity of computations to analyze it) in general depends exponentially on the number of places and transitions. However, in our case, since each newly introduced transition has only one input place and one output place, the addition of these new places and transitions increase the size of the reachability set of the Petri net only linearly. This is a price to pay in order to obtain an easy representation of the state of the TPN.

The state of the TPPN at time k can be described as [7]:

$$S(k) = \{n(p,k,l),\ l = 0, \ldots, \mathcal{D}(p) - 1,\ p \in P\}, \tag{1}$$

where $n(p,k,l)$ denotes the number of tokens $q \in \eta(p,k)$ for which $\sigma_{p,q}(k) = l$, where $\eta(p,k)$ denotes the set of tokens present in place p at time k, and $\sigma_{p,q}(k)$ denotes the time left to the availability of token q in place p at time k. For any $p \in P$ and $q \in \eta(p,k)$,

$$\sigma_{p,q}(k) = \begin{cases} \mathcal{D}(p) - 1, & \text{if token } q \text{ enters place } p \text{ at time } k \\ & \text{(due to firing of } t \in \bullet p \text{ at time } k-1) \\ \sigma_{p,q}(k-1) - 1, & \text{if token } q \text{ enters place } p \text{ before} \\ & \text{time } k \text{ and } \sigma_{p,q}(k-1) > 0 \\ 0, & \text{otherwise,} \end{cases}$$

where $\bullet p := \{t \in T \mid O(p,t) \neq 0\}$ is the set of transitions which precedes place $p \in P$. Similarly, for future reference, $p\bullet := \{t \in T \mid N(p,t) \neq 0\}$ is the set of transitions which follows place $p \in P$, $\bullet t := \{p \in P \mid N(p,t) \neq 0\}$ is the set of places which precedes transition $t \in T$, and $t\bullet := \{p \in P \mid O(p,t) \neq 0\}$ is the set of places which follows transition $t \in T$. Note that, the number of tokens assigned to place $p \in P$ at any time k, $M(p,k)$, is given by

$$M(p,k) = \sum_{l=0}^{\mathcal{D}(p)-1} n(p,k,l) \; .$$

A transition $t \in T$ is said to be *enabled* at time k if $n(p,k,0) \geq N(p,t)$, $\forall\, p \in P$. Let $E(G,k)$ denote the set of transitions of G which are enabled at time k. Furthermore, for $t \in E(G,k)$, let

$$\epsilon_t(k) := \min_{p \in \bullet t} \{Int\,(n(p,k,0)/N(p,t))\}$$

be the *degree of enabledness* of t at time k, where $Int(\cdot)$ denotes the integer part of $(\cdot)$. Then, $t \in E(G,k)$ may fire upto $\epsilon_t(k)$ times at time k. When a transition $t \in E(G,k)$ fires μ times $(0 \leq \mu \leq \epsilon_t(k))$ at time k, it draws $\mu N(p,t)$ tokens from all places $p \in \bullet t$ and deposits $\mu O(p,t)$ tokens into all places $p \in t\bullet$ at time $k+1$. Note that, this model is appropriate to represent events which can happen more than once at any instant (e. g., a machine which can process one or more parts at a given time). If there is a physical limit on the number of times a transition t can fire (e. g., the number of parts a machine can process) for any duration $\tau > 0$, this limitation can be modelled by introducing a place (call it p_L^t) with $N(p_L^t,t) = O(p_L^t,t) = 1$, $N(p_L^t,\theta) = O(p_L^t,\theta) = 0$, $\forall\,\theta \neq t$, $\mathcal{D}(p_L^t) = \tau$, $M(p_L^t,0) = \lambda$, where λ is the upper limit on the number of firings of the transition t for the duration τ (see [7]). Furthermore, this model also allows firing of more than one transition, say $t_1, \ldots, t_\nu$, respectively $\mu_1, \ldots, \mu_\nu$ times, at the same time k, as long as the set $\{(t_1,\mu_1),\ldots,(t_\nu,\mu_\nu)\}$ is enabled at time k. A set ϕ of pairs (t,μ), where $t \in E(G,k)$ and $0 < \mu \leq \epsilon_t(k)$, is said to be *enabled* at time k if

$$\sum_{(t,\mu)\in\phi} \mu N(p,t) \leq n(p,k,0) \; , \qquad \forall\, p \in P \; .$$

Let $\tilde{E}(G,k)$ denote the set of sets ϕ of G which are enabled at time k. When $\phi \in \tilde{E}(G,k)$ fires at time k, the number of tokens in a place $p \in P$ at the next instant is given by

$$M(p,k+1) = M(p,k) + \sum_{(t,\mu)\in\phi} \mu[O(p,t) - N(p,t)] \; . \tag{2}$$

An important undesirable property for Petri nets is deadlock. Deadlock is said to occur in a Petri net at time k if no transition can fire at time k or later. Thus, deadlock occurs in a TPPN G at time k if and only if

$$E(G,k) = E(G,k+1) = \ldots = E(G,k+d_{\max}-1) = \emptyset \; , \tag{3}$$

where $d_{\max} := \max_{p\in P}\{\mathcal{D}(p)\}$.

3. PLACE-STRETCHED PETRI NETS

Place-stretching was first introduced in [7] for TPPNs in which there is no distinction between controllable and uncontrollable transitions (in [7] control of Petri nets was not considered, hence there was no need for such a distinction). In this section, we generalize this method to TPPNs in which the set of transitions is decomposed into the sets of controllable and uncontrollable transitions. Given an original TPPN (OPN), the method of place-stretching produces a new TPPN, called the *place-stretched Petri net* (PSPN). The PSPN has, in general, more places and transitions than the OPN, but each place of the PSPN has a unit holding duration. This property, as we shall present, makes it easier to represent the state of the TPPN and to design a controller for it. To obtain the PSPN, let us first define $P_1 := \{p \in P \mid \mathcal{D}(p) = 1\}$ and $P_2 := \{p \in P \mid \mathcal{D}(p) \geq 2\}$ (thus, $P = P_1 \cup P_2$). Then, for any $p \in P_2$,

i) define $\mathcal{D}(p) - 1$ new places, p_1^p, p_2^p, …, $p_{\mathcal{D}(p)-1}^p$, and $\mathcal{D}(p) - 1$ new uncontrollable transitions, t_1^p, t_2^p, …, $t_{\mathcal{D}(p)-1}^p$. Let $\pi(p) := \{p_1^p, p_2^p, \ldots, p_{\mathcal{D}(p)-1}^p\}$ and $\tau(p) := \{t_1^p, t_2^p, \ldots, t_{\mathcal{D}(p)-1}^p\}$. The newly introduced uncontrollable transitions are such that any one of them (say t) fires $\epsilon_t(k)$ times at any time k (i. e., these transitions do not wait when they become enabled).

ii) keep all the arcs which enter place p, but detach all the arcs which leave place p from p; instead, attach the originating end of these arcs to place $p_{\mathcal{D}(p)-1}^p$.

iii) define new arcs with unity weights from p to t_1^p, from t_1^p to p_1^p, from p_1^p to t_2^p, …, and from $t_{\mathcal{D}(p)-1}^p$ to $p_{\mathcal{D}(p)-1}^p$.

iv) reassign the holding duration of p as unity; also assign unit holding durations to each of the newly introduced places, p_1^p, p_2^p, …, $p_{\mathcal{D}(p)-1}^p$.

v) to define the initial state of the PSPN, keep the tokens $q \in \eta(p, 0)$ with $\sigma_{p,q}(0) = \mathcal{D}(p) - 1$ inside place p, but move the tokens $q \in \eta(p, 0)$ with $\sigma_{p,q}(0) = l$ to the place $p_{\mathcal{D}(p)-1-l}^p$, for $l = 0, \ldots, \mathcal{D}(p) - 2$.

For example, for the OPN shown in Figure 1, the PSPN shown in Figure 2 is obtained by this procedure.[1]

The PSPN is denoted by the tuple $G_s = (P_s, T_c, T_u, T_n, N_s, O_s, m_0)$. Here, $P_s := P \cup P_n$ is the set of places, where $P_n := \cup_{p \in P_2} \pi(p)$ is the set of newly introduced places. T_c and T_u are the sets of, respectively, controllable and uncontrollable original transitions (which are the same as the corresponding sets for G), and $T_n := \cup_{p \in P_2} \tau(p)$ is the set of newly introduced transitions, which are also uncontrollable. The difference between a $t \in T_u$ and a $t \in T_n$ is that a $t \in T_n$ fires $\epsilon_t(k)$ times at any time k; however, a $t \in T_u$

[1]In all the figures, the places, the transitions, and the arcs are respectively shown by circles, bars, and arrows, as customary. Furthermore, the dots in each circle denotes the tokens present in that place at time $k = 0$. A number in parenthesis next to a place indicates the holding duration of that place, a number next to an arc indicates the weight of that arc, and a number next to a token indicates the time left, $\sigma_{p,q}(0)$, to the availability of that token at time $k = 0$. Unity holding durations, unity weights, and zero times left to availability, however, are not explicitly indicated for clarity. Controllable transitions are indicated by a star next to that transition.

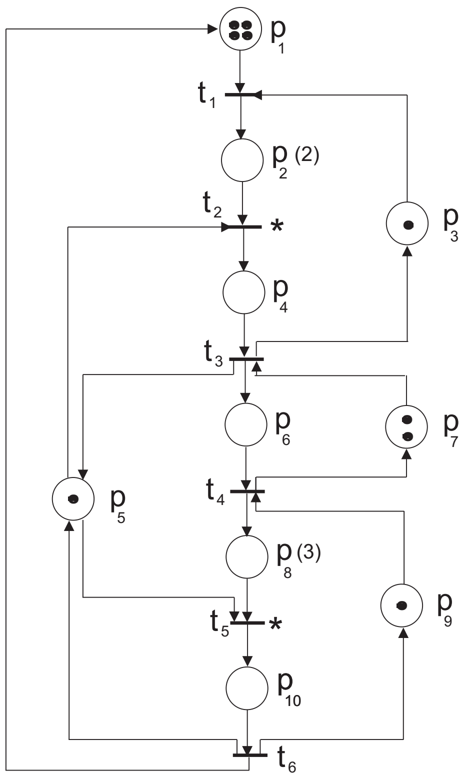

Fig. 1. Example TPPN.

may fire μ times, for any μ between zero and $\epsilon_t(k)$; i. e., a $t \in T_u$ (just like a $t \in T_c$) may wait; however, a $t \in T_n$ does not wait when it becomes enabled. The set of transitions of G_s is given by $T_s := T_c \cup T_u \cup T_n = T \cup T_n$. The input matrix, $N_s : P_s \times T_s \to \mathcal{N}$, is given as

$$N_s(p,t) = \begin{cases} N(p,t), & \text{if } p \in P_1 \text{ and } t \in T \\ 1, & \text{if } p \in P_2 \text{ and } t = t_1^p \\ 1, & \text{if } p = p_i^{\hat{p}} \text{ and } t = t_{i+1}^{\hat{p}} \text{ for some } \hat{p} \in P_2 \\ & \text{with } \mathcal{D}(\hat{p}) \geq 3 \text{ and } i = 1, \ldots, \mathcal{D}(\hat{p}) - 2 \\ N(\hat{p},t), & \text{if } p = p_{\mathcal{D}(\hat{p})-1}^{\hat{p}} \text{ and } t \in T \text{ for some } \hat{p} \in P_2 \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

On the other hand, the output matrix, $O_s : P_s \times T_s \to \mathcal{N}$, is given as

$$O_s(p,t) = \begin{cases} O(p,t), & \text{if } p \in P \text{ and } t \in T \\ 1, & \text{if } p = p_i^{\hat{p}} \text{ and } t = t_i^{\hat{p}} \text{ for some } \hat{p} \in P_2 \\ & \text{and } i = 1, \ldots, \mathcal{D}(\hat{p}) - 1 \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

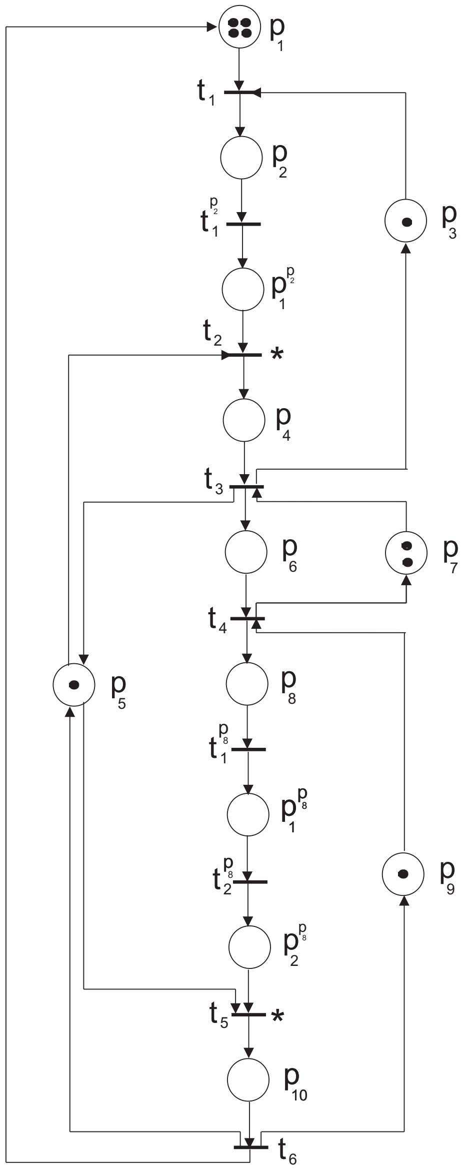

Fig. 2. PSPN for the example TPPN.

Finally, $m_0 : P_s \to \mathcal{N}$ is the initial marking, which is given as

$$m_0(p) = \begin{cases} n(p, 0, \mathcal{D}(p) - 1), & \text{if } p \in P \\ n(\hat{p}, 0, \mathcal{D}(\hat{p}) - 1 - i), & \text{if } p = p^{\hat{p}}_{\mathcal{D}(\hat{p})-i} \text{ for some } \hat{p} \in P_2, \\ & i = 1, \dots, \mathcal{D}(\hat{p}) - 1. \end{cases} \tag{6}$$

Note that, since $d_{\max} = 1$ for the PSPN, the *marking vector*, $m(k) : P_s \to \mathcal{N}$, which gives the number of tokens in each place $p \in P_s$ at time k, completely defines the state of the PSPN at any time k. Therefore, the initial state S_0, in the description of a general TPPN is replaced by the initial marking, $m_0 := m(0)$, for a PSPN. Furthermore, in the definition of a PSPN, there is no need for a function which defines the holding durations of the places, since all the holding durations are unity.

In the PSPN model, a transition $t \in T_s$ is enabled at time k, i. e., $t \in E(G_s, k)$, if $m(p, k) \geq N_s(p, t)$, $\forall p \in P_s$, where $m(p, k)$ is the number of tokens assigned to place $p \in P_s$ at time k. For any $t \in E(G_s, k)$, the degree of enabledness of t at time k is simply given as $\epsilon_t(k) = \min_{p \in \bullet t} \{Int\,(m(p, k)/N_s(p, t))\}$. A set ϕ of pairs (t, μ), where $t \in E(G_s, k)$ and $0 < \mu \leq \epsilon_t(k)$, is enabled at time k, i. e., $\phi \in \tilde{E}(G_s, k)$, if $\sum_{(t,\mu)\in\phi} \mu N_s(p, t) \leq m(p, k)$, $\forall p \in P_s$. When $\phi \in \tilde{E}(G_s, k)$ fires at time k, the marking vector at the next instant is given as

$$m(k + 1) = m(k) + \sum_{(t,\mu)\in\phi} \mu[O_s(t) - N_s(t)]\,, \tag{7}$$

where $N_s(t)$ and $O_s(t)$ respectively denote the columns of N_s and O_s which correspond to transition t.

Note that, as remarked in [7], (7) completely describes the evolution of the state of the PSPN. The corresponding equation, (2), on the other hand, is insufficient to describe the evolution of the state of a general TPPN. To describe the evolution of the state of a general TPPN, one needs to describe the evolution of $n(p, k, l)$, for each $l = 0, \dots \mathcal{D}(p) - 1$, and each $p \in P$. However, once the state (i. e., the marking vector) of the PSPN is determined at any time k, the state of its corresponding TPPN (i. e., the OPN) at time k can easily be obtained as (1), where

$$n(p, k, l) = \begin{cases} m(p, k), & \text{for } l = \mathcal{D}(p) - 1 \\ m(p^p_{\mathcal{D}(p)-1-l}, k), & \text{for } p \in P_2,\ l = 0, \dots, \mathcal{D}(p) - 2, \end{cases} \tag{8}$$

for any $p \in P$ and $l = 0, \dots, \mathcal{D}(p) - 1$. Furthermore, the number of tokens assigned to place $p \in P$ of the OPN at any time k can be obtained as

$$M(p, k) = \begin{cases} m(p, k), & \text{for } p \in P_1 \\ m(p, k) + \sum_{\hat{p}\in\pi(p)} m(\hat{p}, k), & \text{for } p \in P_2. \end{cases} \tag{9}$$

Also note that, due to the fact that $d_{\max} = 1$ for the PSPN, deadlock occurs in the PSPN at time k if and only if $E(G_s, k) = \emptyset$. Furthermore, since there is a one to one correspondence between the state of the PSPN and of its OPN, deadlock occurs in the OPN at time k if and only if deadlock occurs in the PSPN at time k. Therefore, to check for deadlock in the OPN, rather than finding the state of the OPN and using (3), one can simply determine the state of the PSPN (which is easier as explained above), and check whether $E(G_s, k) = \emptyset$.

Therefore, it is in general easier to use the PSPN model to analyze or design a discrete-event system modeled by a TPPN. The state of the TPPN can then be recovered as (1), where $n(p, k, l)$ is given by (8). In fact, many analysis and design algorithms developed for untimed Petri nets can easily be modified and used for a PSPN, although they can not be used directly for a general TPPN. The basic reason for this is that, due to different holding durations of different places, the events which may seem to occur simultaneously in the untimed Petri net, can not in fact occur in the actual TPPN. However, since all the holding durations in the PSPN are unity, this difficulty is resolved. For example, to obtain the *reachability set*, $R(G, S_0)$, i. e., the set of all reachable states from S_0, of a TPPN G, the reachability set, $R(G_s, m_0)$, of its PSPN G_s can first be obtained. The reachability set of the given TPPN can then be obtained using (8) and (1). Algorithm 1, which is taken from [7] (which was obtained by properly modifying the algorithm introduced in [1] for untimed Petri nets), constructs the reachability set of a PSPN.[2] This algorithm requires the PSPN definition, G_s (which includes P_s, T_c, T_u, T_n, N_s, O_s, and m_0), and returns the reachability set, $R(G_s, m_0)$, and the set $\Omega(G_s, m_0)$. The k^{th} element of $\Omega(G_s, m_0)$ is the minimum time required to reach the k^{th} element of $R(G_s, m_0)$ from m_0. This algorithm uses the function *SimSet* which returns the set, $\tilde{E}(G_s, \cdot)$, of enabled sets of pairs (t, μ), at any state $m(\cdot)$. *SimSet*, which is shown as Algorithm 2, requires the PSPN definition G_s (except for m_0) and the marking vector m. Algorithm 1 builds up the reachability set, R, by starting with the initial marking m_0 (which is naturally an element of R). At each iteration, the set Φ of all enabled sets of pairs (t, μ), at each lastly added marking is determined using Algorithm 2. Then, all the markings which are reachable from each of the lastly added markings in one time step are computed. These markings are added to R, unless they are already an element of R. These iterations continue until no new marking is found. Algorithm 2, on the other hand, determines the set Φ, by first determining all the enabled transitions, t, and degree of enabledness, ϵ_t, of each such t. Then, all possible pairs (t, μ) for all enabled t and $\mu = 1, \ldots, \epsilon_t$, are formed. Then, all different combinations of these (t, μ) are collected as an element ϕ of Φ as long as they satisfy the condition $\sum_{(t,\mu)\in\phi} \mu N_s(p, t) \leq m(p, k)$, $\forall\, p \in P_s$. While doing this, special attention is given to $t \in T_n$. Any such t, with its corresponding $\mu = \epsilon_t$, is included in all ϕ, since any newly introduced transition must fire ϵ_t times when it becomes enabled. Both Algorithm 1 and 2 terminate in finite time as long as the OPN is bounded.

4. SUPERVISORY CONTROLLER DESIGN TO AVOID DEADLOCK

In this section, a supervisory controller design approach is introduced to avoid deadlock in TPPNs. Since it is easier to identify deadlock in a PSPN (as explained above), our proposed approach is based on place-stretching. In this approach, given a TPPN (i. e.,

[2]In all algorithms, for a set F, 2^F denotes the power set of F, $|F|$ denotes the number of elements of F and $[F]_i$ denotes the i^{th} element of F ($i = 1, 2, \ldots, |F|$). All the sets are ordered sets. $A \,\hat{\cup}\, B$ is used to denote the set union $A \cup B$, whenever it is known apriori that $A \cap B = \emptyset$. To evaluate $H = A \,\hat{\cup}\, B$ elements of A and of B are simply appended to form H. Furthermore, $\rho(m, \phi) := m + \sum_{(t,\mu)\in\phi} \mu[O_s(t) - N_s(t)]$, $\bar{\rho}(m, t) := m + O_s(t) - N_s(t)$, and $MaX(m) := \max_{p\in P_s}\{m(p)\}$.

Algorithm 1 Algorithm to construct the reachability set of the PSPN.

```
**Input:** $G_s = (P_s, T_c, T_u, T_n, N_s, O_s, m_0)$
**Outputs:** $R = R(G_s, m_0)$ and $\Omega_R = \Omega(G_s, m_0)$

$R = R_1 = \{m_0\}$, $\Omega_R = \Omega_{R_1} = \{0\}$
**loop**
  $R_2 = \emptyset$, $\Omega_{R_2} = \emptyset$
  **for** $i = 1$ to $|R_1|$ **do**
    $m = [R_1]_i$
    $\Phi =$SimSet$[G_s, m]$
    **for** $j = 1$ to $|\Phi|$ **do**
      $\phi = [\Phi]_j$
      $\hat{m} = \rho(m, \phi)$
      **if** $\hat{m} \notin (R_2 \hat{\cup} R)$ **then**
        $R_2 \leftarrow R_2 \hat{\cup} \{\hat{m}\}$
        $\Omega_{R_2} \leftarrow \Omega_{R_2} \hat{\cup} \{[\Omega_{R_1}]_i + 1\}$
      **end if**
    **end for**
  **end for**
  **if** $R_2 == \emptyset$ **then**
    **exit loop**
  **end if**
  $R \leftarrow R \hat{\cup} R_2$
  $\Omega_R \leftarrow \Omega_R \hat{\cup} \Omega_{R_2}$
  $R_1 = R_2$
  $\Omega_{R_1} = \Omega_{R_2}$
**end loop**
**return** $R$, $\Omega_R$
```

the OPN), a PSPN is first obtained as explained in the previous section. A controller is then designed for the PSPN. Based on this controller, a controller is obtained for the OPN in the final phase.

To design the controller for the PSPN, let $R(G_s, m_0)$ denote the reachability set of the PSPN (which can be obtained by using Algorithm 1). For any $m \in R(G_s, m_0)$, let $E_s(m)$ denote the set of all enabled transitions of G_s at marking m, i. e., $E_s(m) := \{t \in T_s \mid N_s(p,t) \leq m(p)\ ,\ \forall p \in P_s\}$. Furthermore, let $\tilde{E}_s(m)$ denote the set of all enabled sets of pairs (t, μ) of G_s at marking m, i. e., $\tilde{E}_s(m) := \{\phi \mid \sum_{(t,\mu)\in\phi} \mu N_s(p,t) \leq m(p)\ ,\ \forall p \in P_s\}$. Then, deadlock occurs in the PSPN if and only if $\tilde{E}_s(m) = \emptyset$ at some $m \in R(G_s, m_0)$. Such an m is called a *deadlock state*. Therefore, in order to avoid deadlock, a controller must make sure that the state of the PSPN does not reach to a deadlock state. Let D_0 denote the set of deadlock states, i. e., $D_0 := \{m \in R(G_s, m_0) \mid \tilde{E}_s(m) = \emptyset\}$. Let $\hat{D}$ denote the set of states that must be avoided in order to avoid deadlock. Clearly $D_o \subset \hat{D}$. Furthermore, note that if the state of the PSPN reaches to a state $m \in R(G_s, m_0) \setminus \hat{D}$ such that firing any $\phi \in \tilde{E}_s(m)$ leads to a state

Algorithm 2 Algorithm to determine the set of enabled sets.

```
Inputs: $G_s = (P_s, T_c, T_u, T_n, N_s, O_s)$ and $m$.
Output: $\tilde{E}_s = \tilde{E}_s(m)$.

FUNCTION SimSet[$G_s, m$]
$T = T_c \cup T_u$, $E_s = \emptyset$, $\Sigma_s = \emptyset$, $E_s^n = \emptyset$, $\tilde{E}_s = \emptyset$, $\bar{E}_s = \emptyset$
for $i = 1$ to $|T|$ do
  $\epsilon = MaX(m)$
  for $j = 1$ to $|P_s|$ do
    if $m([P_s]_j) < N_s([P_s]_j, [T]_i)$ then
      go to Break-1
    end if
    if $N_s([P_s]_j, [T]_i) \neq 0$ then
      $\epsilon = min(\epsilon, Int(m([P_s]_j) / N_s([P_s]_j, [T]_i)))$
    end if
  end for
  $E_s \leftarrow E_s \hat{\cup} \{[T]_i\}$
  $\Sigma_s \leftarrow \Sigma_s \hat{\cup} \{\epsilon\}$
  Break-1: continue
end for
for $i = 1$ to $|E_s|$ do
  for $j = 1$ to $[\Sigma_s]_i$ do
    $\bar{E}_s \leftarrow \bar{E}_s \hat{\cup} \{([E_s]_i, j)\}$
  end for
end for
for $i = 1$ to $|T_n|$ do
  for $j = 1$ to $|P_s|$ do
    if $N_s([P_s]_j, [T_n]_i) == 1$ then
      if $m([P_s]_j) \neq 0$ then
        $E_s^n \leftarrow E_s^n \hat{\cup} \{([T_n]_i, m([P_s]_j))\}$
      end if
      go to Break-2
    end if
  end for
  Break-2: continue
end for
```

(continued on the next page)

Algorithm 2 (continued).

```
$\hat{E}_s = 2^{\bar{E}_s}$
**for** $i = 1$ to $|\hat{E}_s|$ **do**
  $\phi = [\hat{E}_s]_i \hat{\cup} E_s^n$
  **if** $\phi == \emptyset$ **then**
    **go to** JMP
  **end if**
  **for** $l = 1$ to $|P_s|$ **do**
    $cnt = 0$
    **for** $j = 1$ to $|\phi|$ **do**
      $(t, \mu) = [\phi]_j$
      $cnt \leftarrow cnt + \mu N_s([P_s]_l, t)$
    **end for**
    **if** $m([P_s]_l) < cnt$ **then**
      **go to** JMP
    **end if**
  **end for**
  $\tilde{E}_s \leftarrow \tilde{E}_s \hat{\cup} \{\phi\}$
  JMP: **continue**
**end for**
**return** $\tilde{E}_s$
```

in $\hat{D}$ then it is impossible to avoid deadlock. Therefore, a supervisory controller must also avoid such m; i. e., such m must also be in $\hat{D}$. Moreover, if the state of the PSPN reaches to a state $m \in R(G_s, m_0) \setminus \hat{D}$ such that there exists a $\phi \in \tilde{E}_s(m)$ such that $t \in T_{s_u}$, $\forall (t, \mu) \in \phi$, where $T_{s_u} := T_u \cup T_n$ is the set of all uncontrollable transitions of G_s, then a controller can not disable such ϕ. Therefore, if firing this ϕ leads to a state in $\hat{D}$, then such m must also be avoided by the controller. Note that, this latter situation occurs if and only if there exists $t \in E_s(m) \cap T_{s_u}$ such that firing t once leads to a state in $\hat{D}$. In summary, the set of all states that must be avoided in order to avoid deadlock is given as $\hat{D} := \bigcup_{i=0}^{n} D_i$, where n is such that $D_n \neq \emptyset$, but $D_{n+1} = \emptyset$, and, for $i = 1, \ldots, n+1$, $D_i := D_i^c \cup D_i^u$, $D_i^c := \{m \in R(G_s, m_0) \setminus \bigcup_{k=0}^{i-1} D_k \mid \rho(m, \phi) \in \bigcup_{k=0}^{i-1} D_k,\ \forall \phi \in \tilde{E}_s(m)\}$, and $D_i^u := \{m \in R(G_s, m_0) \setminus \bigcup_{k=0}^{i-1} D_k \mid \bar{\rho}(m, t) \in \bigcup_{k=0}^{i-1} D_k$, for some $t \in E_s(m) \cap T_{s_u}\}$, where $\rho(m, \phi)$ denotes the next state when ϕ fires at m, i. e., $\rho(m, \phi) := m + \sum_{(t,\mu)\in\phi} \mu[O_s(t) - N_s(t)]$, and $\bar{\rho}(m, t)$ denotes the next state when t fires once at m, i. e., $\bar{\rho}(m, t) := \rho(m, \{(t, 1)\}) = m + O_s(t) - N_s(t)$. Once $R := R(G_s, m_0)$ is determined (e. g., by Algorithm 1), Algorithm 3 finds $\hat{D}$. Algorithm 3 requires the PSPN definition, G_s, and the reachability set R. Algorithm 3 also uses the function *SimSet*, given as Algorithm 2. To determine $\hat{D}$, Algorithm 3 first determines D_0. If D_0 is empty, then $\hat{D}$ is also empty, and hence the algorithm is terminated. Otherwise, $\hat{D}$ is initiated as D_0 and a loop is entered in which, at its i^{th} iteration, D_i^u, D_i^c, and $D_i := D_i^c \cup D_i^u$ are determined and D_i is appended to $\hat{D}$. The iteration of this loop is terminated when no new element for D_i is found. Algorithm 3 terminates in finite time as long as R is a finite set (which is true when the OPN is bounded).

Once the set $\hat{D}$ is determined, in order to avoid deadlock, at any state m of G_s, a controller must disable firing any $\phi \in \tilde{E}_s(m)$ which leads the state of G_s into $\hat{D}$. Such a controller may be described using the *controller function*:

$$c(m, \phi) := \begin{cases} 0, & \text{if } \rho(m, \phi) \in \hat{D} \\ 1, & \text{otherwise,} \end{cases} \tag{10}$$

for all $\phi \in \tilde{E}_s(m)$ and for all $m \in R(G_s, m_0)$, where $c(m, \phi) = 0$ means that firing of ϕ at state m is disabled by the controller and $c(m, \phi) = 1$ means that firing of ϕ at state m is not disabled by the controller. Note that, by the construction of $\hat{D}$, if $\rho(m, \phi) \in \hat{D}$ for any $m \in R(G_s, m_0) \setminus \hat{D}$ and $\phi \in \tilde{E}_s(m)$, then there exists at least one $(t, \mu) \in \phi$ such that $t \in T_c$. Therefore, the controller (10) can in fact disable firing of ϕ at the state m. This means that, given $m_0 \notin \hat{D}$, the controller (10) can indeed guarantee deadlock-free operation of the PSPN. Note that if $m_0 \in \hat{D}$, then there exists no controller which can guarantee deadlock-free operation of the PSPN. Furthermore, if the initial state S_0 of the OPN is such that $m_0 \in \hat{D}$ (where m_0 is given by (6)), then there exists no controller which can guarantee deadlock-free operation of the OPN.

Algorithm 3 Algorithm to determine the set $\hat{D}$.

```
Inputs: $G_s = (P_s, T_c, T_u, T_n, N_s, O_s)$ and $R = R(G_s, m_0)$.
Output: $\hat{D}$.

$\hat{D} = \emptyset$
for $i = 1$ to $|R|$ do
  $m = [R]_i$
  $\Phi = SimSet[G_s, m]$
  if $\Phi == \emptyset$ then
    $\hat{D} \leftarrow \hat{D} \hat{\cup} \{m\}$
  end if
end for
if $\hat{D} == \emptyset$ then
  go to Fin
end if
$R_1 = R \setminus \hat{D}$
$T_{s_u} = T_u \hat{\cup} T_n$
```

(continued on the next page)

Algorithm 3 (continued).

```
loop
  $D = \emptyset$
  for $i = 1$ to $|R_1|$ do
    $m = [R_1]_i$
    for $j = 1$ to $|T_{s_u}|$ do
      for $r = 1$ to $|P_s|$ do
        if $N_s([P_s]_r, [T_{s_u}]_j) > m([P_s]_r)$ then
          go to Break-1
        end if
      end for
      if $\bar{\rho}(m, [T_{s_u}]_j) \in \hat{D}$ then
        $D \leftarrow D \hat{\cup} \{m\}$
        go to Break-2
      end if
      Break-1: continue
    end for
    $\Phi = SimSet[G_s, m]$
    for $j = 1$ to $|\Phi|$ do
      if $\rho(m, [\Phi]_j) \notin \hat{D}$ then
        go to Break-2
      end if
    end for
    $D \leftarrow D \hat{\cup} \{m\}$
    Break-2: continue
  end for
  if $D = \emptyset$ then
    exit loop
  end if
  $\hat{D} \leftarrow \hat{D} \hat{\cup} D$
  $R_1 \leftarrow R_1 \setminus D$
end loop
Fin: return $\hat{D}$
```

Using controller (10), a controller for the OPN at any state $S \in R(G, S_0)$ can be described as

$$C(S, \phi) = c(m, \hat{\phi}) \tag{11}$$

for all $\phi \in \tilde{E}_o(S)$, where $\tilde{E}_o(S)$ is the set of all enabled sets of pairs (t, μ) of G at S, i. e., $\tilde{E}_o(S) := \{\phi \mid \sum_{(t,\mu)\in\phi} \mu N(p,t) \leq n(p,\cdot,0)\ ,\ \forall p \in P\}$. Here, m is the marking of G_s which corresponds to state S of G, i. e.,

$$m(p) = \begin{cases} n(p,\cdot,\mathcal{D}(p)-1), & \text{if } p \in P \\ n(\hat{p},\cdot,\mathcal{D}(\hat{p})-1-i), & \text{if } p = p^{\hat{p}}_{\mathcal{D}(\hat{p})-i} \text{ for some } \hat{p} \in P_2, \\ & i = 1,\dots,\mathcal{D}(\hat{p})-1, \end{cases}$$

and $\hat{\phi} := \phi \cup \left\{\cup_{t\in E_s(m)\cap T_n} \{(t, m(p_t))\}\right\}$, where p_t is the unique element of $\bullet t$ (note that, by (4), for any $t \in T_n$, there is a unique $p_t \in P_s$ such that $N_s(p_t, t) = 1$ and, for any other $p \in P_s$, $N_s(p, t) = 0$). As for c, $C(S, \phi) = 0$ means that firing of ϕ at state S is disabled by the controller and $C(S, \phi) = 1$ means that firing of ϕ at state S is not disabled by the controller. Since there is a one to one correspondence between the states of the OPN and of the PSPN and that the controller (10) guarantees deadlock-free operation of the PSPN as long as $m_0 \notin \hat{D}$, the controller (11) guarantees deadlock-free operation of the OPN, provided only that the initial state S_0 of the OPN is such that $m_0 \notin \hat{D}$, where m_0 is given by (6). This means that, as long as the OPN is bounded, the presented approach produces a controller (given by (11)) which avoids deadlock, whenever there exists such a controller. Furthermore, note that $\hat{D}$ is the smallest subset of $R(G_s, m_0)$ which must be avoided in order to avoid deadlock in the PSPN. This implies that the controller (10) is maximally permissive for G_s. This in turn implies that the controller (11) is maximally permissive for G.

We note that, although it may be possible to implement the controller (11) by using control places and obtaining a closed-loop Petri net, this is usually not required in practice. This controller can be implemented on the actual plant (which has been modeled by a TPPN) as an *on-line monitoring and control system*, as it is suggested, e. g., in [23]. Such a system would monitor the state of the actual plant and at any state S would disable all ϕ for which $C(S, \phi) = 0$.

Before concluding this section, we remark that the computational complexity of Algorithm 1 is proportional to the size of the reachability set $R = R(G_s, m_0)$. The computational complexity of Algorithm 2 is related polynomially to the size of the Petri net (i. e., the number of transitions and the number of places), which is, in general, much smaller than the size of R. Finally, the computational complexity of Algorithm 3 is proportional to $|R| + \sum_{i=0}^{n} \left|R \setminus \left(\cup_{j=0}^{i} D_j\right)\right| \leq \dfrac{|R|^2 + 3|R|}{2}$, where $|\cdot|$ denotes the size of the set $\cdot$ and n, D_0, …, D_n are as defined in the second paragraph of the present section. Therefore, it is concluded that the overall computational complexity of the design approach is related polynomially to the size of $R = R(G_s, m_0)$, or equivalently to the size of $R(G, S_0)$.

5. EXAMPLE

In this section, as an example, we consider the manufacturing system taken from [25]. This system consists of four pallets, two machines, a shared robot, and a buffer which can store upto two intermediate parts. A part which enters the system is first fixtured on one of the available pallets (no parts can enter if there are no available pallets). A fixtured part then enters the first machine if it is available. After the first machine finishes processing of a part, this part is unloaded by the robot if the robot is available. An unloaded part is then placed into the buffer if there is an empty space in the buffer. At this time both the robot and the first machine become available. An intermediate part in the buffer can enter the second machine when it is available. After the second machine finishes processing of a part, this part is unloaded by the robot, provided that the robot is available. After a part is unloaded from the second machine, it is defixtured from the pallet and leaves the system. At this time the robot, the second machine, and the freed pallet become available.

The Petri net model of the system is shown in Figure 1. In this model, the set of places is $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}\}$, where p_1 represents the available pallets, p_2 represents the processing of a part by the first machine, p_3 denotes the availability of the first machine, p_4 represents unloading an intermediate part from the first machine by the robot, p_5 denotes the robot availability, p_6 represents the intermediate parts stored in the buffer, p_7 denotes the available buffer capacity, p_8 represents the processing of a part by the second machine, p_9 denotes the availability of the second machine, and p_{10} represents unloading a finished part from the second machine by the robot. The set of transitions is $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, where t_1 denotes the entrance of a part into the first machine, t_2 denotes the start of the unloading of the first machine, t_3 denotes the end of the unloading of the first machine and placing the unloaded part into the buffer, t_4 denotes the entrance of a part into the second machine, t_5 denotes the start of the unloading of the second machine, and t_6 denotes the end of the unloading of the second machine and defixturing the finished part from the pallet. It is assumed that the transitions which request the robot are controllable and the other transitions are uncontrollable. Hence, $T_c = \{t_2, t_5\}$ and $T_u = \{t_1, t_3, t_4, t_6\}$. It is further assumed that the processing of a part by the first machine takes two time units, the processing of a part by the second machine takes three time units, and each of the other tasks take one time unit. Hence, $\mathcal{D}(p_2) = 2$, $\mathcal{D}(p_8) = 3$, and $\mathcal{D}(p_1) = \mathcal{D}(p_3) = \mathcal{D}(p_4) = \mathcal{D}(p_5) = \mathcal{D}(p_6) = \mathcal{D}(p_7) = \mathcal{D}(p_9) = \mathcal{D}(p_{10}) = 1$. It is assumed that initially each of the four pallets are available, both machines and the robot are available, and the buffer is empty. Hence, the initial state of the system is

$$S_0 = S(0) = \{4, 0, 0, 1, 0, 1, 0, 2, 0, 0, 0, 1, 0\}$$

where the elements of S are ordered as follows:

$$\begin{aligned} S(\cdot) = \{&n(p_1, \cdot, 0), n(p_2, \cdot, 0), n(p_2, \cdot, 1), n(p_3, \cdot, 0), n(p_4, \cdot, 0) = 0, n(p_5, \cdot, 0),\\ &n(p_6, \cdot, 0), n(p_7, \cdot, 0), n(p_8, \cdot, 0), n(p_8, \cdot, 1), n(p_8, \cdot, 2), n(p_9, \cdot, 0), n(p_{10}, \cdot, 0)\}. \end{aligned} \quad (12)$$

The corresponding PSPN is shown in Figure 2. The set of places for the PSPN is $P_s = P \cup P_n$, where $P_n = \{p_1^{p_2}, p_1^{p_8}, p_2^{p_8}\}$, and the set of transitions is $T_s = T \cup T_n$, where $T_n = \{t_1^{p_2}, t_1^{p_8}, t_2^{p_8}\}$. The initial state (i. e., the initial marking) of the PSPN is $m_0 = m(0) = [4\ 0\ 1\ 0\ 1\ 0\ 2\ 0\ 1\ 0\ 0\ 0\ 0]^T$, where $\cdot^T$ indicates transpose and the elements of the marking vector m are ordered as

$$\begin{aligned} m = [&m(p_1)\ m(p_2)\ m(p_3)\ m(p_4)\ m(p_5)\ m(p_6)\ m(p_7)\ \ldots\\ &\ldots\ m(p_8)\ m(p_9)\ m(p_{10})\ m(p_1^{p_2})\ m(p_1^{p_8})\ m(p_2^{p_8})]^T. \end{aligned} \quad (13)$$

The reachability set, $R = R(G_s, m_0)$, of the PSPN and the corresponding set, $\Omega = \Omega(G_s, m_0)$, of minimum times required to reach each element of $R(G_s, m_0)$ are determined using Algorithm 1. The elements of these sets are respectively shown in the first and the second columns of Table 1. The ordering of the elements of each element m of R is as given in (13). The third column of Table 1 lists the state S of the OPN that corresponds to the marking vector of the PSPN which is given in the first column. That is, the third column lists all the elements of $R(G, S_0)$. The ordering of the elements of each S is as given in (12). The fourth column of Table 1 gives the sets of enabled pairs ϕ of the OPN at each state S, i. e., the elements of $\tilde{E}_o(S)$.

Algorithm 3 determines one deadlock state, which is $[0\ 0\ 0\ 1\ 0\ 2\ 0\ 0\ 0\ 0\ 0\ 0\ 1]^T$, for the PSPN. The corresponding state of the OPN is

$$S_D = \{0,0,0,0,1,0,2,0,0,0,1,0,0\}$$

where the ordering of the elements is as in (12). This deadlock state is indicated by a "$\diamond$" in the last column of Table 1. Algorithm 3 further determines that this state is the sole element of the set $\hat{D}$. Therefore, in order to avoid deadlock, a supervisory controller must disable firing any set ϕ at a state S which leads the state to S_D. This controller, $C(S,\phi)$, is defined in the last column of Table 1 for each $S \in R(G,S_0)$ and each $\phi \in \tilde{E}_o(S)$. It is seen that this controller only disables firing $\{(t_2,1)\}$ at state $\{0,0,1,0,0,1,2,0,0,0,1,0,0\}$. This ensures that the state of the controlled OPN never reaches the deadlock state. Note that, this controller allows reaching to any element of $R(G,S_0)$ other than S_D. Hence, the controller is maximally permissive.

6. CONCLUSION

A supervisory controller design approach to avoid deadlock in DES modeled by TPPNs has been presented. In this approach, given an original TPPN (OPN), its PSPN is first obtained. A supervisory controller to avoid deadlock in the PSPN is then designed. Using this controller, a controller for the OPN is obtained in the final phase. Assuming that the given Petri net is bounded, the proposed approach always finds a controller in finite time whenever there exists one. The determined controller, when exists, guarantees deadlock-free operation of the system and is maximally permissive, i. e., the reachability set of the controlled system under this controller includes the reachability set of the controlled system under any other controller which also avoids deadlock. We note that unboundedness of a Petri net (equivalently of the system which it models) is an undesirable property. Controller design approaches have been developed for both untimed Petri nets [9] and TTPNs [8] to guarantee boundedness. A similar approach to guarantee boundedness in TPPNs can be developed along similar lines by using the place-stretching proposed here. Once such a controller is designed, this controller can be applied first to obtain a bounded Petri net. The controller design approach proposed in the present work can then be applied to avoid deadlock.

As explained in the last paragraph of Section 4, the computational time needed is polynomially related to the size of the reachability set, $R(G,S_0)$. It should, however, be noted that, the size of the reachability set, in general, is related to the size of the Petri net non-polynomially. This is, however, a general drawback of the *forbidden states approach* employed here [23]. An alternative approach would be to employ the *structural approach* [12]. However, structural approach, in general, leads to a non-maximally permissive controller. For large-scale Petri nets, where the burden of constructing the reachability set is too high, decomposition approaches [2] may be used to overcome this burden.

Although we have considered only supervisory controller design to avoid deadlock, the approach of place-stretching can also be used to design controllers for other purposes,

m^T	Ω	S	ϕ	C
[4010102010000]	0	$\{4,0,0,1,0,1,0,2,0,0,0,1,0\}$	$\{(t_1,1)\}$	1
[3100102010000]	1	$\{3,1,0,0,0,1,0,2,0,0,0,1,0\}$		
[3000102010100]	2	$\{3,0,1,0,0,1,0,2,0,0,0,1,0\}$	$\{(t_2,1)\}$	1
[3001002010000]	3	$\{3,0,0,0,1,0,0,2,0,0,0,1,0\}$	$\{(t_3,1)\}$	1
[3010111010000]	4	$\{3,0,0,1,0,1,1,1,0,0,0,1,0\}$	$\{(t_1,1)\}$	1
			$\{(t_4,1)\}$	1
			$\{(t_1,1),(t_4,1)\}$	1
[2100111010000]	5	$\{2,1,0,0,0,1,1,1,0,0,0,1,0\}$	$\{(t_4,1)\}$	1
[3010102100000]	5	$\{3,0,0,1,0,1,0,2,1,0,0,0,0\}$	$\{(t_1,1)\}$	1
[2100102100000]	5	$\{2,1,0,0,0,1,0,2,1,0,0,0,0\}$		
[2000102100100]	6	$\{2,0,1,0,0,1,0,2,1,0,0,0,0\}$	$\{(t_2,1)\}$	1
[2000111010100]	6	$\{2,0,1,0,0,1,1,1,0,0,0,1,0\}$	$\{(t_2,1)\}$	1
			$\{(t_4,1)\}$	1
			$\{(t_2,1),(t_4,1)\}$	1
[2100102000010]	6	$\{2,1,0,0,0,1,0,2,0,1,0,0,0\}$		
[3010102000010]	6	$\{3,0,0,1,0,1,0,2,0,1,0,0,0\}$	$\{(t_1,1)\}$	1
[2000102000110]	6	$\{2,0,1,0,0,1,0,2,0,1,0,0,0\}$	$\{(t_2,1)\}$	1
[2001002000010]	7	$\{2,0,0,0,1,0,0,2,0,1,0,0,0\}$	$\{(t_3,1)\}$	1
[2001011010000]	7	$\{2,0,0,0,1,0,1,1,0,0,0,1,0\}$	$\{(t_3,1)\}$	1
			$\{(t_4,1)\}$	1
			$\{(t_3,1),(t_4,1)\}$	1
[2001002100000]	7	$\{2,0,0,0,1,0,0,2,1,0,0,0,0\}$	$\{(t_3,1)\}$	1
[2000102000101]	7	$\{2,0,1,0,0,1,0,2,0,0,1,0,0\}$	$\{(t_2,1)\}$	1
			$\{(t_5,1)\}$	1
[3010102000001]	7	$\{3,0,0,1,0,1,0,2,0,0,1,0,0\}$	$\{(t_1,1)\}$	1
			$\{(t_5,1)\}$	1
			$\{(t_1,1),(t_5,1)\}$	1

Tab. 1. Results for the Example. (continued on the next page)

m^T	Ω	S	ϕ	C
[2100102000001]	7	$\{2,1,0,0,0,1,0,2,0,0,1,0,0\}$	$\{(t_5,1)\}$	1
[2001002000001]	7	$\{2,0,0,0,1,0,0,2,0,0,1,0,0\}$	$\{(t_3,1)\}$	1
[2010111000001]	8	$\{2,0,0,1,0,1,1,1,0,0,1,0,0\}$	$\{(t_1,1)\}$	1
			$\{(t_5,1)\}$	1
			$\{(t_1,1),(t_5,1)\}$	1
[2010120010000]	8	$\{2,0,0,1,0,1,2,0,0,0,0,1,0\}$	$\{(t_1,1)\}$	1
			$\{(t_4,1)\}$	1
			$\{(t_1,1),(t_4,1)\}$	1
[2010111100000]	8	$\{2,0,0,1,0,1,1,1,1,0,0,0,0\}$	$\{(t_1,1)\}$	1
[2010111000010]	8	$\{2,0,0,1,0,1,1,1,0,1,0,0,0\}$	$\{(t_1,1)\}$	1
[2000002001100]	8	$\{2,0,1,0,0,0,0,2,0,0,0,0,1\}$	$\{(t_6,1)\}$	1
[3010002001000]	8	$\{3,0,0,1,0,0,0,2,0,0,0,0,1\}$	$\{(t_1,1)\}$	1
			$\{(t_6,1)\}$	1
			$\{(t_1,1),(t_6,1)\}$	1
[2100002001000]	8	$\{2,1,0,0,0,0,0,2,0,0,0,0,1\}$	$\{(t_6,1)\}$	1
[1100111000001]	9	$\{1,1,0,0,0,1,1,1,0,0,1,0,0\}$	$\{(t_5,1)\}$	1
[2010011001000]	9	$\{2,0,0,1,0,0,1,1,0,0,0,0,1\}$	$\{(t_1,1)\}$	1
			$\{(t_6,1)\}$	1
			$\{(t_1,1),(t_6,1)\}$	1
[1100011001000]	9	$\{1,1,0,0,0,0,1,1,0,0,0,0,1\}$	$\{(t_6,1)\}$	1
[1100120010000]	9	$\{1,1,0,0,0,1,2,0,0,0,0,1,0\}$	$\{(t_4,1)\}$	1
[1100111100000]	9	$\{1,1,0,0,0,1,1,1,1,0,0,0,0\}$		
[1100111000010]	9	$\{1,1,0,0,0,1,1,1,0,1,0,0,0\}$		
[1000011001100]	10	$\{1,0,1,0,0,0,1,1,0,0,0,0,1\}$	$\{(t_6,1)\}$	1
[1000111000101]	10	$\{1,0,1,0,0,1,1,1,0,0,1,0,0\}$	$\{(t_2,1)\}$	1
			$\{(t_5,1)\}$	1

Results for the Example (continued). (continued on the next page)

m^T	Ω	S	ϕ	C
[1000111100100]	10	$\{1,0,1,0,0,1,1,1,1,0,0,0,0\}$	$\{(t_2,1)\}$	1
[1000120010100]	10	$\{1,0,1,0,0,1,2,0,0,0,0,1,0\}$	$\{(t_2,1)\}$	1
			$\{(t_4,1)\}$	1
			$\{(t_2,1),(t_4,1)\}$	1
[1000111000110]	10	$\{1,0,1,0,0,1,1,1,0,1,0,0,0\}$	$\{(t_2,1)\}$	1
[1001011000001]	11	$\{1,0,0,0,1,0,1,1,0,0,1,0,0\}$	$\{(t_3,1)\}$	1
[1001011000010]	11	$\{1,0,0,0,1,0,1,1,0,1,0,0,0\}$	$\{(t_3,1)\}$	1
[1001020010000]	11	$\{1,0,0,0,1,0,2,0,0,0,0,1,0\}$	$\{(t_4,1)\}$	1
[1001011100000]	11	$\{1,0,0,0,1,0,1,1,1,0,0,0,0\}$	$\{(t_3,1)\}$	1
[1010120000001]	12	$\{1,0,0,1,0,1,2,0,0,0,1,0,0\}$	$\{(t_1,1)\}$	1
			$\{(t_5,1)\}$	1
			$\{(t_1,1),(t_5,1)\}$	1
[1010120000010]	12	$\{1,0,0,1,0,1,2,0,0,1,0,0,0\}$	$\{(t_1,1)\}$	1
[0100120000001]	13	$\{0,1,0,0,0,1,2,0,0,0,1,0,0\}$	$\{(t_5,1)\}$	1
[1010020001000]	13	$\{1,0,0,1,0,0,2,0,0,0,0,0,1\}$	$\{(t_1,1)\}$	1
			$\{(t_6,1)\}$	1
			$\{(t_1,1),(t_6,1)\}$	1
[0100020001000]	13	$\{0,1,0,0,0,0,2,0,0,0,0,0,1\}$	$\{(t_6,1)\}$	1
[0000020001100]	14	$\{0,0,1,0,0,0,2,0,0,0,0,0,1\}$	$\{(t_6,1)\}$	1
[0000120000101]	14	$\{0,0,1,0,0,1,2,0,0,0,1,0,0\}$	$\{(t_2,1)\}$	0
			$\{(t_5,1)\}$	1
[0001020000001]	15	$\{0,0,0,0,1,0,2,0,0,0,1,0,0\}$		$\diamond$

Results for the Example (continued).

such as to enforce liveness, boundedness, and/or reversibility [8] in TPPNs. Further research can also be undertaken to develop stretching-like approaches for timed-arc Petri nets or for mixed type Petri nets, where any combination of places, transitions, and arcs are timed.

(Received May 12, 2011)

REFERENCES

[1] H. Apaydın, A. Manay, A. Aybar, and A. İftar: A program for analysis and control of Petri nets. In: Proc. IEEE International Conference on Computational Cybernetics, Vienna 2004, pp. 309–314.

[2] A. Aybar and A. İftar: Overlapping decompositions and expansions of Petri nets. IEEE Trans. Automat. Control *47* (2002), 511–515.

[3] A. Aybar and A. İftar: Decentralized supervisory controller design to avoid deadlock in Petri nets. Internat. J. Control *76* (2003), 1285–1295. Also see: A. Aybar and A. İftar: Corrections to decentralized supervisory controller design to avoid deadlock in Petri nets. Internat. J. Control *76* (2003), 1584.

[4] A. Aybar and A. İftar: Supervisory controller design for timed Petri nets. In: Proc. IEEE International Conference on System of Systems Engineering, Los Angeles 2006, pp. 59–64.

[5] A. Aybar and A. İftar: Deadlock avoidance controller design for timed Petri nets using stretching. IEEE Systems J. *2* (2008), 178–188.

[6] A. Aybar and A. İftar: Decentralized structural controller design for large-scale discrete-event systems modelled by Petri nets. Kybernetika *45* (2009), 3–14.

[7] A. Aybar and A. İftar: Representation of the state of timed-place Petri nets using stretching. In: Proc. 4th IFAC Workshop on Discrete-Event System Design, Playa de Gandia 2009, pp. 79–84.

[8] A. Aybar and A. İftar: Supervisory controller design to enforce some basic properties in timed-transition Petri nets using stretching. Nonlinear Analysis: Hybrid Systems *6* (2012), 712–729.

[9] A. Aybar, A. İftar, and H. Apaydın-Özkan: Centralized and decentralized supervisory controller design to enforce boundedness, liveness, and reversibility in Petri nets. Internat. J. Control *78* (2005), 537–553.

[10] F. D. J. Bowden: A brief survey and synthesis of the roles of time in Petri nets. Math. Comput. Modelling *31* (2000), 55–68.

[11] C. G. Cassandras and S. Lafortune: Introduction to Discrete Event Systems. Kluwer Academic, Norwell 1999.

[12] M. P. Fanti, B. Maione, and B. Turchiano: Comparing digraph and Petri net approaches to deadlock avoidance in FMS. IEEE Trans. Systems, Man Cybernet. – Part B, *30* (2000), 783–798.

[13] M. P. Fanti and M. Zhou: Deadlock control methods in automated manufacturing systems. IEEE Trans. Systems, Man, Cybernet. – Part A *34* (2004), 5–22.

[14] P. Freedman: Time, Petri nets, and robotics. IEEE Trans. Robotics Automat. *7* (1991), 417–433.

[15] A. Ghaffari, N. Rezg, and X. Xie: Maximally permissive and non blocking control of Petri nets using theory of regions. In: Proc. IEEE International Conference on Robotics and Automation, Washington, D. C. 2002, pp. 1895–1900.

[16] A. Giua, C. Seatzu, and F. Basile: Observer-based state-feedback control of timed Petri nets with deadlock recovery. IEEE Trans. Automat. Control *49* (2004), 17–29.

[17] C. N. Hadjicostis and G. C. Verghese: Structured redundancy for fault tolerance in state-space models and Petri nets. Kybernetika *35* (1999), 39–55.

[18] Z. W. Li, M. C. Zhou, and N. Q. Wu: A survey and comparison of Petri net-based deadlock prevention policies for flexible manufacturing systems. IEEE Trans. Systems, Man, Cybernet. – Part C *38* (2008), 173–188.

[19] T. Murata: Petri nets: Properties, analysis and applications. Proc. IEEE *77* (1989), 541–580.

[20] S. Pinchinat and S. Riedweg: You can always compute maximally permissive controllers under partial observation when they exist. In: Proc. American Control Conference, Portland 2005, pp. 2287–2292.

[21] I. Rivera-Rangel, A. Ramirez-Trevino, L. I. Aguirre-Salas, and J. Ruiz Leon: Geometrical characterization of observability in interpreted Petri nets. Kybernetika *41* (2005), 553–574.

[22] M. Uzam and M. Zhou: An iterative synthesis approach to Petri net-based deadlock prevention policy for flexible manufacturing systems. IEEE Trans. Systems, Man, Cybernet. – Part A *37* (2007), 362–371.

[23] N. Viswanadham, Y. Narahari, and T. L. Johnson: Deadlock prevention and deadlock avoidance in flexible manufacturing systems using Petri net models. IEEE Trans. Robotics Automat. *6* (1990), 713–723.

[24] J. Wang: Timed Petri Nets: Theory and Application. Kluwer Academic, Boston 1998.

[25] M. Zhou and F. DiCesare: Petri Net Synthesis for Discrete Event Control of Manufacturing Systems. Kluwer Academic Publishers, Norwell 1993.

[26] W. M. Zuberek: Timed Petri nets in modeling and analysis of cluster tools. IEEE Trans. Robotics Automat. *17* (2001), 562–575.

Aydın Aybar, Dept. of Electrical and Electronics Engineering, Anadolu University, 26470 Eskişehir. Turkey.
e-mail: aaybar@anadolu.edu.tr

Altuğ İftar, Dept. of Electrical and Electronics Engineering, Anadolu University, 26470 Eskişehir. Turkey.
e-mail: aiftar@anadolu.edu.tr