Dynamic Checking and Solution to Temporal Violations in Concurrent Workflow Processes

YanHua Du, PengCheng Xiong, YuShun Fan, and Xitong Li

***Abstract*—Current methods that deal with concurrent workflow temporal violations only focus on checking whether there are any temporal violations. They are not able to point out the path where the temporal violation happens and thus cannot provide specific solutions. This paper presents an approach based on a sprouting graph to find out the temporal violation paths in concurrent workflow processes as well as possible solutions to resolve the temporal violations. First, we model concurrent workflow processes with time workflow net and a sprouting graph. Second, we update the sprouting graph at the checking point. Finally, we find out the temporal violation paths and provide solutions. We apply the approach in a real business scenario to illustrate its advantages: 1) It can dynamically check temporal constraints of multiple concurrent workflow processes with resource constraints; 2) it can give the path information in the workflow processes where the temporal violation happens; and 3) it can provide solution to the temporal violation based on the analysis.**

***Index Terms*—Dynamic checking and solution, Petri nets (PNs), sprouting graph, temporal constraint, time workflow net (TWF-net).**

I. Introduction

WORKFLOW management is a key technology for the collaboration of various business processes, e.g., loan approval and customer order processing [1]–[4]. By constructing process models and enacting them in the workflow server, the workflow management system (WfMS) can help to streamline business processes, to deliver tasks and documents among users, and to monitor the overall performance of the processes. WfMS designers often set temporal constraints when they define workflow processes [5]–[9]. Therefore, it is important to check if all temporal constraints are consistent. Consider a real scenario for manufacturing a new "Lenovo" cell phone (see detail in Section VI). It consists of two concurrent processes: manufacturing an electronic main board and manufacturing peripheral parts. A temporal constraint is given as "manufacturing an electronic main board ends no later than 16 time units after writing a technical document of peripheral parts begins."

Checking temporal constraints can be divided into static checking and dynamic checking. Some previous works [5]–[9] focus on static analysis technique. The main drawback of this technique is that it is hard to comprehensively consider all the potential temporal violations during the run time [3], [4]. By reconsidering the previous scenario, it is hard to use static analysis technique to check if the temporal constraint "manufacturing an electronic main board ends no later than 16 time units after writing a technical document of peripheral parts begins" is satisfied at some time point (e.g., the fifth time unit) at run time because the execution time of some activities is nondeterministic.

Dynamic checking differs from static checking by taking into consideration the real execution time of the activities during run time. According to the number of workflow processes, dynamic checking can be divided into single-process checking and multiple-process checking. There are several methods for dynamic checking of temporal constraints in a single process [10]–[14]. These methods are usually based on the assumption that there is only one single workflow process and there is no shared resource. In practice, however, multiple workflow processes may execute concurrently under shared resource constraints in a WfMS. For example, Li and Yang [15] propose a method based on analyzing the temporal relationship and resource constraints among activities in multiple workflow processes. Although their method can give answer to whether there is any temporal violation, they can neither provide the exact information about the path violation nor a feasible solution.

However, the information about path violation is very important. Reconsidering the previous scenario, suppose that the temporal constraint "manufacturing an electronic main board ends no later than 16 time units after writing a technical document of peripheral parts begins" is violated at the fifth time unit. If the information about path violation is provided, there will be at least three aspects of benefit.

1) This information is helpful for workflow exception handling. With the information, the designer can focus on the activities in the path with violation. The information also gives valuable hint for the designer to correctly place exception handling.
2) This information can facilitate workflow model improvement. With the exact path information, the manager can

Manuscript received December 28, 2009; revised July 31, 2010; accepted November 2, 2010. Date of publication March 14, 2011; date of current version October 19, 2011. This work was supported by the National Natural Science Foundation of China under Grant 61004109. This paper was recommended by Associate Editor M. P. Fanti.

Y. H. Du is with the School of Mechanical Engineering, University of Science and Technology Beijing, Beijing 100083, China (e-mail: duyanhua@ustb.edu.cn).

P. C. Xiong is with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: xiong@gatech.edu).

Y. S. Fan is with the Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: fanyus@tsinghua.edu.cn).

X. Li is with the MIT Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02142 USA (e-mail: xitongli@mit.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMCA.2011.2116003

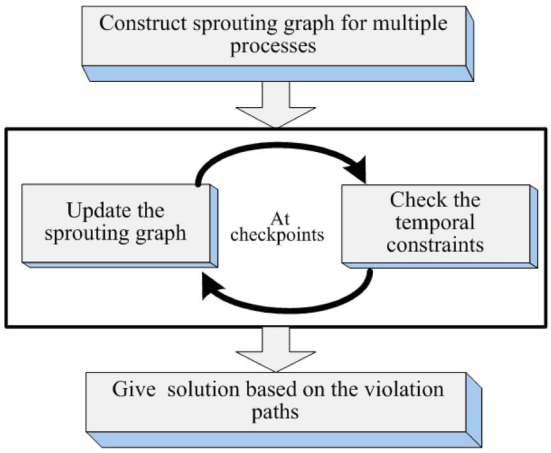

Fig. 1. Our approach skeleton.

try to correct the execution of the activities in the path, e.g., to shorten the execution time of some activities to solve the violation.

3) This information can assist workflow execution analysis and diagnostic. With this information, we can make a statistic analysis of the violation paths. The more frequently the activity is contained in the violation paths, the more possibility the error is within that activity.

As shown in Fig. 1, this paper proposes a novel approach to identify the violation path and possible solutions. First, we construct sprouting graph models for multiple workflow processes. The reason to adopt a sprouting graph lies in that it cannot only depict the real-time process execution but also offer rich analysis capability to support the temporal verification. Second, we update the sprouting graph at different checking points and check the temporal constraints. Finally, we give the violation paths and solutions.

Compared with the existing works, the contributions of this paper are as follows.

1) Our approach can dynamically check the temporal violations of multiple concurrent workflow processes with resource constraints and find out the violation paths.
2) If some temporal constraint is violated, we provide solutions to modify the duration of some activities so that the violations are addressed.

The rest of this paper is organized as follows. The next section introduces the preliminaries, including temporal constraints and formal problem statement. Section III presents the concept of sprouting graph and its constructing procedure. In Section IV, we show how we check temporal constraints based on a sprouting graph. Section V gives solutions to temporal violations by modifying durations of activities. Section VI illustrates an application of our approach by presenting a real example in a manufacturing enterprise, and our approach is validated by using a popular model checking software. Section VII gives an overview of the related work, including static and dynamic checking techniques of temporal constraints. Finally, Section VIII concludes this paper.

II. Preliminaries

A. Temporal constraint

A temporal constraint is a statement which denotes when an activity should start or finish in terms of absolute time or relative time. Here, time is expressed in basic time unit types, e.g., minutes, hours, or days. The time unit type is selected according to the specific workflow applications. Temporal constraints are classified into two types [15], i.e., absolute temporal constraints and relative temporal constraints. An absolute temporal constraint defines the absolute time when an activity should start or complete during workflow execution, e.g., the deadline for the submission of an application is December 15th. A relative temporal constraint defines when an activity should start or end relative to the starting or ending time of another activity, e.g., activity a_j should end no later than s time units after activity a_i starts.

A temporal violation occurs if a temporal constraint cannot be satisfied. For example, in the real scenario for manufacturing a new "Lenovo" cell phone, a temporal violation occurs at the fifth time unit if the temporal constraint "manufacturing an electronic main board ends no later than 16 time units after writing a technical document of peripheral parts begins" cannot be satisfied at the fifth time unit due to delayed execution of some activities.

Here, we use $S(a_i) \leq_t time_i$ to denote that activity a_i should start at or before absolute time $time_i$, and $E(a_j) \leq_t time_j$ to represent that activity a_j should end at or before absolute time $time_j$. $D_R(a_i, a_j) = E(a_j) - S(a_i)$ is called run-time duration from the starting time of a_i to the ending time of a_j. For the relative temporal constraints, we use $D_R(a_i, a_j) \leq s$ to denote that a_j should end its execution no later than s time units after a_i starts. Here, a_i is called a reference activity.

In this paper, we use relative temporal constraints and take $D_R(a_i, a_j) \leq s$ (a_i is a reference point) as the *canonical representation* for temporal constraints [15].

B. TWF-Net

Definition 1 (PNs) [17]–[20]: A Petri net [$PN = (P, T, F)$, where P is a set of places, T is a set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$] is a set of directed arcs linking places and transitions.

We usually use a bar to represent a transition, a circle to represent a place, and a dot to represent a token. Postset of transition t is the set of output places of t, denoted by $t^\bullet$. Preset of transition t is the set of input places of t, denoted by ${}^\bullet t$. Postset (preset) of p is the set of output (input) transitions of p, denoted by $p^\bullet$ and ${}^\bullet p$, respectively.

$M : P \to Z$ is a marking where $M(p)$ represents the number of tokens in place p and $Z = \{0, 1, 2, \ldots\}$. An initial marking is denoted by M_0. p is marked by M iff $M(p) > 0$. A transition $t \in T$ is enabled under M, if and only if $\forall p \in^\bullet t$: $M(p) > 0$, denoted as $M[t>$. If $M[t>$ holds, t may fire, resulting in a new marking M', denoted as $M[t > M'$, such that $M'(p) = M(p) - 1$ if $\forall p \in^\bullet t \setminus t^\bullet$, $M'(p) = M(p) + 1$ if $\forall p \in t^\bullet \setminus^\bullet t$, and otherwise $M'(p) = M(p)$.

Assume that there are m places and n transitions in a PN. Backward incidence matrix F_1 is an $m \times n$ matrix, $F_1(p_i, t_j) = 1$, if $p_i \in {}^\bullet t_j$; $F_1(p_i, t_j) = 0$, otherwise. Forward incidence matrix F_2 is an $m \times n$ matrix, $F_2(p_i, t_j) = 1$, if $p_i \in t_j^\bullet$; $F_2(p_i, t_j) = 0$, otherwise.

Definition 2 (WF-Net): A PN is called a workflow net (WF-net) if and only if the following conditions are true.

1) PN has two special places: ε and θ. Place ε is a source, and ${}^\bullet\varepsilon = \varnothing$. θ is a sink, and $\theta^\bullet = \varnothing$.
2) If we add a new transition to PN which connects place θ with ε, i.e., ${}^\bullet t = \{\theta\}$ and $t^\bullet = \{\varepsilon\}$, then the resulting PN is strongly connected.

A WF-net gives only process control specification of a workflow model. To realize its time dimension verification and analysis, its temporal behavior should be specified, and some extensions are needed. Different works [21]–[26] introduce time into PN-based workflow models. Based on the semantics of time PN, time WF-net (TWF-net) [21]–[23] is proposed by treating a timing constraint as a delay pair consisting of its lower and upper bounds. The following definitions and notations come from [21]–[23].

Definition 3 (TWF-Net): A TWF-net is a three tuple (WF-net, FI, M), where

WF-net $= (P, T, F)$	workflow net;
$P = \{p_1, p_2, \ldots, p_m\}$	set of places representing the state of a transaction instance or the condition of its output transitions;
$T = \{t_1, t_2, \ldots, t_n\}$	set of transitions representing activities of the workflow;
F	set of directed arcs linking places and transitions, and used to describe precedence relations among activities;
FI	set of nonnegative real number pairs $[l, u]$ related to each transition, which is used to represent the minimum firing time and the maximum firing time, respectively;
M	vector of m-dimensional markings, where $M(p)$ denotes the number of tokens representing the number of transaction instances in p.

There are usually two types of transitions in TWF-net, i.e., activity and routing transitions. The former ones model the activity nodes in a workflow. The latter ones determine the control flow among former ones, e.g., and-split, and-join, or-split, and or-join. Since routing transitions fire as soon as they are enabled, they are associated with a time interval [0, 0]. For simplicity, we omit the time interval tag of routing transitions.

In this paper, TWF-net is based on the weak semantics [21] of time PN, in which the firing of transitions can be freely chosen by decisions local to them, and independently from the conflicts with other ones. Assume transition $t \in T$ is enabled in a state M and is associated with a time interval $[l, u]$, $(0 \le l \le u)$. Then, let s and $\tau(t)$ denote the enabled time and the actual firing time of t, respectively. We have $s + l \le \tau(t) \le s + u$. An example of a workflow process A modeled using a TWF-net is shown in Fig. 2, in which we have $2 \le \tau(t_{11}) \le 3$ since its enabled time $s = 0$.

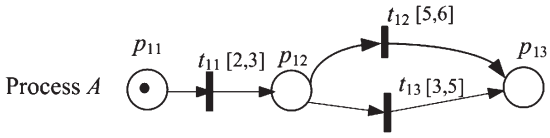

Fig. 2. Workflow process A based on TWF-net.

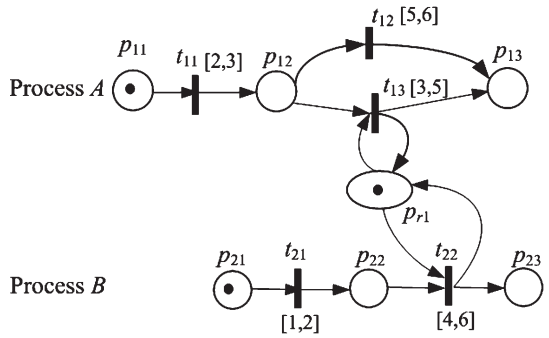

Fig. 3. Two processes based on a TWF-net with resource constraints.

When activities are executed, they use two kinds of resources, i.e., shared and private ones. Shared resources can be accessed by different activities, while private resources are reserved for only one activity. Shared resources will cause conflicts, while private ones will not. In this paper, we assume that shared resources have exclusive property, i.e., the resources can be used by only one activity without preemption. We assume that the first-come first-served (FCFS) policy is applied to allocate resources to activities [15], i.e., an activity is allocated with the resource first if the transition representing that activity is enabled first. An activity will be postponed for execution until all required resources are available. When the activity finishes, it will release all the resources.

However, several transitions in multiple processes may be enabled at the same time, and this case cannot be handled by the FCFS policy. Hence, the other scheduling policies for resource allocation are needed as the complementary policies, e.g., random (RANDOM), short processing time (SPT), and longest processing time (LPT) [27]. In order to simplify our approach, the SPT policy is used when several transitions in multiple processes are enabled at the same time.

If two transitions t_i and t_j share a resource, then there is a resource constraint between t_i and t_j. Here, we denote the set of transitions having resource constraints with t_i as $RC(t_i)$. We use a resource place p_r to denote the shared resource and add corresponding directed arcs to/from the transitions to denote the request and release of the resource. $M_0(p_r) = 1$ means that there is one resource instance at the initial marking.

An example of two workflow processes A and B based on a TWF-net is shown in Fig. 3, in which we assume that t_{13} and t_{22} share a resource p_{r1} which is denoted by an oval.

C. Problem Statement

Given a TWF-net W which contains multiple processes $\{W_1, W_2, \ldots, W_w\}$ and a temporal constraint $D_R(t_i, t_j) \le s$, where transition t_i and t_j may belong to different processes in

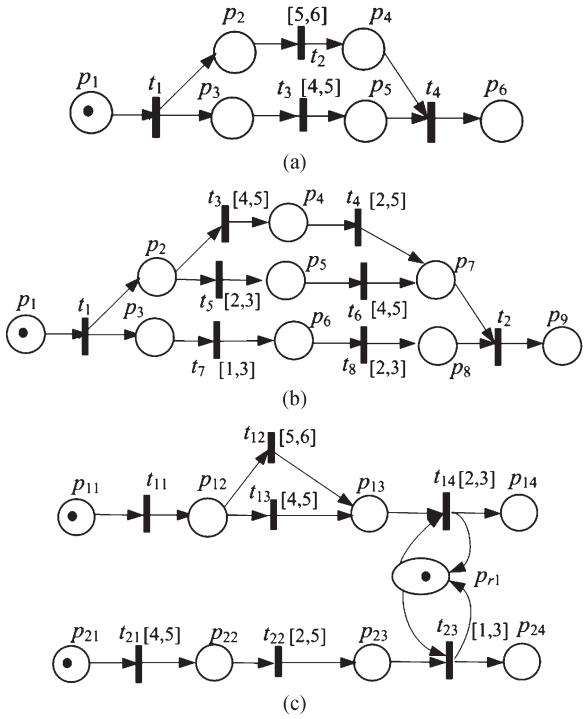

Fig. 4. Three TWF-nets.

W, check if the difference between the firing time of t_j (the time when t_j ends) and the enabled time of t_i (the time when t_i starts) is less than s. If a temporal constraint is violated, find the paths with violations in the workflow processes. After the violation path is found, a solution needs to be provided, which may, for example, modify the execution durations of some of the activities on the paths.

III. TWF-Net and Sprouting Graph

In this section, we first give the definitions and operations of paths and time intervals of TWF-nets and a sprouting graph [16]. Then, we present the procedure to construct a sprouting graph.

A. Operations of Paths and Time Intervals in a TWF-Net

Definition 4 (Path): In a TWF-net, a path is defined as a sequence of transitions $\{t_1, t_2, \ldots, t_k\}$ such that $\forall i, 1 \leq i \leq k-1$, $\exists p \in P$, arc (t_i, p) and (p, t_{i+1}) exist.

For example, $\{t_1, t_2, t_4\}$ and $\{t_1, t_3, t_4\}$ in Fig. 4(a) are paths. The path defined here is similar to the transition path in [25].

Following [16], some operations on paths and path sets are defined as follows. They will be used to simplify the representation of computing and recording path information for transitions in a TWF-net when constructing and updating a sprouting graph.

Definition 5 (Union Operation of Two Paths): Supposing that $path_1$ and $path_2$ are two paths, where $path_1 = \{t_g, t_{g+1}, \ldots, t_h\}$ and $path_2 = \{t_i, t_{i+1}, \ldots, t_j\}$, the union of two paths is expressed as $path_1 \Delta path_2 = \{(t_g, t_{g+1}, \ldots, t_h) \| (t_i, t_{i+1}, \ldots, t_j)\}$, where $\|$ means the union operator of sets.

This operation is used to describe the union of two paths before and-join. For example, there are two paths in Fig. 4(a), i.e., $path_x = \{t_1, t_2\}$ and $path_y = \{t_1, t_3\}$; then, we have $path_x \Delta path_y = \{(t_1, t_2) \| (t_1, t_3)\}$.

Definition 6 (Intercross Operation of Two Paths): Supposing that $path_1$ and $path_2$ are two paths, where $path_1 = \{t_g, t_{g+1}, \ldots, t_h\}$ and $path_2 = \{t_i, t_{i+1}, \ldots, t_j\}$, the intercross of two paths is expressed as $path_1 / path_2 = \{t_g, t_{g+1}, \ldots, t_h (t_i, t_{i+1}, \ldots, t_j)\}$.

This operation is used to describe a resource constraint after t_h and t_j fire. For example, there are two paths in Fig. 3, i.e., $path_x = \{t_{11}\}$ and $path_y = \{t_{21}\}$. After t_{11} and t_{21} fire, we will have a resource conflict. Then, we use $path_x / path_y = t_{11}(t_{21})$ to denote that there is a resource constraint after t_{11} and t_{21} fire.

Definition 7 (Adding a Transition Into a Path): Letting t be a transition, adding t into the path $\{t_g, t_{g+1}, \ldots, t_h\}$ is expressed as $path \oplus t = \{t_g, t_{g+1}, \ldots, t_h, t\}$.

For example, in Fig. 4(a), there exists $path_x = \{t_1, t_2\}$; then, we have $path_x \oplus t_4 = \{t_1, t_2, t_4\}$.

Let PS_1 and PS_2 be two sets of paths, where $PS_1 = \{path_g, path_{g+1}, \ldots, path_h\}$ and $PS_2 = \{path_i, path_{i+1}, \ldots, path_j\}$. The merge, Cartesian union, and Cartesian intercross operations of paths are defined as follows.

Definition 8 (Merge Operation): The merge operation between PS_1 and PS_2 is defined by $PS_1 \odot PS_2 = \{path_g, path_{g+1}, \ldots, path_h, path_i, path_{i+1}, \ldots, path_j\}$. Adding a transition t into PS_1 is defined as $PS_1 \Delta t = \{path_g \oplus t, path_{g+1} \oplus t, \ldots, path_h \oplus t\}$.

For example, in Fig. 4(b), there are two paths from p_1 to p_7, i.e., $\{t_1, t_3, t_4\}$ and $\{t_1, t_5, t_6\}$. We have $PS_a = \{\{t_1, t_3, t_4\}\}$ and $PS_b = \{\{t_1, t_5, t_6\}\}$. Then, $PS_a \odot PS_b = \{\{t_1, t_3, t_4\}, \{t_1, t_5, t_6\}\}$, and $PS_a \Delta t_2 = \{\{t_1, t_3, t_4, t_2\}\}$.

Definition 9 (Cartesian Union Operation): The Cartesian union operation between PS_1 and PS_2 is defined as $PS_1 \Theta PS_2$, where

$$\begin{aligned} PS_1 \theta PS_2 = \{ & path_g \Delta path_i, path_g \Delta path_{i+1}, \ldots, \\ & path_g \Delta path_j, path_{g+1} \Delta path_i, \\ & path_{g+1} \Delta path_{i+1}, \ldots, path_{g+1} \Delta path_j, \\ & \ldots\ldots \\ & path_h \Delta path_i, path_h \Delta path_{i+1}, \ldots, \\ & path_h \Delta path_j \} \end{aligned}$$

where Δ is the union operator between paths.

For example, in Fig. 4(b), there are two paths from p_1 to p_7, i.e., $\{t_1, t_3, t_4\}$ and $\{t_1, t_5, t_6\}$. We have $PS_a = \{\{t_1, t_3, t_4\}, \{t_1, t_5, t_6\}\}$. There is one path $\{t_1, t_7, t_8\}$ from p_1 to p_8. We have $PS_b = \{\{t_1, t_7, t_8\}\}$. Then, we have $PS_a \Theta PS_b = \{\{(t_1, t_3, t_4) \| (t_1, t_7, t_8)\}, \{(t_1, t_5, t_6) \| (t_1, t_7, t_8)\}\}$.

Definition 10 (Cartesian Intercross Operation): The Cartesian intercross operation between PS_1 and PS_2 is defined as $PS_1 O PS_2$

$$\begin{aligned} PS_1 O PS_2 = \{ & path_g / path_i, path_g / path_{i+1}, \ldots, \\ & path_g / path_j, path_{g+1} / path_i, \\ & path_{g+1} / path_{i+1}, \ldots, path_{g+1} / path_j, \\ & \ldots\ldots \\ & path_h / path_i, path_h / path_{i+1}, \ldots, \\ & path_h / path_j \} \end{aligned}$$

where / is the intercross operator between paths.

There are two processes shown in Fig. 4(c). Two or-split and or-join paths from p_{11} to p_{13} are denoted as a set $PS_a = \{\{t_{11}, t_{12}\}, \{t_{11}, t_{13}\}\}$, and one path from p_{21} to p_{23} is denoted as $PS_b = \{\{t_{21}, t_{22}\}\}$. After t_{12} and t_{22} fire or after t_{13} and t_{22} fire, we will have a resource conflict. Then, we get $PS_a O PS_b = \{\{t_{11}, t_{12}(t_{21}, t_{22})\}, \{t_{11}, t_{13}(t_{21}, t_{22})\}\}$.

We present several operations on time intervals and time interval sets as follows. They will be used to simplify the representation of computing and recording time information for transitions in a TWF-net when constructing and updating a sprouting graph.

Definition 11 (Max Operation of Time Intervals): Assume that a time interval $D_1 = [a, b]$, where $0 \le a \le b \le \infty$, and another time interval $D_2 = [c, d]$, where $0 \le c \le d \le \infty$. Then, $Max(D_1, D_2) = [\max(a, c), \max(b, d)]$, where the function $\max(x, y) = x$, if $y \le x$; $\max(x, y) = y$, otherwise.

For example, for $D_a = [2, 3]$ and $D_b = [3, 4]$, we have $Max(D_a, D_b) = [3, 4]$.

Definition 12 (Addition Operation of Time Intervals): Assume that a time interval $D_1 = [a, b]$, where $0 \le a \le b \le \infty$, and a time interval $D_2 = [c, d]$, where $0 \le c \le d \le \infty$. Then, time interval addition is defined as $D_1 + D_2 = [a + c, b + d]$.

For example, for $D_a = [2, 3]$ and $D_b = [3, 4]$, we have $D_a + D_b = [5, 7]$.

Definition 13 (Operations of Time Interval Sets): Let DS_1 and DS_2 be two sets of time intervals, where $DS_1 = \{D_g, D_{g+1}, \ldots, D_h\}$ and $DS_2 = \{D_i, D_{i+1}, \ldots, D_j\}$. The union of DS_1 and DS_2 is defined as $DS_1 \cup DS_2 = \{D_g, D_{g+1}, \ldots, D_h, D_i, D_{i+1}, \ldots, D_j\}$.

Letting D be a time interval, the addition of DS_1 and D is defined as $DS_1 \blacktriangle D = \{D_g + D, D_{g+1} + D, \ldots, D_h + D\}$, where "+" is addition operation of time interval.

The multiplication of DS_1 and DS_2 is defined by $DS_1 \Diamond DS_2$

$$\begin{aligned} DS_1 \Diamond DS_2 = \{&Max(D_g, D_i), Max(D_g, D_{i+1}), \ldots, \\ &Max(D_g, D_j) Max(D_{g+1}, D_i), \\ &Max(D_{g+1}, D_{i+1}), \ldots, Max(D_{g+1}, D_j) \\ &\ldots\ldots \\ &Max(D_h, D_i), Max(D_h, D_{i+1}), \ldots, \\ &Max(D_h, D_j)\}. \end{aligned}$$

For example, supposing that $DS_x = \{[1, 2], [2, 3]\}$, $DS_y = \{[1, 3], [3, 4]\}$, and $D = [1, 3]$, we have $DS_x \cup DS_y = \{[1, 2], [2, 3], [1, 3], [3, 4]\}$, $DS_x \blacktriangle D = \{[2, 5], [3, 6]\}$, and $DS_x \Diamond DS_y = \{[1, 3], [3, 4], [2, 3], [3, 4]\}$.

B. Sprouting Graph

Definition 14 (Sprouting Graph): A sprouting graph of a TWF-net model W is defined as a tuple (V, E), where V is a set of nodes and E is a set of directed arcs between nodes.

1) A node $v \in V$ is labeled (p, b) which corresponds to a place p in W. b is an ordered pair (Path_set, Time_set). Path_set is a set of paths. Time_set is a set of time intervals, and each time interval in Time_set represents the time that the corresponding path in Path_set costs. If the ith path in the set of paths is $\varnothing$, then the ith time interval of the set of time intervals is $\varnothing$ as well.
2) A directed arc $e \in E$ is labeled (t, d) which corresponds to a transition t in W. d is the time interval of t.

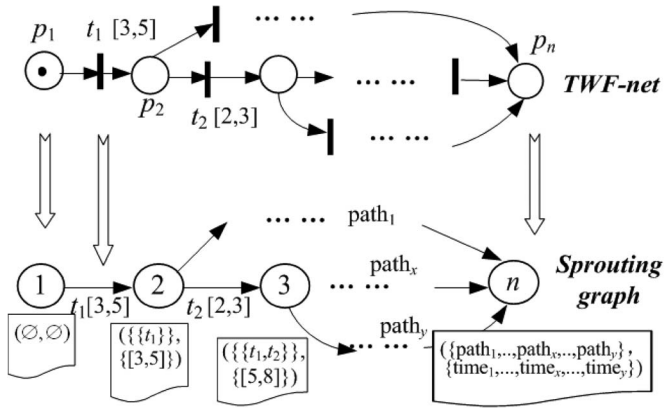

Fig. 5. Mapping between a TWF-net and a sprouting graph.

Note that, in a sprouting graph, an arc and its input/output nodes correspond to a transition and its input/output places. The set of time intervals of its input node is the possible enable time of the transition. The set of time intervals of its output node is the possible firing time of its corresponding transition. The sets of paths in its input and output nodes include the paths of instances executing before and after its corresponding transition.

There is a workflow process in the upper part of Fig. 5, which begins at the zeroth time unit. Each transition in a TWF-net corresponds to an arc in the sprouting graph. Node 1 corresponds to the initial place p_1; the path and time information are $\varnothing$ because p_1 is the initial marking place (i.e., $M_0(p_1) = 1$). Node 2 corresponds to p_2. Then, the path information is $\{t_1\}$ and the time information is [3, 5], because the path from the initial place to p_2 is $\{t_1\}$ and it takes at least three time units and at most five time units. Similarly, node n corresponds to the ending place p_n; the path information contains all the paths from p_1 to p_n, i.e., $\{path_1, \ldots, path_x, \ldots, path_y\}$, while the time information includes the corresponding times $\{\text{time}_1, \ldots, \text{time}_x, \ldots, \text{time}_y\}$.

For another example, as shown in Fig. 3, there are two workflow processes A and B. We assume that the two processes begin at the same time. The corresponding sprouting graph is shown in Fig. 6. Each transition in a TWF-net corresponds to an arc in the sprout graph, and its name and time interval are also labeled. Because there is only one path $\{t_{11}\}$ from p_{11} to p_{12} and a workflow instance needs to spend at least two and at most three time units on this path, node 3 corresponds to place p_{12}, and its corresponding path set is $\{\{t_{11}\}\}$ and time set is {[2, 3]}. Node 5 corresponds to place p_{13}, and its corresponding path set is $\{\{t_{11}, t_{12}\}, \{t_{11}(t_{21}), t_{13}\}\}$ and time set is {[7, 9], [8, 13]}. This shows that there are two paths $\{t_{11}, t_{12}\}$ and $\{t_{11}(t_{21}), t_{13}\}$ from p_{11} to p_{13}, and a workflow instance needs to spend, respectively, at least seven and at most nine time units on path $\{t_{11}, t_{12}\}$, or at least 8 and at most 13 time units on path $\{t_{11}(t_{21}), t_{13}\}$ in which $t_{11}(t_{21})$ is used

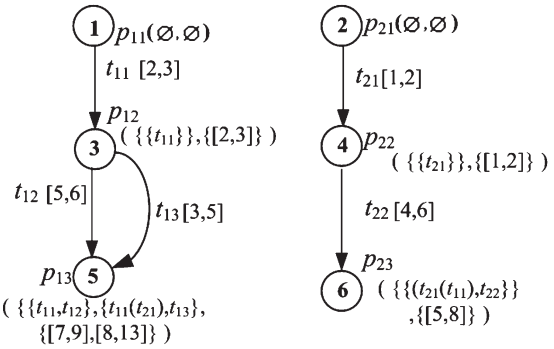

Fig. 6. Sprouting graph of the example in Fig. 3.

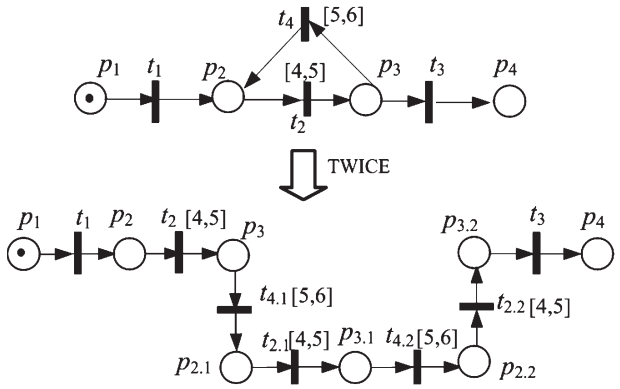

Fig. 7. Transformation of iteration structure in TWF-nets.

to denote that there is a resource constraint after t_{11} and t_{21} fire. The explanation of other nodes is omitted here.

C. Construct Sprouting Graph

Four kinds of basic control structures, namely, sequential, parallel, selective, and iterative structures, have been defined in the workflow reference model [22], [23]. The *iterative structure* occurs when some activities are scheduled iteratively. In the example of the upper part of Fig. 7, activities t_2 and t_4 may be scheduled many times before t_3 is scheduled.

If a TWF-net does not contain iterative structures, we can directly build a sprouting graph. Otherwise, we approximate the number of loops in a finite iterative structure and transform it to a sequence of transition by expanding cycles [28], as shown in Fig. 7. The resources used by a transition in an iterative control structure should also be repeatedly used by its new "copied" transitions in the transformed model. A detailed example about this is in Section VI.

When we construct the sprouting graph for a TWF-net, all the possible firing path and time information of each transition need to be computed and added in the nodes of the sprouting graph.

In order to reduce the complexity of the computing, the construction of sprouting graph can be divided into two phases [16]: 1) to obtain the transition sequence Q of the TWF-nets without considering the time interval of each transition and 2) to construct the sprouting graph based on Q.

The backward incidence and forward incidence matrices are adopted to obtain the transition sequence Q for the whole

F_1=

	t_{11}	t_{12}	t_{13}	t_{21}	t_{22}
p_{11}	1	0	0	0	0
p_{12}	0	1	1	0	0
p_{13}	0	0	0	0	0
p_{21}	0	0	0	1	0
p_{22}	0	0	0	0	1
p_{23}	0	0	0	0	0
p_{r1}	0	0	1	0	1

F_2=

	t_{11}	t_{12}	t_{13}	t_{21}	t_{22}
p_{11}	0	0	0	0	0
p_{12}	1	0	0	0	0
p_{13}	0	1	1	0	0
p_{21}	0	0	0	0	0
p_{22}	0	0	0	1	0
p_{23}	0	0	0	0	1
p_{r1}	0	0	1	0	1

F_{11}=

	t_{11}	t_{12}	t_{13}	t_{21}	t_{22}
p_{11}	0	0	0	0	0
p_{12}	0	1	1	0	0
p_{13}	0	0	0	0	0
p_{21}	0	0	0	1	0
p_{22}	0	0	0	0	1
p_{23}	0	0	0	0	0
p_{r1}	0	0	1	0	1

F_{22}=

	t_{11}	t_{12}	t_{13}	t_{21}	t_{22}
p_{11}	0	0	0	0	0
p_{12}	0	0	0	0	0
p_{13}	0	1	1	0	0
p_{21}	0	0	0	0	0
p_{22}	0	0	0	1	0
p_{23}	0	0	0	0	1
p_{r1}	0	0	1	0	1

Fig. 8. Computing procedure of backward/forward incidence matrices.

TWF-net [16]. This is shown as the following Algorithm 1.

Algorithm 1. Compute transition sequence
Input: TWF-nets $\{W_1, W_2, \ldots, W_m\}$.
Output: Transition sequence Q.
Step 1: Get the backward incidence matrix F_1 and forward incidence matrix F_2 for TWF-nets.
Step 2: Find a place p_i that satisfies $F_2(p_i, t) = 0$ for every transition t, and set $F_1(p_i, t) = 0$ for every transition t of the TWF-net.
Step 3: Find a transition t_j that satisfies that $F_1(p, t_j) = 0$ for every place p, and put the transition t_j into Q. Set $F_2(p, t_j) = 0$ for every place. If all transitions in models have been put into Q, then end; otherwise, go to step 2.

For example, the backward incidence matrix F_1 and forward incidence matrix F_2 for Fig. 3 are shown hereinafter. We can see that $F_2(p_{11}, t) = 0$ and $F_2(p_{21}, t) = 0$ for every transition t in Fig. 8. We choose p_{11} first and set $F_1(p_{11}, t) = 0$ for every transition t as shown in F_{11}. Then, we find that transition t_{11} satisfies $F_{11}(p, t_{11}) = 0$ for any place p. We put t_{11} into Q and set $F_2(p, t_{11}) = 0$ for every place as shown in F_{22}. We do the same to p_{21}, p_{r1}, etc. Finally, we get the transition sequence Q of the TWF-net as $\{t_{11}, t_{21}, t_{12}, t_{13}, t_{22}\}$.

Assuming that there are n transitions and m places in the model, the time complexity of Algorithm 1 is $\mathrm{O}(nm)$.

After obtaining the transition sequence Q of the TWF-nets according to Algorithm 1, we construct the sprouting graph based on Q. The main idea of the constructing procedure is to compute and record firing time and path of each transition in Q[16]. Hence, the procedure for each transition is divided into three computing steps: 1) to get its possible enabled time intervals; 2) to compute the firing time and path; and 3) to construct the node and arc to record information.

Assume that t_i is the first transition from transition sequence as shown in Fig. 9, and its time interval is $[d(t_i), D(t_i)]$.

Computing step 1: Get its possible enabled time intervals

Supposing that $p_j \in^{\bullet} t_i \ \ (j = s, s+1, \ldots, r)$, the path sets and time sets of the node corresponding

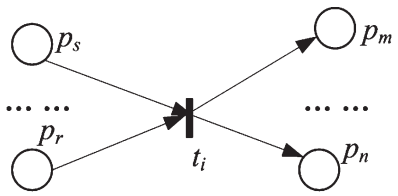

Fig. 9. Any one transition in the Q.

to p_j are Path_set$_j$ and Time_set$_j$, respectively. The possible enabled time intervals of t_i are $PB(t_i) = (\text{Time\_set}_s \Diamond \ldots \Diamond \text{Time\_set}_r)$.

Computing step 2: Compute the firing time and path

We discuss it in two cases according to the resource constraints between t_i and other process.

Case 1: It does not have resource constraints with others: The path and time information when t_i is triggered: Path_set $= (\text{Path\_set}_s \Theta \ldots \Theta \text{Path\_set}_r)\Delta t_i$, and Time_set $= PB(t_i) \blacktriangle [d(t_i), D(t_i)]$.

Case 2: It has resource constraints with others: Assume that t_j has resource constraints with t_i and its time interval is $[d(t_j), D(t_j)]$, the possible enabled time intervals $PB(t_j) = (\text{Time\_set}_m \Diamond \ldots \Diamond \text{Time\_set}_n)$, in which $p_k \in^{\bullet} t_j$ $(k = m, m+1, \ldots, n)$ and the time sets of node corresponding to p_k are Time_set$_k$. We adjust its enabled time intervals in order to avoid resource conflict in execution based on the FCFS policy: For $\forall\ [c, d] \in PB(t_i)$ and $\forall\ [a, b] \in PB(t_j)$, if $a \leq c$, $[c, d]$ is modified as $[\max\{a + d(t_j), c\}, \max\{b + D(t_j), d\}]$.

The time and the path information of this transition after adjustment are Time_set $= PB(t_i) \blacktriangle [d(t_i), D(t_i)]$ and Path_set $= (\text{Path\_set}_s \text{OPath\_set}_{s+1} \text{O} \ldots \text{OPath\_set}_r)\ \Delta t_i$.

Computing step 3: Construct the node and arc to record information

For $\forall\ p_o \in t_i^{\bullet}$ $(o = x, x+1, \ldots, y)$, assume that the path sets and time sets for p_o are $(\text{Path\_set}_o, \text{Time\_set}_o)$, and we will create or modify nodes according to the following two cases.

Case 1: If there exists a node corresponding to p_o: Draw directed arcs labeled with $(t_i, [d(t_i), D(t_i)])$ from each node of $p_j (s \leq j \leq r)$ to p_o, and change $(\text{Path\_set}_o, \text{Time\_set}_o)$ to $(\text{Path\_set} \odot \text{Path\_set}_o, \text{Time\_set} \cup \text{Time\_set}_o)$.

Case 2: If there does not exist a node corresponding to p_o: Create a new node for p_o which is labeled with (Path_set, Time_set), and draw directed arcs labeled with $(t_i, [d(t_i), D(t_i)])$ from each node of $p_j (s \leq j \leq r)$ to p_o.

After the aforementioned three computing steps, we delete t_i from Q. If $Q = \varnothing$, then the procedure ends. Otherwise, we get the next transition and go to this procedure again.

Based on the earlier discussion, we present an algorithm to construct a sprouting graph in Algorithm 2.

Algorithm 2. Construct sprouting graph

Input: TWF-nets $\{W_1, W_2, \ldots, W_w\}$ and transition sequence Q.

Output: Sprouting graph G, whose nodes and arcs are corresponding to places and transitions in TWF-nets.

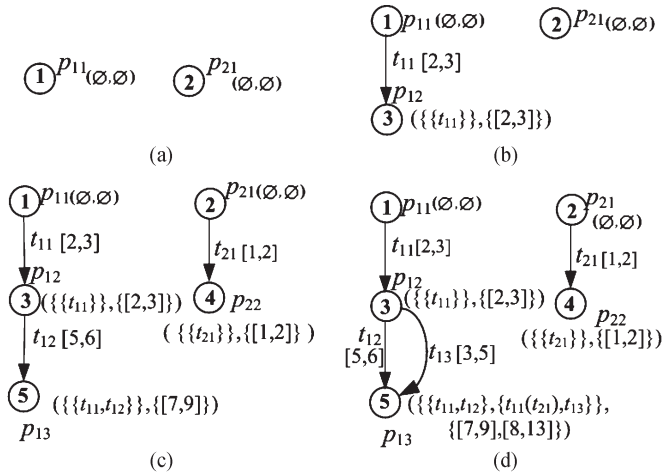

Fig. 10. Build a sprouting graph for the example in Fig. 3.

/* Initialization of root nodes */

Step 1: Create a root node for each initial marking place p for each $W \in \{W_1, W_2, \ldots, W_w\}$. Label the node corresponding to p with $(p, (\varnothing, \varnothing))$.

/* Obtain the first transition from Q */

Step 2: Select the first transition t_i from Q, and assume its time interval as $[d(t_i), D(t_i)]$.

/* Compute and record firing time and path for each transition */

Steps 3–5: Follow the aforementioned computing steps 1–3.

Step 6: Delete t_i from Q. If $Q = \varnothing$, then the algorithm ends. Otherwise, go to step 2.

Assuming that there are n transitions and m places in the models, the time complexity of Algorithm 2 is $\text{O}(nm)$ when there is no resource constraint and $\text{O}(nm^2)$ when there is resource constraint.

For example, in Fig. 3, we have TWF-nets $\{W_1, W_2\}$ and transition sequence $Q = \{t_{11}, t_{21}, t_{12}, t_{13}, t_{22}\}$. We first construct two root nodes 1 and 2, corresponding to p_{11} and p_{21} as shown in Fig. 10(a). The path sets and time sets of the two nodes are $(\varnothing, \varnothing)$. Since $p_{11} \in^{\bullet} t_{11}$, Path_set $= \varnothing \Delta t_{11} = \{\{t_{11}\}\}$ and Time_set $= \varnothing \blacktriangle [2, 3] = \{[2, 3]\}$. A new node 3 corresponding to p_{12} is created, which is labeled with $(\{\{t_{11}\}\}, \{[2, 3]\})$, and a directed arc labeled with $(t_{11}, [2, 3])$ is drawn from node p_{11} to it, as shown in Fig. 10(b). Then, we delete t_{11} from Q and go to the next loop.

In the same way, we construct nodes 4 and 5. Now, $Q = \{t_{13}, t_{22}\}$, and the sprouting graph is shown in Fig. 10(c). We select the next transition t_{13} from Q. Note that there is a resource constraint between t_{13} and t_{22}. According to case 2 step 4 in Algorithm 2, $t_{22} \in RC(t_{13})$, we have $[2, 3] \in PB(t_{13}), [1, 2] \in PB(t_{22})$. Thus, the enabled time of t_{13} is later than the enabled time of t_{22}. t_{22} will get the resource first. [2, 3] is modified as $[\max\{1 + 4, 2\}, \max\{2 + 6, 3\}] = [5, 8]$. Then, Time_set $= [5,8] \blacktriangle [d(t_{13}), D(t_{13})] = [5,8] + [3,5] = [8, 13]$, and Path_set $= (\{\{t_{11}\}\} \text{O} \{\{t_{21}\}\}) \Delta t_{13} = \{\{t_{11}(t_{21}), t_{13}\}\}$.

Then, according to case 1 step 5 in Algorithm 2, a directed arc labeled with $(t_{13}, [3, 5])$ is drawn from node 3 to node 5,

and the path sets and time sets of node 5 are changed to $(\{\{t_{11}, t_{12}\}, \{t_{11}(t_{21}), t_{13}\}\}, \{[7, 9], [8, 13]\})$. We then process the final transition t_{22} from Q, and the final sprouting graph is shown in Fig. 6.

IV. Dynamic Checking Based on Sprouting Graph

After constructing the sprouting graph of a TWF-net with resource constraints, we discuss how to dynamically check temporal constraints based on the sprouting graph.

A. Dynamically Update the Sprouting Graph

A checkpoint selection strategy is responsible for selecting checkpoints for conducting temporal verification at runtime execution stage. Currently, there are several checkpoint selection strategies. Eder *et al.* [10] take every activity as a checkpoint at run time. Chen *et al.* [12] present a new runtime checkpoint selection strategy which illustrates how to dynamically select appropriate checkpoints based on the active interval of activities along the workflow execution. What policy is adopted to select checkpoints is beyond the scope of this paper, and we assume that checkpoints have been selected in the execution of workflow processes based on a corresponding selection strategy.

At checkpoint H, dynamic checking of temporal constraint $D_R(t_i, t_j) \leq s$ can be divided into two phases, i.e., updating sprouting graph and checking of temporal constraints.

In order to update the sprouting graph at the checkpoints, the time information of nodes in the graph needs to be recomputed with the time information from the transitions triggered. Since the structure of TWF-net does not change, the path information in the sprouting graph is unchanged—only the time information needs to be computed.

First, we delete already fired transitions from Q before computing, since the time of fired transitions is definite, which means that it has removed the randomness.

Second, we make the transitions have the same new starting time point and update the path information of input nodes of the arcs corresponding to the enabled transitions. For each enabled transition t, assume that its enabled time is H_t. Modify the time interval $[d(t), D(t)]$ of the arc corresponding to t as $[d(t) - (H - H_t), D(t) - (H - H_t)]$ and the time set of the input nodes of the arc as [0, 0]. The path set of the input nodes of the arc is also set as the real executing path of instance.

Finally, for each transition from Q, we follow the steps in Algorithm 2.

In this way, we present an algorithm to update the sprouting graph in Algorithm 3.

Algorithm 3. Update sprouting graph

Input: Sprouting graph G, transition sequence Q, and checkpoint H.

Output: Updated sprouting graph G'.

/* Make the transitions have the same new starting time point and update the path information of input nodes of the arcs corresponding to the enabled transitions */

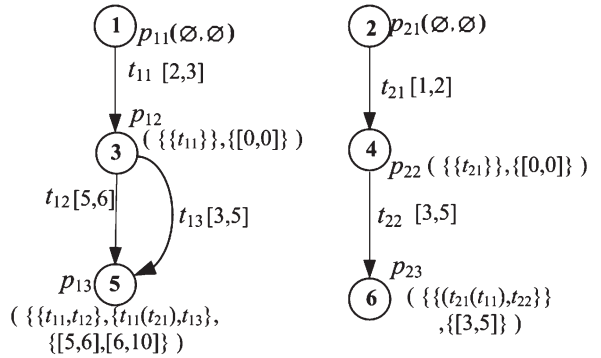

Fig. 11. Updated sprouting graph at the third time unit.

Step 1: Delete already fired transitions from Q. For each enabled transition t, assume that its enabled time is H_t. Modify the time interval $[d(t), D(t)]$ of the arc corresponding to t as $[d(t) - (H - H_t), D(t) - (H - H_t)]$ and the time set of the input nodes of the arc as [0, 0]. The path set of the input nodes of the arc is also set as the practical executing path of instance.

/* Obtain the first transition from Q */

Step 2: Select the first transition that has not been fired as t_i from Q, and assume its time interval as $[d(t_i), D(t_i)]$.

Steps 3–6: Follow steps 3–6 in Algorithm 2.

Assume that there are n transitions in Q, which are not fired or enabled, and there are m places in the models. The time complexity of the Algorithm 3 is $\mathrm{O}(nm)$ when there is no resource constraint and $\mathrm{O}(nm^2)$ when there is resource constraint.

For example, assume that we want to update the sprouting graph in Fig. 6 at the third time unit. Assume that the firing of t_{11} takes three time units and that of t_{21} takes two time units. Then, t_{22} is enabled at the second time unit, and t_{12} and t_{13} are enabled at the third time unit. According to step 1, we modify the time interval of the arc corresponding to t_{22} as $[4 - (3 - 2), 6 - (3 - 2)] = [3, 5]$ and set the time set of nodes 3 and 4 as {[0, 0]}. According to step 1, transitions t_{11} and t_{21} that have already fired are deleted from Q; then, we have $Q = \{t_{12}, t_{13}, t_{22}\}$. The path information of nodes 1–4 remains unchanged.

Now, the first transition in Q is t_{12}. According to step 3, since the input place of t_{12} is p_{12}, its possible enabled time interval is [0, 0]. Because $RC(t_{12}) = \varnothing$, the Time_set is $\{[0, 0]\}\blacktriangle[5, 6] = \{[5, 6]\}$ and Path_set $= \{\{t_{11}, t_{12}\}\}$, and we update the path and time sets of node 5 to $(\{\{t_{11}, t_{12}\}\}, \{[5, 6]\})$ and delete t_{12} from Q.

Then, the new first transition in Q is t_{13}. $t_{22} \in RC(t_{13})$, and we have $[0, 0] \in PB(t_{13})$ and $[0, 0] \in PB(t_{22})$. Since t_{22} is enabled at the second time unit and t_{13} is enabled at the third time unit, t_{22} is enabled before t_{13}. We have Time_set $= \{[3, 5]\}\blacktriangle[3, 5] = \{[6, 10]\}$ and Path-set $= \{\{t_{11}(t_{21}), t_{13}\}\}$. Therefore, we update the path and time sets of node 5 to $(\{\{t_{11}, t_{12}\}, \{t_{11}(t_{21}), t_{13}\}\}, \{[5, 6], [6, 10]\})$ and delete t_{13} from Q.

Then, the new first transition in Q is t_{22}. We go to the next loop. Finally, the updated sprouting graph is shown in Fig. 11.

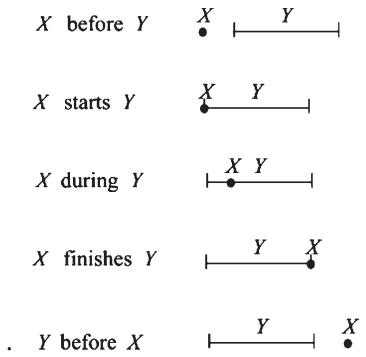

Fig. 12. Temporal relations between time point X and interval Y.

B. Check Temporal Constraints Based on Updated Sprouting Graph

Now, we can check whether $D_R(t_i, t_j) \leq s$ based on the updated sprouting graph at checkpoint H, i.e., check if t_j ends no later than s time units after t_i starts.

There are two kinds of status of t_i.

Condition A: t_i has been fired or enabled before checkpoint H. We assume that its enabled time is S and the time set of output nodes of the arc corresponding to t_j is Time_set$_j$. For $\forall\ [a, b] \in$ Time_set$_j$, where a is the earliest ending time of t_j and b is the latest ending time.

We check if Y is before X, where $X = s$ and $Y = [a + H - S, b + H - S]$. There are three cases as shown in Fig. 12.

Case 1: $s < a + H - S$. This corresponds to the case "X before Y." The temporal constraint is violated.

Case 2: $b + H - S \leq s$. This corresponds to the cases "X finishes Y" and "Y before X." The temporal constraint is satisfied.

Case 3: $a + H - S \leq s < b + H - S$. This corresponds to the cases "X starts Y" and "X during Y." The temporal constraint cannot be decided at H.

Condition B: t_i has not been fired or enabled before checkpoint H. We assume that the time set of the input nodes of arc corresponding to t_i is Time_set$_i$ and the time set of the output nodes of arc corresponding to t_j is Time_set$_j$. For $\forall\ [c, d] \in$ Time_set$_i$ and $\forall\ [a, b] \in$ Time_set$_j$, we check if Y is before X, where $X = s$ and $Y = [a - d, b - c]$. There are three cases.

Case 1: $s < a - d$. This corresponds to the case "X before Y." The temporal constraint is violated.

Case 2: $b - c \leq s$. This corresponds to the cases "X finishes Y" and "Y before X." The temporal constraint is satisfied.

Case 3: $a - d \leq s < b - c$. This corresponds to the cases "X starts Y" and "X during Y." The temporal constraint cannot be decided at H.

Based on the aforementioned statements, we propose the checking procedure of temporal constraints in the following Algorithm 4.

Algorithm 4. Check temporal constraints

Input: Updated sprouting graph G' at checkpoint H and temporal constraint $D_R(t_i, t_j) \leq s$ for checking.

Output: Results for temporal violation and the path where the violation happens.

Step 1: We take Time_set$_j$ for the output node of the arc corresponding to t_j. If t_i has already been fired or enabled, go to step 2. Otherwise, go to step 3.

/* Condition A: t_i has been fired or enabled before checkpoint H. */

Step 2: If t_i has been fired or enabled, S is the enabled time. For $\forall\ [a, b] \in$ Time_set$_j$, we check if Y is before X, where $X = s$ and $Y = [a + H - S, b + H - S]$ according to the cases in condition A. Go to step 4.

/* Condition B: t_i has not been fired or enabled before checkpoint H. */

Step 3: We take Time_set$_i$ for the input node of arc corresponding to t_i. For $\forall\ [c, d] \in$ Time_set$_i$ and $\forall\ [a, b] \in$ Time_set$_j$, we check if Y is before X, where $X = s$ and $Y = [a - d, b - c]$ according to the cases in condition B. Go to step 4.

/* If the temporal constraint is violated, return the erroneous paths. */

Step 4: If the temporal constraint is violated, return the erroneous paths according to the following cases. Otherwise, the algorithm ends.

/*If t_i and t_j are in the same process, the path corresponding to $[a, b]$ includes t_i and t_j. Thus, it is considered to solve the violations.*/

Case 1: Two transitions t_i and t_j are in the same process. The path corresponding to $[a, b]$ in Path_set$_j$ is returned.

/*If t_i and t_j are not in the same process and t_i has not been fired or enabled before checkpoint H, two paths corresponding to $[c, d]$ of t_i and $[a, b]$ of t_j are considered to solve the violations. But only the path corresponding to $[a, b]$ of t_j is returned if t_i has been fired or enabled before checkpoint H.*/

Case 2: Two transitions t_i and t_j are not in the same process. If t_i has not been fired or enabled before checkpoint H, the path corresponding to $[a, b]$ in Path_set$_j$ and the path corresponding to $[c, d]$ in Path_set$_i$ are returned together. But only the path corresponding to $[a, b]$ of t_j is returned, if t_i has been fired or enabled before checkpoint H.

The time complexity of Algorithm 4 is constant.

For example, suppose that we want to check a temporal constraint $D_R(t_{12}, t_{22}) \leq 6$ for the workflow model in Fig. 3 at a checkpoint (third time unit). Assume that the firing of t_{11} takes three time units and that of t_{21} takes two time units. t_{22} is enabled at the second time unit, and t_{12} and t_{13} are enabled at the third time unit. The updated sprouting graph is shown in Fig. 11. Because t_{12} is enabled at the third time unit, $S = 3$. The time set of output nodes of the arc corresponding to t_{22} is {[3, 5]}. We take the Time_set for node 6, i.e., {[3, 5]}. We have $X = s = 6$, $[a, b] = [3, 5]$, and $H = S = 3$. Then, $Y = [a + H - S, b + H - S] = [3, 5]$. Because $5 < 6$, we know that $D_R(t_{12}, t_{22}) \leq 6$ is satisfied at the third time units based on step 2 of Algorithm 4.

For another example, suppose that we want to check temporal constraint $D_R(t_{21}, t_{22}) \leq 4$. Because t_{21} is enabled at the initial time, $S = 0$. Then, we have $X = s = 4$, $[a, b] = [3, 5]$, and $H = 3$. Thus, $Y = [a + H - S, b + H - S] = [6, 8]$. Because $4 < 6$, $D_R(t_{21}, t_{22}) \leq 4$ is violated at the third time units. Since t_{21} and t_{22} are in the same process and the path $\{t_{21}(t_{11}), t_{22}\}$ is corresponding to time interval $[a, b] = [3, 5]$, $\{t_{21}(t_{11}), t_{22}\}$

is returned based on step 4 of Algorithm 4. This means that the path $\{t_{21}(t_{11}), t_{22}\}$ is the erroneous path for the violation of $D_R(t_{21}, t_{22}) \leq 4$ at the third time units.

Note that we can check more than one temporal constraint by using the same updated sprouting graph. The detailed example is illustrated in Section VI. Thus, our approach is scalable, i.e., we do not have to create a new sprouting graph for checking a new temporal constraint. Our approach has higher efficiency, particularly when it deals with a large number of temporal constraints.

V. Solution to Temporal Violations

If a temporal constraint $D_R(t_i, t_j) \leq s$ is violated at checkpoint H on a specific path, the designer needs to modify the workflow models in order to make the temporal constraints satisfied again.

A. Solution to Temporal Violations

Generally speaking, there are several options for the workflow designer, e.g., inserting/removing activities, adding/removing resources, modifying the duration of activities, etc. [34]. To modify the duration of activities may be the easiest one since it can keep the original workflow topological structure. Assume that we can "reduce" the time interval $[a, b]$ to $[\max\{(a - Dur), 0\}, b - Dur]$ or "expand" the time interval $[a, b]$ to $[a + Dur, b + Dur]$, where Dur is the time value that needs to be decided.

If a temporal constraint $D_R(t_i, t_j) \leq s$ is violated at a checkpoint H, there are two conditions.

Condition A: Two transitions are in the same process.

According to step 4 of Algorithm 4, the only erroneous path for the temporal violation is returned. This erroneous path should include t_i and t_j. We assume that the path formalization is $\{t_x, \ldots, t_i, \ldots, t_j\}$.

The solution is to reduce the time intervals of transitions between t_i and t_j in the path, which has not been fired or enabled at checkpoint H. Here, the designer needs to know how much time should be reduced between t_i and t_j to solve the problem.

There are two cases.

Case 1: t_i has been fired or enabled at checkpoint H.

Assume that $[a, b] \in \text{Time\_set}_j$ corresponds to the path where there is a temporal violation. We set $Reduce = b + H - S - s$ as the minimal time value to be reduced in order to make $(b - Reduce) + H - S \leq s$. Moreover, we need to consider the time delay caused by waiting resources because some transitions on the path $\{t_i, \ldots, t_j\}$, which are not fired or enabled at checkpoint H, may have resource constraints with other transitions. In the worst case, the waiting time is $\sum_{z=1}^{x} u_z$, where u_z $(1 \leq z \leq x)$ is the maximum firing time of the zth transition which has resource constraint with the transitions on the path $\{t_i, \ldots, t_j\}$. Hence, the minimal time to be reduced is $Dur = Reduce + \sum_{z=1}^{x} u_z$.

Case 2: t_i has not been fired or enabled at checkpoint H.

Assume that $[c, d] \in \text{Time\_set}_i$ and $[a, b] \in \text{Time\_set}_j$ are responding to the erroneous path which has temporal violation. We set $Reduce = b - c - s$ as the minimal time value to be reduced in order to make $(b - Reduce) - c \leq s$. Similarly, we consider the time delay caused by waiting resources and set the minimal time value to be reduced as $Dur = Reduce + \sum_{z=1}^{x} u_z$.

Condition B: Two transitions are in different processes.

According to step 4 of Algorithm 4, there are two cases.

Case 1: t_i has been fired or enabled at checkpoint H.

The solution is to reduce the time intervals of transitions before t_j in the returned erroneous path $\{t_b, \ldots, t_j\}$ }, which has not been fired or enabled at checkpoint H.

Similar to case 1 in condition A, assume that $[a, b] \in \text{Time\_set}_j$ is corresponding to the path where there is a temporal violation. We set $Reduce = b + H - S - s$ as the minimal time value to be reduced to make $(b - Reduce) + H - S \leq s$. Moreover, we need to consider the time delay caused by waiting resources because some transitions on the path $\{t_b, \ldots, t_j\}$, which are not fired or enabled at checkpoint H, may have resource constraints with other transitions. In the worst case, the waiting time is $\sum_{z=1}^{x} u_z$, where u_z $(1 \leq z \leq x)$ is the maximum firing time of the zth transition which has resource constraint with the transitions on the path $\{t_b, \ldots, t_j\}$. Hence, the minimal time to be reduced is $Dur = Reduce + \sum_{z=1}^{x} u_z$.

Case 2: t_i has not been fired or enabled at checkpoint H.

Similar to case 2 in condition A, assume that $[c, d] \in \text{Time\_set}_i$ and $[a, b] \in \text{Time\_set}_j$ are responding to the erroneous path which has temporal violation. We set $Reduce - Expand = b - c - s$ in order to make $(b - Reduce) - (c - Expand) \leq s$. Here, $Reduce$ is the minimal time value to be reduced on the returned erroneous path $\{t_b, \ldots, t_j\}$, and $Expand$ is the minimal time value to be expanded on the returned erroneous path $\{t_a, \ldots, t_i\}$. For simplicity, we only consider the "reduce" method. Then, we have $Expand = 0$ and $Reduce = b - c - s$. Similarly, we consider the time delay caused by waiting resources and set the minimal time value to be reduced as $Dur = Reduce + \sum_{z=1}^{x} u_z$.

We assume that the transitions on the path, which have not been fired or enabled at checkpoint H, are $t_r, \ldots, t_s$. The time interval of $t_r, \ldots, t_s$ is $[d(t_r), D(t_r)], [d(t_{r+1}), D(t_{r+1})], \ldots, [d(t_s), D(t_s)]$. $x_r, \ldots, x_s$ are the time values to be decided to reduce the time interval of $t_r, \ldots, t_s$. Then, we can formalize the problem as

$$x_r + x_{r+1} + \ldots x_s = Dur$$

s.t.

$$\begin{aligned} d(t_r) - x_r &\geqq 0 \\ d(t_{r+1}) - x_{r+1} &\geqq 0 \\ &\ldots \\ d(t_s) - x_s &\geqq 0 \\ x_r \geqq 0, \ldots, x_s &\geqq 0. \end{aligned}$$

Obviously, we have multiple solutions for both conditions.

B. Correctness Analysis

By duration modification, we can handle a specific temporal violation. We have to check if the solution will affect the other satisfied temporal constraints. If none of the other satisfied

TABLE I
ACTIVITIES OF TWO WORKFLOW PROCESSES

Activities	Transitions	Time intervals
analyze requirement	t_{11}	[2,2]
write a requirement document	t_{12}	[1,2]
design software	t_{14}	[1,2]
design electrical circuit part	t_{15}	[2,3]
reconstruct based on other similar models	t_{16}	[5,5]
approved by head engineer	t_{18}	[4,5]
manufacture	t_{19}	[4,6]
marketing	t_{21}	[1,1]
feasibility analysis	t_{22}	[1,2]
write technical document	t_{23}	[5,5]
approved by head engineer	t_{24}	[3,4]
manufacture	t_{25}	[2,4]
additional market survey	t_{26}	[1,1]
and–split	t_{13}	[0,0]
and–join	t_{17}	[0,0]

temporal constraints are affected, we can adopt the solution. Otherwise, we have to make a new solution to the new temporal violation following the same method until there is no more temporal violation. If this method cannot converge, there should be some conflicts in temporal constraints. This is beyond the scope of this paper and will be our future work.

VI. CASE STUDY

In this section, we illustrate an application of our approach by a real example.

A. Real Scenario

The production of a new "Lenovo" cell phone is mainly composed of two processes, i.e., production of an electronic main board and production of peripheral parts as shown in Table I. The first process is composed of requirement analysis to final production. The second one is composed of marketing to final production. The two processes execute concurrently, and there are two human resources (an analyst and a head engineer) shared by these processes.

The market competition of electronic products, particularly cell phone, is usually severe. In order to increase the speed to release products into the market and obtain a bigger market share, two following temporal constraints are specified.

1) $D_R(t_{23}, t_{25}) \leq 20$. Manufacturing of peripheral parts ends no later than 20 time units after writing a technical document begins.
2) $D_R(t_{23}, t_{19}) \leq 16$. Manufacturing of electronic main board ends no later than 16 time units after writing a technical document of peripheral parts begins.

In order to ensure that the temporal constraints are satisfied at run time, we need to dynamically check the two temporal constraints at checkpoints. If temporal constraints are violated, some solutions should be taken.

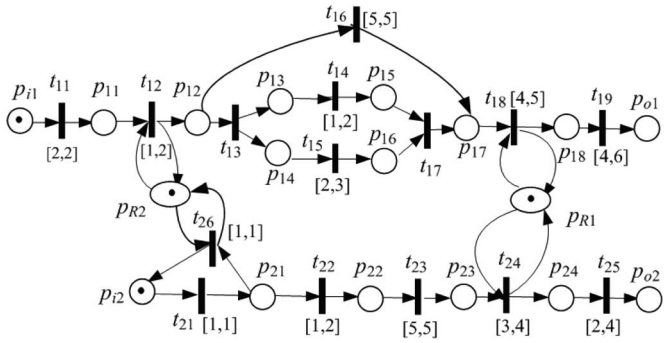

Fig. 13. TWF-net model for the example.

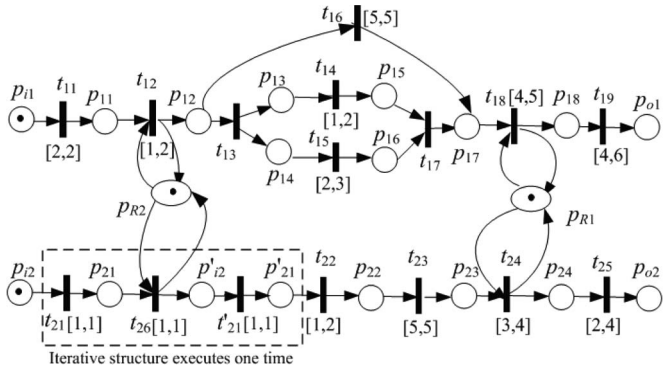

Fig. 14. Transformation of iteration structure.

B. Build the TWF-Net

The transitions for these activities and corresponding time information in the workflow processes are shown in Table I.

As shown in Fig. 13, two transitions t_{18} and t_{24} share a human resource (the head engineer) represented by P_{R1}. t_{12} and t_{26} share a human resource (the analyst) represented by P_{R2}.

C. Build the Sprouting Graph

For simplification, we assume that the two processes begin at the same time and the iterative structure in the second process is executed one time. If one process begins later, we can always insert a dummy transition with exactly the same interval before the initial place of this process. After transformation of iteration structure, the TWF-net of the example is shown in Fig. 14.

First, we get transition sequence $Q = \{t_{11}, t_{21}, t_{26}, t'_{21}, t_{12}, t_{22}, t_{13}, t_{14}, t_{23}, t_{15}, t_{16}, t_{17}, t_{24}, t_{18}, t_{25}, t_{19}\}$ by Algorithm 1. Second, we construct the sprouting graph by Algorithm 2 as shown in Fig. 15 and Table II.

D. Update the Sprouting Graph

Suppose that we choose the fifth time unit after the process instances started as the checkpoint. In addition, we assume that the firing of t_{21}, t_{26}, and t'_{21} all takes one time unit, that of t_{22} takes two time units, and that of t_{11} takes four time units. Thus, t_{23} and t_{12} are enabled at the fifth and fouth time units. Based

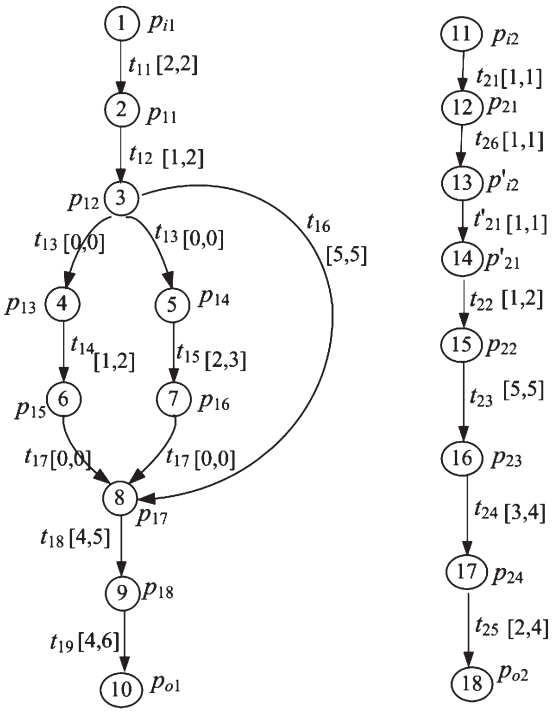

Fig. 15. Sprouting graph for the example.

TABLE II
NODES OF SPROUTING GRAPH

Node	Time sets	Path sets
1	∅	∅
2	{[2,2]}	{{t_{11}}}
3	{[3,4]}	{{t_{11}(t_{21}),t_{12}}}
4	{[3,4]}	{{t_{11}(t_{21}),t_{12},t_{13}}}
5	{[3,4]}	{{t_{11}(t_{21}),t_{12},t_{13}}}
6	{[4,6]}	{{t_{11}(t_{21}),t_{12},t_{13},t_{14}}}
7	{[5,7]}	{{t_{11}(t_{21}),t_{12},t_{13},t_{15}}}
8	{[5,7], [8,9]}	{{(t_{11}(t_{21}),t_{12},t_{13},t_{14}) ‖ (t_{11}(t_{21}),t_{12},t_{13},t_{15}),t_{17}}, {t_{11}(t_{21}),t_{12},t_{16}}}
9	{[9,12], [12,14]}	{{(t_{11}(t_{21}),t_{12},t_{13},t_{14}) ‖ (t_{11}(t_{21}),t_{12},t_{13},t_{15}),t_{17}(t_{21}(t_{11}),t_{26}, t'_{21}, t_{22},t_{23}), t_{18}}, {t_{11}(t_{21}),t_{12}, t_{16}(t_{21}(t_{11}),t_{26}, t'_{21},t_{22}, t_{23}), t_{18}}}
10	{[13,18], [16,20]}	{{(t_{11}(t_{21}),t_{12},t_{13},t_{14}) ‖ (t_{11}(t_{21}),t_{12},t_{13},t_{15}),t_{17}(t_{21}(t_{11}),t_{26}, t'_{21},t_{22},t_{23}), t_{18},t_{19}}, {t_{11}(t_{21}),t_{12},t_{16}(t_{21}(t_{11}),t_{26}, t'_{21},t_{22}, t_{23}),t_{18},t_{19}}}
11	∅	∅
12	{[1,1]}	{{t_{21}}}
13	{[2,2]}	{{t_{21}(t_{11}),t_{26}}}
14	{[3,3]}	{{t_{21}(t_{11}),t_{26}, t'_{21}}}
15	{[4,5]}	{{t_{21}(t_{11}),t_{26},t'_{21},t_{22}}}
16	{[9,10]}	{{ t_{21}(t_{11}),t_{26},t'_{21},t_{22},t_{23}}}
17	{[12,16], [15,18]}	{{t_{21}(t_{11}),t_{26},t'_{21},t_{22},t_{23}((t_{11}(t_{21}),t_{12},t_{13},t_{14}) ‖ (t_{11}(t_{21}),t_{12}, t_{13}, t_{15}),t_{17}),t_{24}}, {t_{21}(t_{11}),t_{26}, t'_{21},t_{22},t_{23}(t_{11}(t_{21}),t_{12}, t_{16}) , t_{24}}}
18	{[14,20], [17,22]}	{{t_{21}(t_{11}),t_{26},t'_{21}, t_{22},t_{23}((t_{11}(t_{21}),t_{12},t_{13},t_{14}) ‖ (t_{11}(t_{21}),t_{12}, t_{13},t_{15}), t_{17}),t_{24},t_{25}},{ t_{21}(t_{11}),t_{26}, t'_{21},t_{22},t_{23}(t_{11}(t_{21}),t_{12}, t_{16}), t_{24}, t_{25}}}

on the time information of t_{11}, t_{21}, t_{26}, t'_{21}, t_{22}, t_{23}, and t_{12}, we update the sprouting graph according to Algorithm 3 as shown in Fig. 16 and Table III.

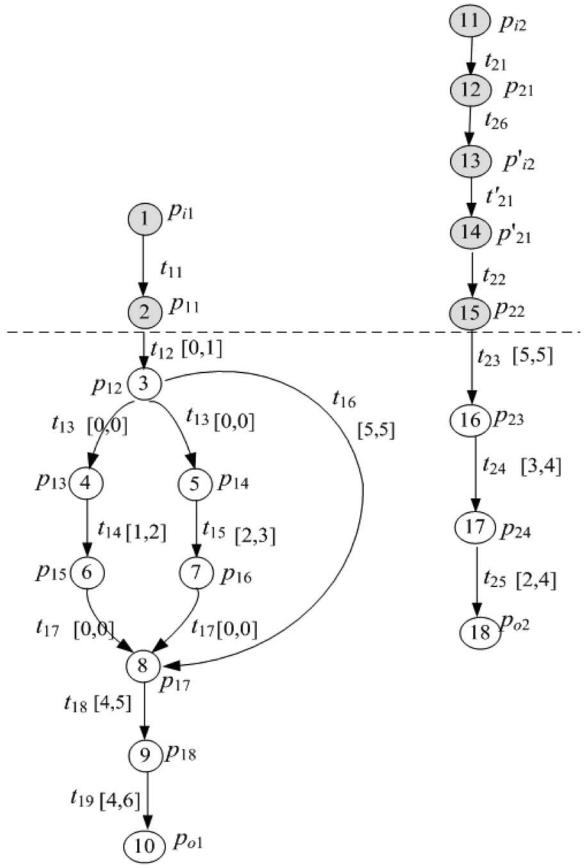

Fig. 16. Updated sprouting graph for the example.

E. Check the Temporal Constraints

1) $D_R(t_{23}, t_{25}) \leq 20$. Because t_{23} is enabled at the fifth time unit, $S = 5$. We take the Time_set of node 18, i.e., {[11, 17], [10, 13]}. This means that the time spent on two paths $\{t_{21}(t_{11}), t_{26}, t'_{21}, t_{22}, t_{23}((t_{11}(t_{21}), t_{12}, t_{13}, t_{14}) \| (t_{11}(t_{21}), t_{12}, t_{13}, t_{15}), t_{17}), t_{24}, t_{25}\}$ and $\{t_{21}(t_{11}), t_{26}, t'_{21}, t_{22}, t_{23}(t_{11}(t_{21}), t_{12}, t_{16}), t_{24}, t_{25}\}$ are at most 17 and 13 time units, respectively.

   For [11, 17], we have $X = s = 20$ and $H = S = 5$. Then, $Y = [11 + H - S, 17 + H - S] = [11, 17]$. Because $17 < 20$, we know that the temporal constraint is satisfied on path $\{t_{21}(t_{11}), t_{26}, t'_{21}, t_{22}, t_{23}((t_{11}(t_{21}), t_{12}, t_{13}, t_{14}) \| (t_{11}(t_{21}), t_{12}, t_{13}, t_{15}), t_{17}), t_{24}, t_{25}\}$ according to condition A case 2 in Algorithm 4.

   For [10, 13], we have $Y = [10 + H - S, 13 + H - S] = [10, 13]$. Because $13 < 20$, we know that the temporal constraint is satisfied on path $\{t_{21}(t_{11}), t_{26}, t'_{21}, t_{22}, t_{23}(t_{11}(t_{21}), t_{12}, t_{16}), t_{24}, t_{25}\}$ according to condition A case 2 in Algorithm 4.

   Hence, the temporal constraint $D_R(t_{23}, t_{25}) \leq 20$ is satisfied at this checkpoint.

2) $D_R(t_{23}, t_{19}) \leq 16$. Because t_{23} is enabled at the fifth time unit, $S = 5$. We take the Time_set of node 10, i.e., {[10, 15], [17, 20]}. This means that the time spent on two paths $\{(t_{11}(t_{21}), t_{12}, t_{13}, t_{14}) \| (t_{11}(t_{21}), t_{12}, t_{13}, t_{15}), t_{17}(t_{21}(t_{11}), t_{26}, t'_{21}, t_{22}, t_{23}), t_{18}, t_{19}\}$ and $\{t_{11}(t_{21}), t_{12}, t_{16}(t_{21}(t_{11}), t_{26}, t'_{21}, t_{22}, t_{23}), t_{18}, t_{19}\}$ are at most 15 and 20 time units, respectively.

TABLE III
NODES OF UPDATED SPROUTING GRAPH

Node	Time sets	Path sets
1	∅	∅
2	{[0,0]}	{{t_{11}}}
3	{[0,1]}	{{$t_{11}(t_{21})$,t_{12}}}
4	{[0,1]}	{{$t_{11}(t_{21})$,t_{12},t_{13}}}
5	{[0,1]}	{{$t_{11}(t_{21})$,t_{12},t_{13}}}
6	{[1,3]}	{{$t_{11}(t_{21})$,t_{12},t_{13},t_{14}}}
7	{[2,4]}	{{$t_{11}(t_{21})$,t_{12},t_{13},t_{15}}}
8	{[2,4], [5,6]}	{{($t_{11}(t_{21})$,t_{12},t_{13},t_{14}) ‖ ($t_{11}(t_{21})$,t_{12},t_{13},t_{15}),t_{17}}, {$t_{11}(t_{21})$,t_{12},t_{16}}}
9	{[6,9], [13,14]}	{{($t_{11}(t_{21})$,t_{12},t_{13},t_{14}) ‖ ($t_{11}(t_{21})$,t_{12},t_{13},t_{15}),t_{17}($t_{21}(t_{11})$,t_{26}, t'_{21}, t_{22},t_{23}), t_{18}}, {$t_{11}(t_{21})$,t_{12}, t_{16}($t_{21}(t_{11})$,t_{26}, t'_{21},t_{22}, t_{23}), t_{18}}}
10	{[10,15], [17,20]}	{{($t_{11}(t_{21})$,t_{12},t_{13},t_{14}) ‖ ($t_{11}(t_{21})$,t_{12},t_{13},t_{15}),t_{17}($t_{21}(t_{11})$,t_{26}, t'_{21},t_{22},t_{23}), t_{18},t_{19}}, {$t_{11}(t_{21})$,t_{12},t_{16}($t_{21}(t_{11})$,t_{26}, t'_{21},t_{22}, t_{23}),t_{18},t_{19}}}
11	∅	∅
12	{[0,0]}	{{t_{21}}}
13	{[0,0]}	{{$t_{21}(t_{11})$,t_{26}}}
14	{[0,0]}	{{$t_{21}(t_{11})$,t_{26}, t'_{21}}}
15	{[0,0]}	{{$t_{21}(t_{11})$,t_{26}, t'_{21},t_{22}}}
16	{[5,5]}	{{ $t_{21}(t_{11})$,t_{26}, t'_{21},t_{22},t_{23}}}
17	{[9,13], [8,9]}	{{$t_{21}(t_{11})$,t_{26}, t'_{21},t_{22},t_{23}(($t_{11}(t_{21})$,t_{12},t_{13},t_{14}) ‖ ($t_{11}(t_{21})$,t_{12}, t_{13}, t_{15}),t_{17}),t_{24}}, {$t_{21}(t_{11})$,t_{26}, t'_{21},t_{22},t_{23}($t_{11}(t_{21})$,t_{12}, t_{16}), t_{24}}}
18	{[11,17], [10,13]}	{{$t_{21}(t_{11})$,t_{26}, t'_{21}, t_{22},t_{23}(($t_{11}(t_{21})$,t_{12},t_{13},t_{14}) ‖ ($t_{11}(t_{21})$,t_{12}, t_{13},t_{15}), t_{17}),t_{24},t_{25}},{ $t_{21}(t_{11})$,t_{26}, t'_{21},t_{22},t_{23}($t_{11}(t_{21})$,t_{12}, t_{16}), t_{24}, t_{25}}}

For [10, 15], we have $X = s = 16$ and $H = S = 5$. Then, $Y = [10+H-S, 15+H-S] = [10, 15]$. Because $15 < 16$, we know that the temporal constraint is satisfied on path $\{(t_{11}(t_{21}), t_{12}, t_{13}, t_{14}) \| (t_{11}(t_{21}), t_{12}, t_{13}, t_{15}), t_{17}(t_{21}(t_{11}), t_{26}, t'_{21}, t_{22}, t_{23}), t_{18}, t_{19}\}$ according to condition A case 2 in Algorithm 4.

For [17, 20], we have $Y = [17+H-S, 20+H-S] = [17, 20]$. Because $16 < 17$, the temporal constraint is violated on path $\{t_{11}(t_{21}), t_{12}, t_{16}(t_{21}(t_{11}), t_{26}, t'_{21}, t_{22}, t_{23}), t_{18}, t_{19}\}$ by condition A case 1 in Algorithm 4.

Because two transitions t_{23} and t_{19} are not in the same process and t_{23} has been enabled at this checkpoint, we return the erroneous path $\{t_{11}(t_{21}), t_{12}, t_{16}(t_{21}(t_{11}), t_{26}, t'_{21}, t_{22}, t_{23}), t_{18}, t_{19}\}$ according to step 4 of Algorithm 4.

F. Solution to Temporal Violation

The temporal constraint $D_R(t_{23}, t_{19}) \leq 16$ is violated at the fifth time unit, and the erroneous path is $\{t_{11}(t_{21}), t_{12}, t_{16}(t_{21}(t_{11}), t_{26}, t'_{21}, t_{22}, t_{23}), t_{18}, t_{19}\}$. Then, we follow our approach to solve the problem.

Since t_{23} has been enabled at this checkpoint and t_{23} is not in the same workflow process with t_{19}, according to the solution mentioned in case 1 condition B in Section V, we can reduce the time intervals of transitions before t_{19} in path $\{t_{11}(t_{21}), t_{12}, t_{16}(t_{21}(t_{11}), t_{26}, t'_{21}, t_{22}, t_{23}), t_{18}, t_{19}\}$, which has not been fired or enabled at this checkpoint. Because t_{11} and t_{12} have been enabled, the designer can reduce the time interval of t_{16}, t_{18}, and t_{19}.

We know that [17, 20] is responding to the erroneous path. First, because $H = 5$, $S = 5$, and $s = 16$, we have $Dr = b + H - S - s = 20 + 5 - 5 - 16 = 4$. Then, we consider the resource constraint with transition t_{24} and set $Dur = Dr + u_{t24} = 4 + 4 = 8$ as the minimal time value to be shortened for the duration modification operation. Here, $u_{t24} = 4$ is the maximum firing time of t_{24} at the worst case. Because the original time intervals of t_{16}, t_{18}, and t_{19} are [5, 5], [4, 5], and [4, 6], respectively, we can formalize the problem as

$$x_{16} + x_{18} + x_{19} = 8$$

s.t.

$$5 - x_{16} \geqq 0$$
$$4 - x_{18} \geqq 0$$
$$4 - x_{19} \geqq 0$$
$$x_{16} \geqq 0, x_{18} \geqq 0\ x_{19} \geqq 0.$$

$x_{16} = 3$, $x_{18} = 2$, and $x_{19} = 3$ are a solution. That is to say, if we reduce the time interval of t_{16}, t_{18}, and t_{19} by three, two, and two, respectively, then we can meet the temporal constraint. Finally, we reset the time interval of t_{16}, t_{18}, and t_{19} as $[5-3=2, 5-3=2]$, $[4-2=2, 5-2=3]$, and $[4-3=1, 6-3=3]$, respectively.

We can prove that our duration modification operation does not affect another temporal constraint $D_R(t_{23}, t_{25}) \leq 20$. Hence, our solution "resetting time interval of t_{16}, t_{18}, and t_{19} as [2, 2], [2, 3], and [1, 3]" is a correct solution.

G. Validation

Model checking is a widely used technique for verifying finite-state concurrent systems such as sequential circuit designs and communication protocols. It has a number of advantages over traditional approaches that are based on simulation, testing, and deductive reasoning.

Uppaal [29] is an integrated tool for model checking of real-time systems, and its typical application areas include real-time controllers and communication protocols, those where timing aspects are critical. Recently, there are some papers that discuss how to use Uppaal to check the time aspects or constraints in processes [30]–[33]. Here, we use it to validate our approach.

First, we transform the TWF-net model to equivalent timed automata (TA) model [33]. For a transition, we define a TA with two locations *disabled* and *enabled*, one local clock x, and some integer parameters. In addition, the edges between *disabled* and *enabled* with guard conditions represent the firing of transition. For example, transition t_{11} is transformed into the TA as shown in Fig 17(a).

Second, we construct two TA observers in which the temporal constraints are denoted as the guard conditions labeled with the corresponding edges, i.e., Observer1 and Observer2 for checking the temporal constraints $D_R(t_{23}, t_{25}) \leq 20$ and $D_R(t_{23}, t_{19}) \leq 16$, respectively, as shown in Fig. 17(b) and (c).

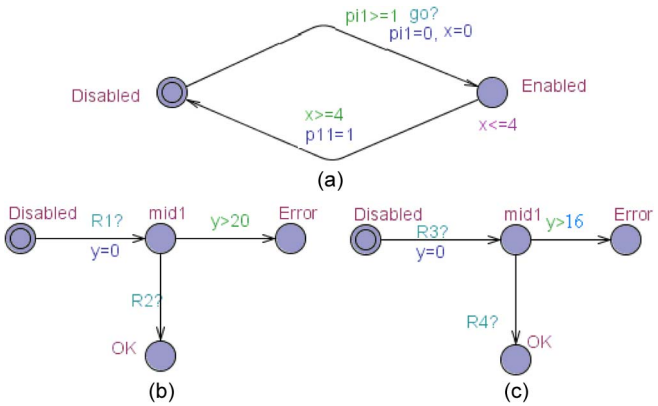

Fig. 17. Translated TA models in Uppaal.

The two temporal constraints are specified as the following queries of TA models:

A[] not Observer1.Error /*Temporal constraint 1*/

A[] not Observer2.Error /*Temporal constraint 2*/

Finally, both the transformed TA models and queries for temporal constraints are input into the model checking engine of Uppaal.

With the original time intervals, Uppaal returns "*Property is satisfied*" for the first query, which indicates that temporal constraint $D_R(t_{23}, t_{25}) \leq 20$ is not violated. Uppaal returns "*Property is not satisfied*" for the second query, and it indicates that the temporal constraint $D_R(t_{23}, t_{19}) \leq 16$ is violated. The returned results from Uppaal are consistent with the results from our approach which prove the correctness of our checking results.

After we reset the time interval of t_{16}, t_{18}, and t_{19} as [2, 2], [2, 3], and [1, 3], for both of the two queries of temporal constraints, Uppaal returns "*Property is satisfied*." This means that, by using our solution, the temporal constraint $D_R(t_{23}, t_{19}) \leq 16$ is satisfied again. The results from Uppaal prove the correctness of our solution.

Note that, compared with the existing methods of Uppaal, the advantages of our approach are as follows: 1) It can give erroneous path information in the workflow processes where the temporal violation happens, and 2) it can give solution to the temporal violation based on the erroneous paths.

VII. Related Work

This section gives an overview of current work in this research area and makes comparisons.

A. Static Checking of Temporal Constraints

Based on a graph-based workflow model, Eder *et al.* [5] develop a timed graph which shows the working duration with the earliest time and the latest finish time. Then, the algorithms to build the timed graph and to detect violation of timing constraints are constructed. Marjanovic and Orlowska [6] present a general classification and formal modeling of temporal constraints in workflows, and a polynomial algorithm to compute longest/shortest workflow instances. Zhuge *et al.* [7] propose a timed workflow process model through incorporating the time constraints, the duration of activities, the duration of flow, and the activity distribution with respect to the multiple time axes into the conventional workflow processes. Adam *et al.* [8] take timing constraints as external conditions of a workflow to analyze structural correctness of a PN-based workflow model. Han *et al.* [9] propose the idea of probabilistic time constraint WF-net and attempt to analyze successful execution ratios of activities, subprocesses, and the whole process at build time.

The main drawback of static checking is that it cannot make a comprehensive consideration of all the situations during the run time [3], [4].

B. Dynamic Checking of Temporal Constraints

Eder *et al.* [10] use the modified critical path method to calculate temporal constraints at run time. Marjanovic describes two concepts of time visualizations [11], i.e., the duration space and the instantiation space for the representation of relative and absolute time, respectively. Then, he presents the procedure of dynamic verification of temporal constraint consistency. Chen *et al.* [12] present a new run-time checkpoint selection strategy based on the time intervals of activities along the workflow execution path. Dynamic verification method is also developed, which makes the temporal verification more efficient by using previous verification results and avoiding unnecessary temporal verification. Chen and Yang [13] analyze the dependence between multiple fixed-date temporal constraints and its impact on the temporal verification. Furthermore, some dynamic temporal verification methods and algorithms are developed. Based on a PN-extended stochastic model, Han *et al.* [14] propose a dynamic approach for analyzing time constraints during process execution, whenever an activity instance is completed.

The aforementioned works assume that there is only one single workflow process and there is no resource constraint. However, multiple workflow processes may execute concurrently under resource constraints in a WfMS in practice. Moreover, their work can only answer whether there is temporal violation. They do not provide the erroneous violation paths or a concrete solution. Although Li and Yang [15] take resource constraints into consideration, their method neither provides the violation path information nor specific solutions. Compared with previous works, our approach can perform the following: 1) It can give the answer whether or not there is temporal violation; 2) it can provide the erroneous violation paths; and 3) it can propose a concrete solution.

VIII. Conclusion

Dynamic checking of temporal constraints of workflow processes is very important for workflow management. Although existing methods can give answers to whether there are any temporal violations, they neither provide the violation path information nor specific solutions.

In this paper, based on a sprouting graph of TWF-nets, we have presented a dynamic checking approach of temporal constraints for concurrent workflow processes with resource constraints. First, we construct sprouting graph models for multiple workflow processes. Second, we update the sprouting

graph at different checking points and check the temporal constraints. Finally, and most importantly, the violation paths and solutions are given. Moreover, we use Uppaal to verify the correctness of our approach, and we also use a concrete example to prove the usability and scalability of our approach.

In the future, we would like to extend our approach in the following aspects.

1) The complexity of constructing the sprouting graphs might be bad when it is used in large-scale workflows with lots of resource constraints. In the future, we plan to use net reducing technology [22], [23] and component-based technology [7] to reduce the complexity.
2) Our solution may cause a series of changes and make some satisfied temporal constraints at checkpoint H be violated [34]. If there is a conflict among temporal constraints, there is no solution. In the future, we will try to find other method to deal with these cases, which is based on modifying the structure of TWF-nets, e.g., inserting/removing activities, adding/ removing resources, etc. [34].

REFERENCES

[1] S. Jablonski and C. Bussler, *Workflow Management: Modeling Concepts, Architecture, and Implementation*. London, U.K.: Int. Thomson Computer Press, 1996.

[2] Y. S. Fan, *Fundamentals of Workflow Management Technology*. Beijing, China: Tsinghua Univ. Press, 2001.

[3] Z. Luo, A. Sheth, K. Kochut, and J. Miller, "Exception handling in workflow systems, applied intelligence," *Int. J. Appl. Intell., Neural Netw. Complex Problem-Solving Technol.*, vol. 13, no. 2, pp. 125–147, Sep./Oct. 2000.

[4] S. Yoonki and H. Dongsoo, "Exception specification and handling in workflow systems," in *Proc. 5th Asia-Pacific Web Conf. Web Technol. Appl.*, 2003, pp. 594–601.

[5] J. Eder, E. Panagos, and H. Pozewaunig, "Time management in workflow systems," in *Proc. 8th Int. Conf. Business Inf. Syst.*, 1999, pp. 265–280.

[6] O. Marjanovic and M. E. Orlowska, "Modeling and verification of temporal constraints in production workflows," *Knowl. Inf. Syst.*, vol. 1, no. 2, pp. 157–192, May 1999.

[7] H. Zhuge, T. Cheung, and H. Pung, "A timed workflow process model," *J. Syst. Softw.*, vol. 55, no. 3, pp. 231–243, Jan. 2001.

[8] N. R. Adam, V. Atluri, and W. K. Huang, "Modeling and analysis of workflow using Petri nets," *J. Intell. Inf. Syst.: Special Issue Workflow Process*, vol. 10, no. 2, pp. 131–158, Mar./Apr. 1998.

[9] R. Han, Y. Liu, L. Wen, and J. Wang, "Probability timing constraint WF-nets and their application to timing schedulability analysis of workflow management systems," in *Proc. Int. Conf. CSIE*, 2009, pp. 669–673.

[10] J. Eder, E. Panagos, and M. Abinovich, "Time constraints in workflow systems," in *Proc. 11th Conf. Adv. Inf. Syst. Eng.*, 1999, pp. 286–300.

[11] O. Marjanovic, "Dynamic verification of temporal constraints in production workflows," in *Proc. 11th Australian Database Conf.*, 2000, pp. 74–81.

[12] J. J. Chen, Y. Yang, and T. Y. Chen, "Dynamic verification of temporal constraints on-the-fly for workflow systems," in *Proc. 11th Asia-Pacific Softw. Eng. Conf.*, 2004, pp. 30–37.

[13] J. J. Chen and Y. Yang, "Temporal dependency for dynamic verification of fixed-date constraints in grid workflow systems," in *Proc. 7th APWeb Conf.*, 2005, pp. 820–831.

[14] R. Han, Y. Liu, L. Wen, and J. Wang, "Dynamically analyzing time constraints in workflow systems with fixed-date constraint," in *Proc. Int. Conf. APWeb*, 2010, pp. 99–105.

[15] H. C. Li and Y. Yang, "Dynamic checking of temporal constraints for concurrent workflows," *Electron. Commerce Res. Appl.*, vol. 4, no. 2, pp. 124–142, 2005.

[16] W. T. Jong, Y. S. Shiau, Y. J. Horng, H. H. Chen, and S. M. Chen, "Temporal knowledge representation and reasoning techniques using time petri nets," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 29, no. 4, pp. 541–545, Aug. 1999.

[17] J. Q. Li, Y. S. Fan, and M. C. Zhou, "Performance modeling and analysis of workflow," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 34, no. 2, pp. 229–242, Mar. 2004.

[18] P. C. Xiong, Y. S. Fan, and M. C. Zhou, "A Petri net approach to analysis and composition of web services," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 40, no. 2, pp. 376–387, Mar. 2010.

[19] P. C. Xiong, M. C. Zhou, and C. Pu, "A Petri net siphon based solution to protocol-level service composition mismatches," in *Proc. ICWS*, 2009, pp. 952–958.

[20] P. C. Xiong, C. Pu, and M. C. Zhou, "Protocol-level service composition mismatches: A Petri net siphon based solution," *Int. J. Web Serv. Res.*, vol. 7, no. 4, pp. 1–20, 2010.

[21] S. Ling and H. Schmidt, "Time petri nets for workflow modeling and analysis," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2000, pp. 3039–3044.

[22] Y. Qu and C. Lin, "Linear temporal inference of workflow management systems based on time Petri nets models," in *Proc. Int. Conf. EDCIS*, 2002, pp. 30–44.

[23] C. Lin and Y. Qu, "Temporal inference of workflow systems based on time Petri nets: Quantitative and qualitative analysis," *Int. J. Intell. Syst.*, vol. 19, no. 5, pp. 417–442, May 2004.

[24] J. P. Tsai and S. J. Yang, "Timing constraint Petri nets and their application to schedulability analysis of real-time system specification," *IEEE Trans. Softw. Eng.*, vol. 21, no. 1, pp. 32–49, Jan. 1995.

[25] J. Q. Li, Y. S. Fan, and M. C. Zhou, "Timing constraint workflow nets for workflow analysis," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 33, no. 2, pp. 179–193, Mar. 2003.

[26] H. Q. Wang and Q. T. Zeng, "Modeling and analysis for workflow constrained by resources and nondetermined time: An approach based on Petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 38, no. 4, pp. 802–817, Jul. 2008.

[27] S. Rajakumar, V. P. Arunachalam, and V. Selladurai, "Workflow balancing strategies in parallel machine scheduling," *Int. J. Adv. Manuf. Technol.*, vol. 23, no. 5/6, pp. 366–374, Mar. 2004.

[28] J. Ezpeleta, J. M. Colom, and J. Martinez, "A petri net based deadlock prevention policy for flexible manufacturing systems," *IEEE Trans. Robot. Autom.*, vol. 11, no. 2, pp. 173–184, Apr. 1995.

[29] Uppaal, Uppaal, 2010. [Online]. Available: http://www.it.uu.se/research/group/darts/uppaal/

[30] V. Gruhn and R. Laue, "Using timed model checking for verifying workflows," in *Proc. Int. Conf. CSAC*, 2005, pp. 75–88.

[31] F. Cassez1 and O. H. Roux, "Structural translation from time Petri nets to timed automata," *Electron. Notes Theoretical Comput. Sci.*, vol. 128, no. 6, pp. 145–160, May 2005.

[32] D. Maria, A. Montanari, and M. Zantoni, "An automaton-based approach to the verification of timed workflow schemas," in *Proc. Int. Conf.Temporal Representation and Reasoning (TIME)*, 2006, pp. 87–94.

[33] Z. H. Gu and G. S. Kang, "Analysis of event-driven real-time systems with time Petri nets: A translation-based approach," in *Proc. IFIP 17th World Comput. Congr.—TC10 Stream DIPES*, 2002, pp. 31–40.

[34] H. J. Hsu and F. J. Wang, "An incremental analysis for resource conflicts to workflow specifications," *J. Syst. Softw.*, vol. 81, no. 10, pp. 1770–1783, Oct. 2008.

YanHua Du received the B.S. and M.S. degrees in computer science from Zhengzhou University, Zhengzhou, China, in 2000 and 2003, respectively, and the Ph.D. degree in transportation planning and management from the China Academy of Railway Sciences, Beijing, China, in 2006.

He is currently a Lecturer with the School of Mechanical Engineering, University of Science and Technology Beijing, Beijing. He has published more than 20 research papers in journals and conferences. His research interests are in the areas of business process reengineering, workflow management, knowledge management systems, and Petri net modeling.

PengCheng Xiong received the B.S. degree from Huazhong University of Science and Technology, Wuhan, China, and the M.S. and Ph.D. degrees from Tsinghua University, Beijing, China.

He is currently a Research Assistant with the College of Computing, Georgia Institute of Technology, Atlanta. His research interests include Web service modeling and composition, Petri nets and applications, business process reengineering, and system integration.

YuShun Fan received the Ph.D. degree in control theory and application from Tsinghua University, Beijing, China, in 1990.

From September 1993 to 1995, he was a Visiting Scientist, supported by Alexander von Humboldt Stiftung, with the Fraunhofer Institute for Production System and Design Technology (FhG/IPK), Germany. He is currently a Professor with the Department of Automation, Director of the System Integration Institute, and Director of the Networking Manufacturing Laboratory, Tsinghua University. He has authored ten books in enterprise modeling, workflow technology, intelligent agent, object-oriented complex system analysis, and computer integrated manufacturing. He has published more than 300 research papers in journals and conferences. His research interests include enterprise modeling methods and optimization analysis, business process reengineering, workflow management, system integration and integrated platform, object-oriented technologies and flexible software systems, Petri nets modeling and analysis, and workshop management and control.

Xitong Li received the B.S. and Ph.D. degrees in control science and engineering from Tsinghua University, Beijing, China. He is currently working toward the Ph.D. degree in management science at the Information Technologies Group, MIT Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA.

His research interests include all aspects of effective management and use of data/information, e.g., data/information integration, economics of using data/information, data/ information quality, service-oriented computing, Web services, business process analysis, etc.