



Dynamic checking of temporal constraints for concurrent workflows [☆]

Hongchen Li ^{a,b}, Yun Yang ^{a,*}

^a CICEC-Centre for Internet Computing and E-Commerce, Faculty of Information and Communication Technologies, Swinburne University of Technology, PO Box 218, Hawthorn, Melbourne, Vic. 3122, Australia

^b School of Computer Science, Henan University, Kaifeng 475001, China

Received 21 October 2003; received in revised form 9 June 2004; accepted 5 September 2004

Available online 22 September 2004

Abstract

At present, workflow management systems have not sufficiently dealt with the issues of time, involving time modelling at build-time and time management at run-time. They are lack of the ability to support the checking of temporal constraints at run-time. Although some approaches have been devised to tackle this problem, they are limited to a single workflow and use only static techniques to verify temporal constraints. In reality, there are multiple workflows executing concurrently in a workflow management system. There may well exist resource constraints between these concurrent workflows, which affect significantly the verification of temporal constraints at run-time. This paper proposes a novel approach for dynamic verification of temporal constraints for concurrent workflows. We first investigate resource constraints in workflow management systems, and then define concurrent workflow executions. Based on these definitions, we propose a verification method by analysing the temporal relationship and resource constraints between activities among concurrent workflows.

© 2004 Elsevier B.V. All rights reserved.

Keywords: Concurrent workflows; Workflow specifications; Workflow verification; Temporal constraints; Resource constraints

1. Introduction

Workflow management concerns with coordination and control of business processes, which are abstractly and computerisedly represented as workflows, making use of information technology to reduce the costs and flow times and to increase the quality of service and productivity. A

[☆] A short and earlier version of this paper has been published in the proceedings of the sixth Asia-Pacific Web Conference (APWeb 2004), Springer Verlag, LNCS 3007, Hangzhou, China, April 14–17, 2004: 804–813 [11].

* Corresponding author. Tel.: +61 3 9214 8752; fax: +61 3 9819 0823.

E-mail addresses: lihch@founder.com (H. Li), yyang@it.swin.edu.au (Y. Yang).

workflow is a collection of tasks (or activities) organised to accomplish some business goal. Each of these tasks serves a given function in the overall process. Over the past decades, the design and development of Workflow Management Systems (WfMS) have emerged as an important area in both theory and practice [7,18].

One of the critical challenges for WfMSs is their ability to support the processing of time information, including time modelling in workflow specification (build-time) and time management in workflow execution (run-time) [12,13,5,20,4]. Typical examples are planning of workflow execution in time, estimating workflow execution durations and avoiding deadline violations. In reality, workflows execute along the time dimension. Therefore, the ability to deal with time information is of fundamental importance for workflow management because it is an intrinsic property of business processes in the real world.

Today's WfMSs are insufficient for processing time information although a few of them have taken into account this problem, such as ADEPT [15] and WIDE [3]. They usually assume that actors¹ have enough time to execute their work and no deadline for a workflow. But this is not the case in the real-world processes. Recently, time modelling and time management have attracted increasing attention in the workflow community [14,20,21,2]. The existing approaches address this problem, precisely speaking, by checking temporal constraints limited to a single workflow, while, in fact, there exist multiple workflows executing concurrently in a WfMS. These concurrent workflows may be correlated due to the fact that two activities from different workflows access the same resources in their executions. In a WfMS, some resources can be shared among different activities. At the start of an activity's execution, it must obtain the required resources. During its execution, some resources are assumed to be occupied exclusively by the activity. After the completion, these resources are released and can be accessed by other

activities. However, the relationship between concurrent workflows caused by resource access has been omitted in the existing approaches on checking temporal constraints. So, we believe the verification results may be insufficient. Consider a scenario where an activity in a workflow tries to access a resource that is being occupied by another activity in a different workflow and cannot be shared between them at the same time. The activity had to wait for this resource until the completion of the other activity. Therefore, the verification of temporal constraints on the former workflow should take into account the activity's waiting time.

In [10], we presented our work on the verification of resource consistency for a workflow specification at build-time. Based on the analysis of resource constraints in that paper, this paper proposes an approach for dynamic checking of temporal constraints on workflows in an environment, where there are multiple workflows executing concurrently. At build-time, we employ the same method as that in [10,13,21] to model time information into workflow specifications. At run-time, however, the proposed verification method is developed by analysing the temporal relationship and resource constraints between activities among concurrent workflows. Compared with the previous approaches, the verification method reported in this paper is more accurate and, hence, more useful in practice.

1.1. Problem statement: an informal description

The primary goal of introducing time management into WfMSs is to perform workflows effectively and efficiently under the restriction of time and resources and the avoidance of time failures [14]. To this end, time information (including temporal constraints) is usually incorporated into workflow specifications at build-time. Then workflow enactment components can check these temporal constraints at run-time based on the time information and workflow states. By doing so, WfMSs can control and coordinate workflows more effectively and efficiently. Therefore, the

¹ An actor can be either a person or a software agent, who acts a role to perform an activity or be responsible for its execution [6].

ability to check temporal constraints dynamically becomes very important to a WfMS.

Based on the previous discussion, we claim that the static checking method limited to a single workflow is too weak to verify temporal constraints sufficiently in an environment with concurrent workflows. A more complex approach should be introduced by taking into account the relationship between concurrent workflows. Therefore, the problem to be tackled in this paper can be stated informally as follows: *Given a set of workflows executing concurrently and a temporal constraint on one of them, our task is to check if the temporal constraint can be satisfied at a given time point.*

In WfMSs, after the above checking algorithm has been derived, the dynamic checking for temporal constraints can be realised by invoking this algorithm at any selected check point.

1.2. Outline

The rest of this paper is organised as follows. The next section introduces some basic concepts and notations we shall use later. Section 3 discusses resource constraints in WfMSs. Section 4 defines some concepts on concurrent workflow execution. The proposed verification method for temporal constraints is addressed in detail in Section 5. Section 6 illustrates a practical application of the proposed approach with an example. Section 7 presents the related work on the verification of temporal constraints in WfMSs. Finally, Section 8 draws concluding remarks.

2. Preliminaries

2.1. Workflow specification

Conceptually, a workflow is a collection of activities, together with their partial order of invocation and information flow. An activity is an application-specific unit scheduled by a WfMS. It can be defined as an entity with some attributes, such as input data, output data, actors, and states [1]. Basically, activities are classified as two types, namely, atomic activity and composite activity. An atomic activity cannot be divided further and

can be directly executed by a workflow engine;² while a composite activity is an abstract description of another process and can be decomposed into a workflow [9]. Sometimes, we do not differentiate them and call them activities only if this does not cause ambiguity for understanding. Dependencies between activities define their execution orders in a workflow. The execution orders compose the control structure of the workflow. Four kinds of basic control structures, namely, sequential, parallel, selective and iterative structures, have been defined in the Workflow Reference Model [9].

Usually, business processes are specified to be workflow specifications according to specific syntax rules. An activity is denoted by a node (called activity node). Moreover, four types of nodes (called control nodes), namely and-split (*as*), and-join (*aj*), or-split (*os*) and or-join (*oj*), are introduced for the representation of the above control structures [9]. Formally, a workflow specification can be defined as follows:

Definition 2.1. (Workflow specification) A workflow specification, ws , is abstracted as a 3-tuple $\langle N, F, \mathfrak{R} \rangle$, where (i) $N = \{n_1, n_2, \dots, n_t\}$ is a union of a set of activity nodes $AN = \{a_1, a_2, \dots, a_n\}$ and a set of control nodes $CN = \{cn_1, cn_2, \dots, cn_m\}$. Each element in CN has one of the above four types, that is, *as*, *aj*, *os*, or *oj*. (ii) $F \subset N \times N$ is a set of flows between these nodes. (iii) $\mathfrak{R} : AN \rightarrow R$ is a resource set accessed by an activity, where $R = \{R_1, R_2, \dots, R_n\}$ is a superset. R_i , $i = 1, \dots, n$, is a resource set accessed by a_i . (iv) ws has a unique start activity (denoted as a_s) and at least one end activity (denoted as a_e).

To visualise a workflow specification, we can use a directed acyclic graph to represent its process structure (see Fig. 1). The meanings of all symbols are annotated in Fig. 1, where small squares represent activity nodes involved in a workflow, small circles and diamonds denoting control nodes in the workflow, arrows denoting flows between those nodes, c_1 and c_2 attached to some arrows indicating conditions on the flows, and letters in a pair of braces below an activity denoting resources

² A workflow engine is a software service that provides the run-time execution environment for a workflow [9].

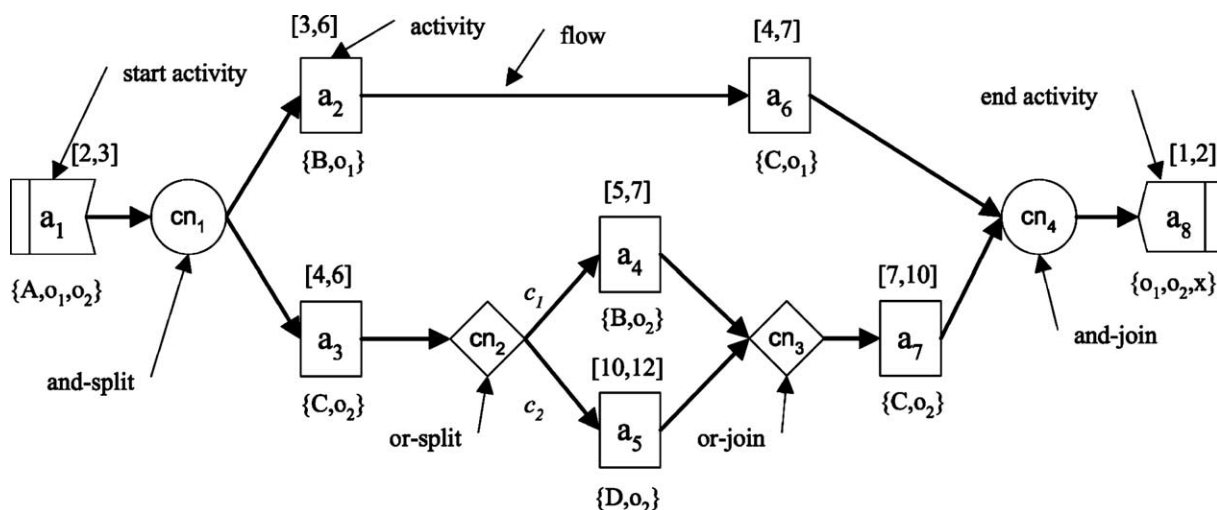


Fig. 1. Graphic representation of a workflow specification.

accessed by the activity. Usually, to simplify the analysis work on a complex workflow specification, it is possible to encapsulate a part at one level into a composite activity at another level. Accordingly, the resource set accessed by this composite activity is composed of all resources accessed by those activities within this composite activity. The temporal constraints on them can be transformed into those on the composite activity.

In order to model time information of activities into a workflow, we need to augment each of them with two time values, namely, the minimum and maximum durations. So, at build-time, we define $d(a)$ and $D(a)$, where $d(a) < D(a)$, as the minimum and maximum durations of activity a for its execution, respectively [21]. Time is expressed in some basic time units, such as minutes, hours, or days. The granularity is selected according to specific workflow applications. As shown in Fig. 1, two numbers in a pair of square brackets above an activity denote the minimum (left) and maximum (right) durations respectively. For example, activity a_2 needs at least 3 time units and 6 time units at most for its execution at run-time.

2.2. Workflow execution

In reality, a workflow is an execution case of a workflow specification, beginning at the start

activity and ending at the end activity. Each workflow is assumed to have an identifier, which distinguishes it from others. For a workflow specification, each execution case corresponds to a unique workflow. Similarly, each execution case of an activity corresponds to a unique activity instance with an identifier in the workflow. The detailed discussion about state transitions of a workflow and an activity can be referred to [9].

Note that a workflow contains a subset of activities explicitly specified in the associated workflow specification. During a workflow execution, activities are scheduled with respect to the flows which prescribe the precedence relationship between these activities. This is to say, given a workflow specification ws with $\langle n_i, n_j \rangle \in F$, if both n_i and n_j are scheduled in a workflow, n_j must start to execute only after the completion of n_i , denoted as $n_i < n_j$. More formally

Definition 2.2. (Workflow) A workflow, w , is a 4-tuple $\langle id, ws, A, < \rangle$, reflecting the execution of a workflow specification, where (i) id is an identifier assigned to the workflow. (ii) ws is the associated workflow specification. (iii) $A \subseteq N$ is a set of activities (include control nodes), which contains a subset of activities in ws . (iv) The execution order $< \subseteq (A \times A)$ is the partial order such that if $a_i, a_j \in A$ and $\langle a_i, a_j \rangle \in F$ in ws , then $a_i < a_j$.

Note that the execution order $<$ is transitive. That is to say, let $a_i, a_j, a_k \in A$ be three activities in workflow w , if there exist $a_i < a_j$ and $a_j < a_k$, then $a_i < a_k$. Actually, in addition to the regular activities, a workflow may contain other activities, for example, compensating activities, whose function is to undo the effect of an execution of those regular activities. We omit them in this paper because of being less related to the topic. Schuld et al. [17] and Grefen et al. [8] investigate into more depth on addressing compensating activities and their relationship with regular activities.

At run-time, a completed activity a has a start time (denoted as $S(a)$) and an end time (denoted as $E(a)$). It is active during the period from its start time to end time. This period is called the *active interval* of activity a , denoted as $[S(a), E(a)]$. $D_R(a) = E(a) - S(a)$ is defined as its *run-time duration*. Accordingly, $D_R(a_i, a_j)$ denotes the run-time duration from the start time of a_i to the end time of a_j , where $a_i < a_j$. Under the normal condition, we have:

- (1) $d(a) \leq D_R(a) \leq D(a)$,
- (2) $E(a_i) \leq S(a_j)$ if $\exists f = \langle a_i, a_j \rangle \in F$.

In case that an exception happens, the duration of the exception handling is classified as a part of the duration of the related activity according to the place where the exception happens.

2.3. Temporal constraint and its representation

Temporal constraints are classified into three kinds in [21], namely, the fixed-point constraint, the duration constraint and the interdependent constraint. In fact, the interdependent constraint is a combination of the first two. In order to simplify the verification work, we differentiate temporal constraints between two classes, namely absolute temporal constraints and relative temporal constraints. An *absolute temporal constraint* defines, in terms of absolute time, when an activity should start or complete during workflow execution. The deadline for submission of an application, for example, is December 15. Here, the date is an absolute time value. A *relative temporal con-*

straint defines when an activity should start or end relative to the start or end of another one. For instance, suppose $a_i < a_j$, activity a_j should complete no later than p time units after activity a_i ends. Here, p is a relative time value.

Temporal constraints in a workflow specification are consistent if and only if they could be satisfied based on the syntax of the workflow specification and the expected minimum and maximum durations of activities [13]. In other words, all activities are schedulable during workflow execution.

Here, we use $S(a_i) \leq_t t_i$ to represent that a_i should start to execute on or before absolute time t_i , and $E(a_i) \leq_t t_i$ to represent that a_i should end its execution on or before absolute time t_i . For the relative temporal constraints, we use $D_R(a_i, a_j) \leq p$ to denote that a_j should end its execution no more than p time units after a_i starts. Here, a_i is called a *reference point*.

In fact, absolute temporal constraints and relative temporal constraints can be transformed into each other. For example, a_k is selected as a reference point, suppose $a_k < a_i$ and $S(a_k) \leq_t t_k$, then, given an absolute temporal constraint $E(a_i) \leq_t t_i$, it can be transformed into a relative temporal constraint $D_R(a_k, a_i) \leq t_i - t_k$. On the contrary, given a relative temporal constraint $D_R(a_k, a_i) \leq p$, it can be transformed into an absolute temporal constraint $E(a_i) \leq_t t_k + p$.

Therefore, in this paper, we only demonstrate the dynamic checking of relative temporal constraints, and take $D_R(a_k, a_i) \leq p$ (a_k is a reference point) as the *canonical representation* for temporal constraints.

3. Resource constraints in WfMSs

3.1. Resource access in WfMSs

As stated earlier, activities in a workflow usually access resources during their executions. In WfMSs, a resource is defined to be any entity required by an activity for its execution, such as a document, a database table, an appliance (for example, printer), an application, or even an actor. According to the access property of resources in a

WfMS, they are classified as two types, namely, shared resources and private resources. Shared resources can be accessed by different activities within a workflow or from different workflows, while private resources cannot be shared between activities and are only accessed by an activity. So, it is unnecessary to involve the private resources in our verification because they do not give rise to resource constraints. For those shared resources, we assume that a locking mechanism is used to control the concurrent access of resources by activities. In this mechanism, activities acquire and release locks on resources in two different modes, namely, shared mode and exclusive mode. In the shared mode, an activity acquires a shared lock on a resource if this resource can be shared simultaneously by activities and the access does not change the state of the resource. On the other hand, in the exclusive mode, the activity acquires an exclusive lock on the resource. For an activity and a specific resource, which kind of lock will be issued on the resource and what policy is adopted to avoid deadlocks are beyond the scope of this paper and need further research in our ongoing work.

The compatibility table of shared and exclusive locks is illustrated in Table 1, where ‘+’ indicates that the locks are compatible and ‘-’ indicates the locks are in conflict. At the start of an activity’s execution, it must obtain the required resources by issuing locks on them. As indicated in Table 1, a shared lock followed by a shared locking request is compatible; then the locking request can be granted. The other three cases are in conflict. That is to say, a locking request will be delayed until the corresponding held lock is released by other activities.

Here, we focus on those resources with exclusive locks. According to the mechanism, the resources are assumed to be occupied exclusively

by those activities during their executions, and cannot be accessed by other activities until their completion.

3.2. Resource constraints between activities

Based on the above discussion, we claim that resource constraints exist between activities, which are implicit rules that influence the execution order of those activities and further the result of a workflow. In WfMSs, a resource can be denoted as r , with a unique identifier. All resources accessed by an activity a_i consist of a set $R_i = \{r_1, \dots, r_m\}$. The mapping function $\mathfrak{R}(a_i)$ (see Definition 2.1) returns all resources accessed by a_i , that is, $\mathfrak{R}(a_i) = R_i$. Here, we introduce some definitions on resource constraints.

Definition 3.1. (Resource dependency) Given two activities a_i and a_j , ($i \neq j$), within a workflow or from two different workflows, we say that a_i and a_j have a *resource dependency* if $\mathfrak{R}(a_i) \cap \mathfrak{R}(a_j) \neq \phi$.

If a_i and a_j have a resource dependency, a_i cannot execute simultaneously with a_j at run-time. Otherwise, a conflict may arise from the competition for the same resources. We call this kind of conflict as resource conflict. More formally:

Definition 3.2. (Resource conflict) Given a set of concurrent workflows, a_i has a *resource conflict* with a_j if they have a resource dependency and $([S(a_i), E(a_i)] \cap [S(a_j), E(a_j)]) \neq \phi$.

An example illustrating resource constraints between activities is given below.

Example 3.1. (Resource dependency and conflict) Fig. 2 shows two workflow specifications ws_1 and ws_2 and two concurrent workflows w_1 and w_2 associated with them, respectively. As shown in Fig. 2, a_{11} and a_{12} have a resource dependency because they access the same resource o_1 ; and a_{11} and a_{22} have a resource dependency due to resource A . Since there is a flow $\langle a_{11}, a_{12} \rangle$ in ws_1 , no resource conflict exists between a_{11} and a_{12} . However, suppose two workflows w_1 and w_2 execute concurrently as shown in Fig. 2, activities a_{11} and a_{22} will compete for the same resource A . This results

Table 1
The compatibility table of shared and exclusive locks

Held	Acquired	
	Shared	Exclusive
Shared	+	-
Exclusive	-	-

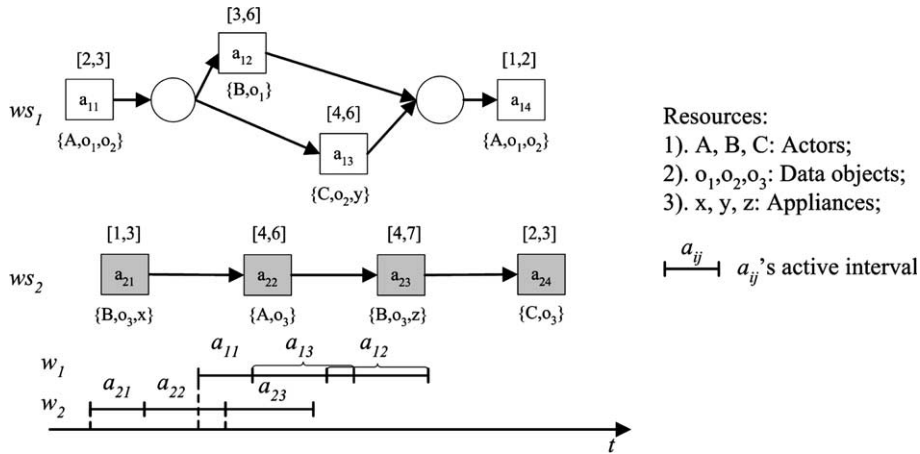


Fig. 2. Resource constraints between activities.

in a resource conflict because *A* cannot be shared between them at the same time.

In [10], we presented a method for identifying potential resource conflicts due to resource dependencies within a workflow specification. Here, we pay more attention to resource dependencies between activities among concurrent workflows. Given a set of concurrent workflows $W = \{w_1, w_2, \dots, w_m\}$, correlations may exist between them due to the resource dependency between activities. More formally:

Definition 3.3. (Correlation) Two workflows w_i and w_j ($i \neq j$), are correlated if and only if there are two activities $a_{ik} \in A_i$ and $a_{jl} \in A_j$ such that they have a resource dependency.

4. Concurrent workflow execution

In general, there are multiple workflows executing concurrently in a WfMS, each of which is controlled and coordinated by a workflow engine independently. For each workflow, we have the following assumptions.

Assumption 1. (Consistent specification) A workflow specification is correctly defined, without errors or inconsistencies as identified in [10,16].

Assumption 2. (Correct execution) Each activity in a workflow executes under the normal case. These mean that: (1) Each activity executes and commits correctly without exceptions. (2) For any activity $a \in A$, there is $d(a) \leq D_R(a) \leq D(a)$. (3) A workflow is also guaranteed to terminate correctly without exceptions.

For the sake of brevity, here we introduce a terminology, *workflow schedule*, to denote a set of concurrent workflows.

Definition 4.1. (Workflow schedule) A workflow schedule, S , is a 3-tuple $\langle W, A_S, \prec_S \rangle$, reflecting the concurrent execution of workflows, where: (i) $W = \{w_1, w_2, \dots, w_n\}$ is a set of workflows, where $w_i = \langle id_i, ws_i, A_i, \prec_i \rangle$. (ii) $A_S = \{a_{ij} | (a_{ij} \in A_i) \wedge (w_i \in W)\}$ is a set of activities of all workflows in W . (iii) \prec_S is a partial order between activities of A_S with $\prec_S \subseteq (A_S \times A_S)$ and $\prec_S = \cup_{w_i \in W} \prec_i$.

The definition of a workflow schedule reflects the concurrent execution of workflows at the activity level. Activities within a workflow are scheduled by workflow engine according to flows defined in the associated workflow specification. However, activities from different workflows have no such limitation. That is to say, given two activities $a_{ik} \in A_i$, $a_{jl} \in A_j$, ($i \neq j$), either $S(a_{ik}) \prec_t S(a_{jl})$ or $S(a_{jl}) \prec_t S(a_{ik})$. This results in

a correct workflow schedule. A workflow schedule is correct if it has no resource conflicts. More formally:

Definition 4.2. (Correct workflow schedule) A workflow schedule S is correct if for any two activities $a_{ik}, a_{jl} \in A_S$ with a resource dependency, we have $([S(a_{ik}), E(a_{ik})] \cap [S(a_{jl}), E(a_{jl})]) = \phi$. Therefore, when an activity to be scheduled is identified to be in conflict with another one, it will be postponed for execution or allowed to pre-empt the execution of the other one with a lower priority. Here, we assume that the First-Come-First-Served (FCFS) policy is applied to allocate resources to activities. That is to say, an activity with resource conflicts will be postponed for execution until all resources required are available.

Example 4.1. (Correct and incorrect workflow schedules) As an example, consider two workflows w_1 and w_2 being executed concurrently as described in Fig. 3(a), workflow schedule S_1 is correct because of no resource conflict. However, consider again two workflows w_3 and w_4 as described in Fig. 3(b), workflow schedule S_2 is incorrect because of resource conflicts between a_{11} and a_{22} and between a_{12} and a_{23} , respectively.

Proposition 4.1. In a correct workflow schedule S , for any two activities a_{ik} and a_{jl} with a resource dependency, if $S(a_{ik}) <_t S(a_{jl})$, then $a_{ik} < a_{jl}$.

Proof. Proof of the proposition follows from Definitions 3.1, 3.2, 4.1, 4.2. According to Definition 4.2, we have $([S(a_{ik}), E(a_{ik})] \cap [S(a_{jl}), E(a_{jl})]) = \phi$ for activities a_{ik} and a_{jl} because workflow schedule S is correct. Since $S(a_{ik}) <_t S(a_{jl})$, $a_{ik} < a_{jl}$ can be derived. \square

5. Dynamic verification of temporal constraints

So far, we have introduced some concepts and notations on workflow specifications and discussed resource constraints in WfMSs and concurrent workflow execution. In this section, we address the checking method for temporal constraints on workflows. The problem to be tackled can be formally stated as: given a workflow schedule $S = \langle W, A_S, <_S \rangle$ and a temporal constraint $D_R(a_{ik}, a_{il}) \leq p$, we check if the temporal constraint can be satisfied at reference point a_{ik} in the workflow schedule.

5.1. Estimated active intervals

To achieve the dynamic verification of temporal constraint $D_R(a_{ik}, a_{il}) \leq p$, where $a_{ik} < a_{il}$, for example, in an environment with concurrent workflows, we should calculate the value of $D_R(a_{ik}, a_{il})$, and then compare it with p . If the value of $D_R(a_{ik}, a_{il})$ is less than or equal to p , we say that the

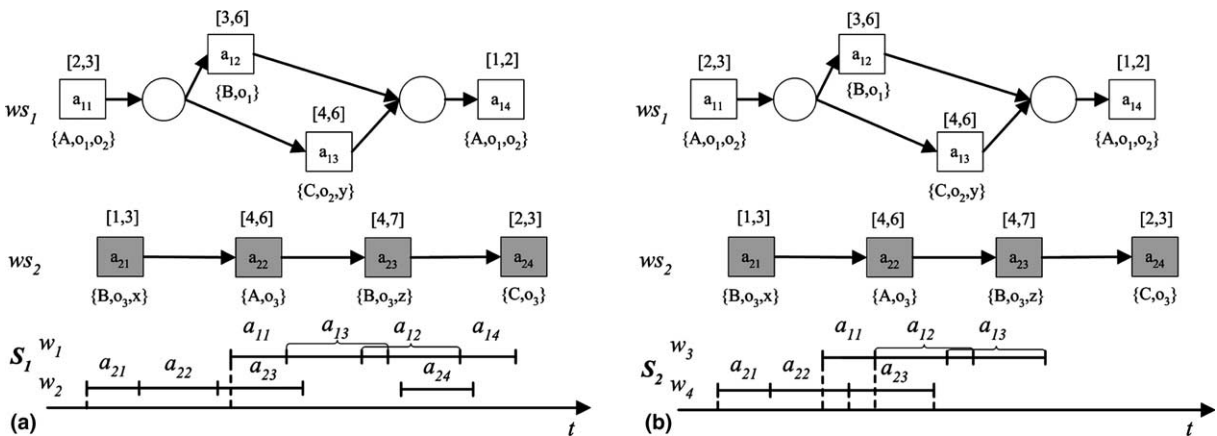


Fig. 3. Correct and incorrect workflow schedules.

temporal constraint can be satisfied. Otherwise, the result indicates that the temporal constraint might be violated. However, the value of $D_R(a_{ik}, a_{il})$ can be determined only if a_{ik} and a_{il} have completed. At that moment, if the temporal constraint is violated, then it is too late for any preventive or corrective action, and the checking makes no sense for workflow management. Therefore, it is necessary to dynamically check the temporal constraint before a_{il} starts to execute. To this end, we should estimate some activities' active intervals relative to the reference point.

Given a set of workflow specifications $WS = \{ws_1, ws_2, \dots, ws_m\}$ associated with a workflow schedule $S = \langle W, A_S, <_S \rangle$, each of which is defined in Definition 2.1 with time information, we have:

Definition 5.1. (Reachability) Node n_{il} is *reachable* from node n_{ik} if there is a path in ws_i consisting of a series of flows from n_{ik} to n_{il} .

Let $Reachable(n_{ik}, n_{il})$ be a Boolean function to denote the reachability from node n_{ik} to n_{il} such that

$$Reachable(n_{ik}, n_{il}) = \begin{cases} \text{True} & \text{if } (\exists path = \langle n_{ik}, \dots, n_{il} \rangle), \\ \text{False} & \text{otherwise.} \end{cases}$$

Note that all nodes reachable from n_{ik} form a *reachable set* of n_{ik} , denoted as $Re(n_{ik})$, which is a subset of N_i and can be calculated as follows:

$$Re(n_{ik}) = \{n | n \in N_i \wedge ((\langle n_{ik}, n \rangle \in F_i) \vee (\exists n_{il} \in N_i, n_{il} \in Re(n_{ik}) \wedge n \in Re(n_{il})))\}.$$

Here, the relation of *reachability* is assumed to be reflexive, that is, $n_{ik} \in Re(n_{ik})$.

Let a_{ik} be a reference point, for activity $a_{il} \in Re(a_{ik})$, we have:

Definition 5.2. (Earliest start time) The earliest start time of a_{il} , $EST(a_{il})$, is its start time relative to a_{ik} under the condition where $(\forall a \in AN_j, j = 1, \dots, m) \wedge (a_{ik} < a)$ such that $D_R(a) = d(a)$.

Definition 5.3. (Earliest end time) The earliest end time of a_{il} , $EET(a_{il})$, is its end time relative to a_{ik} under the condition where $(\forall a \in AN_j, j = 1, \dots, m) \wedge (a_{ik} < a)$ such that $D_R(a) = d(a)$.

Definition 5.4. (Latest end time) The latest end time of a_{il} , $LET(a_{il})$, is its end time relative to a_{ik} under the condition where $(\forall a \in AN_j, j = 1, \dots, m) \wedge (a_{ik} < a)$ such that $D_R(a) = D(a)$.

Definition 5.5. (Estimated active interval) The estimated active interval of a_{il} is defined as the period between $EST(a_{il})$ and $LET(a_{il})$, where $EST(a_{il}) < LET(a_{il})$, denoted as $[EST(a_{il}), LET(a_{il})]$.

The latest start time of an activity is not defined due to not being used in the paper. From the above definitions, we conclude that:

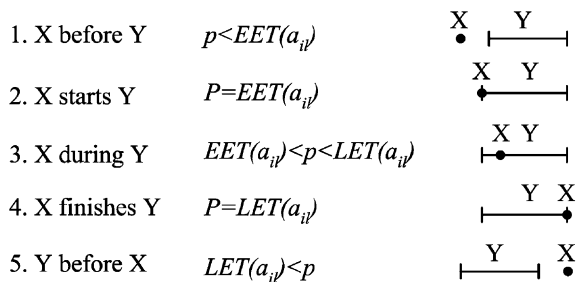
Proposition 5.1. $EET(a_{il}) = EST(a_{il}) + d(a_{il})$ for activity $a_{il} \in Re(a_{ik})$.

Proof. The proof follows directly from Definitions 5.2, 5.3 and 5.4. \square

Proposition 5.2. For any activity $a_{il} \in Re(a_{ik})$, the active interval $[S(a_{il}), E(a_{il})]$ is within the estimated active interval $[EST(a_{il}), LET(a_{il})]$ at reference point a_{ik} in workflow schedule S ,

that is, $[S(a_{il}), E(a_{il})] \subseteq [EST(a_{il}) + S(a_{ik}), LET(a_{il}) + S(a_{ik})]$.

Proof. Suppose that a_{ik} is being initiated in S at present with start time $S(a_{ik}) = t_i$, at that moment, a_{il} does not start to execute, and then the values of $S(a_{il})$ and $E(a_{il})$ are not available. However, they are composed of two parts: one is the available run-time t_i ; another is the remaining part that is unknown at present. For any activity $(a \in AN_j, j = 1, \dots, m) \wedge (a_{ik} < a)$, we have $d(a) \leq D_R(a) \leq D(a)$. From Definitions 5.2 and 5.4, we have $EST(a_{il}) + S(a_{ik}) \leq S(a_{il})$ and $E(a_{il}) \leq LET(a_{il}) + S(a_{ik})$. So, the proposition holds. \square



X is a point with $X=[p]$, Y is an interval with $Y=[EET(a_{ij}), LET(a_{ij})]$

Fig. 4. Temporal relations between p and $Y = [EET(a_{ij}), LET(a_{ij})]$.

Theorem 1. Given a correct workflow schedule $S = \langle W, A_S, \prec_S \rangle$, a reference point a_{ik} and a temporal constraint $D_R(a_{ik}, a_{il}) \leq p$ on w_i , where $a_{il} \in Re(a_{ik})$, we assert: (i) $D_R(a_{ik}, a_{il}) \leq p$ can be violated at a_{ik} if $p < EET(a_{il})$. (ii) $D_R(a_{ik}, a_{il}) \leq p$ can be satisfied at a_{ik} if $LET(a_{il}) \leq p$. (iii) $D_R(a_{ik}, a_{il}) \leq p$ cannot be decided at a_{ik} if $EET(a_{il}) \leq p < LET(a_{il})$.

Proof. Since a_{ik} is a reference point, following Propositions 5.1 and 5.2, we have $EET(a_{il}) + S(a_{ik}) \leq E(a_{il}) \leq LET(a_{il}) + S(a_{ik})$, that is, $EET(a_{il}) \leq D_R(a_{ik}, a_{il}) \leq LET(a_{il})$. Considering the single time line, the temporal relations between p and

interval $[EET(a_{il}), LET(a_{il})]$ can be described by Allen’s interval logic extended in [19] as shown in Fig. 4. The temporal relations presented in Fig. 4 are exclusive and exhaustive. We classify them into the following cases:

- Case 1: $p < EET(a_{il})$: corresponding to case ‘X before Y’ in Fig. 4, we have $p < D_R(a_{ik}, a_{il})$, then the temporal constraint can be violated at a_{ik} .
- Case 2: $LET(a_{il}) \leq p$: corresponding to cases ‘X finishes Y’ and ‘Y before X’ in Fig. 4, we have $D_R(a_{ik}, a_{il}) \leq p$, then the temporal constraint can be satisfied at a_{ik} .
- Case 3: $EET(a_{il}) \leq p < LET(a_{il})$: corresponding to cases ‘X starts Y’ and ‘X during Y’ in Fig. 4, then the relation between $D_R(a_{ik}, a_{il})$ and p cannot be decided at a_{ik} . \square

5.2. Calculation of estimated active intervals within a single workflow

Given a workflow specification ws_i and a reference point a_{ik} , for an activity $a_{il} \in Re(a_{ik})$, we can calculate its estimated active interval within this workflow specification. According to the definitions, we have $EST(a_{ik}) = 0$ and $LET(a_{ik}) = D(a_{ik})$. Now given the estimated active intervals of a_{ik} and

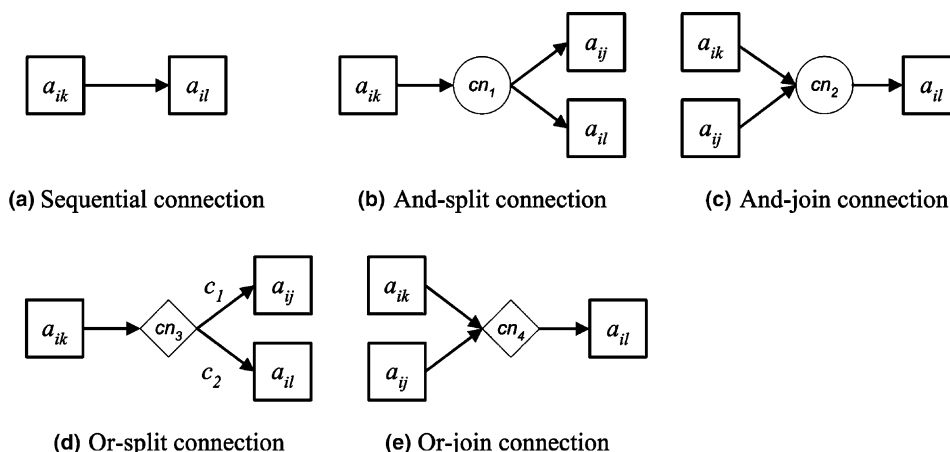


Fig. 5. Basic control structures in workflow specifications. (a) Sequential connection. (b) And-split connection. (c) And-join connection. (d) Or-split connection. (e) Or-join connection.

a_{ij} , a_{il} 's estimated active interval can be calculated with respect to the basic control structures respectively as detailed next.

5.2.1. Basic control structures

A sequential connection is defined as a segment of a workflow, where activities are executed in sequence. In a workflow specification, if there is a sequential connection between activities a_{ik} and a_{il} (see Fig. 5 (a), denoted as $a_{ik} \cdot a_{il}$), $EST(a_{il})$ and $LET(a_{il})$ can be calculated as follows:

$$EST(a_{il}) = EST(a_{ik}) + d(a_{ik}),$$

$$LET(a_{il}) = LET(a_{ik}) + D(a_{il}).$$

An and-split connection is defined as a single thread of control splitting into two or more parallel activities. As shown in Fig. 5 (b), activities a_{ij} and a_{il} are the and-split successors of activity a_{ik} , denoted as $a_{ik} \cdot (a_{ij} \wedge a_{il})$. The four time values can be calculated as follows:

$$EST(a_{ij}) = EST(a_{ik}) + d(a_{ik}),$$

$$EST(a_{il}) = EST(a_{ik}) + d(a_{ik}).$$

$$LET(a_{ij}) = LET(a_{ik}) + D(a_{ij}),$$

$$LET(a_{il}) = LET(a_{ik}) + D(a_{il}).$$

An and-join connection is defined as two or more parallel executing activities converging into a single common thread of control. Fig. 5 (c) shows an and-join connection, where activity a_{il} executes after the completion of a_{ik} and a_{ij} , denoted as $(a_{ik} \wedge a_{ij}) \cdot a_{il}$. $EST(a_{il})$ and $LET(a_{il})$ are calculated as follows:

$$EST(a_{il}) = \text{MAX}\{EST(a_{ik}) + d(a_{ik}), \\ EST(a_{ij}) + d(a_{ij})\},$$

$$LET(a_{il}) = \text{MAX}\{LET(a_{ik}), \\ LET(a_{ij})\} + D(a_{il}).$$

An or-split connection is defined as a single thread of control making a decision upon which branch to take when encountered with multiple threads of branches. As shown in Fig. 5 (d), activities a_{ij} and a_{il} are the or-split successors of a_{ik} , denoted as $a_{ik} \cdot (a_{ik} \vee a_{il})$. The four time values can be calculated as follows:

$$EST(a_{ij}) = EST(a_{ik}) + d(a_{ik}),$$

$$EST(a_{il}) = EST(a_{ik}) + d(a_{ik}),$$

$$LET(a_{ij}) = LET(a_{ik}) + D(a_{ij}),$$

$$LET(a_{il}) = LET(a_{ik}) + D(a_{il}).$$

An or-join connection is defined as two or more workflow branches re-converging into a single thread of control without any synchronisation. As shown in Fig. 5 (e), where activity a_{il} executes after the completion of a_{ik} or a_{ij} , denoted as $(a_{ik} \vee a_{ij}) \cdot a_{il}$. $EST(a_{il})$ and $LET(a_{il})$ are calculated as follows:

$$EST(a_{il}) = \text{MIN}\{EST(a_{ik}) + d(a_{ik}), \\ EST(a_{ij}) + d(a_{ij})\},$$

$$EST(a_{ij}) + d(a_{ij})\},$$

$$LET(a_{il}) = \text{MAX}\{LET(a_{ik}), \\ LET(a_{ij})\} + D(a_{il}).$$

$$LET(a_{ij})\} + D(a_{il}).$$

5.2.2. Parallel and selective connections

A parallel connection is defined as a segment of a workflow, where activities are executing in parallel and there are multiple threads of control. Referring to Fig. 6 (a), activities a_{ij} and a_{il} execute in

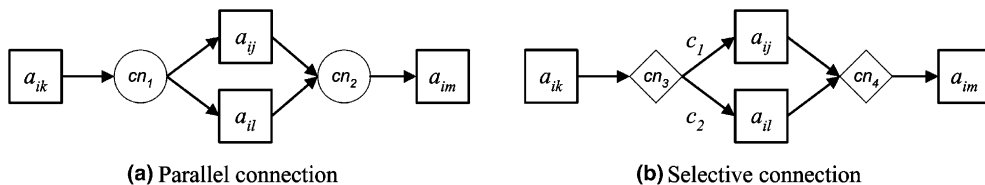


Fig. 6. Parallel (a) and selective (b) connections in workflow specifications.

parallel, denoted as $(a_{ij}||a_{il})$. $EST(a_{ik}||a_{il})$ and $LET(a_{ij}||a_{il})$ are calculated as follows:

$$EST(a_{ij}||a_{il}) = EST(a_{ik}) + d(a_{ik}),$$

$$LET(a_{ij}||a_{il}) = \text{MAX}\{LET(a_{ik}) + D(a_{ij}), LET(a_{ik}) + D(a_{il})\}.$$

A selective connection is defined as a segment of a workflow, where one thread of control is selected from multiple branches based on a condition. As shown in Fig. 6 (b), either activity a_{ij} or a_{il} is selected in a workflow, denoted as $(a_{ij}|a_{il})$. $EST(a_{ij}|a_{il})$ and $LET(a_{ij}|a_{il})$ are calculated as follows:

$$EST(a_{ij} | a_{il}) = EST(a_{ik}) + d(a_{ik}),$$

$$LET(a_{ij} | a_{il}) = \text{MAX}\{LET(a_{ik}) + D(a_{ij}), LET(a_{ik}) + D(a_{il})\}.$$

For a workflow specification composed of the above control structures, given a reference point a_{ik} , we can design an algorithm to calculate $[EST(a_{ij}), LET(a_{ij})]$ for each $a_{ij} \in Re(a_{ik})$. Here, the algorithm, being relatively straightforward, is omitted due to the space limit.

5.3. Adjusting estimated active intervals for activities with resource dependency

The calculation of estimated active intervals within a single workflow does not consider resource dependencies between activities among concurrent workflows in a workflow schedule. Hence, the values calculated are inconsistent with the case in an environment with concurrent workflows, and then need to be adjusted in terms of resource dependencies.

Given two concurrent workflows w_i and w_j , a_{ik} and a_{jm} are selected as two reference points. Now, we can calculate $[EST(a_{ip}), LET(a_{ip})]$ for each activity $a_{ip} \in Re(a_{ik})$ within w_i and $[EST(a_{js}), LET(a_{js})]$ for each $a_{js} \in Re(a_{jm})$ within w_j , respectively, using the method presented in Section 5.2. However, suppose there is a resource dependency between two activities $a_{ip} \in Re(a_{ik})$ and $a_{js} \in Re(a_{jm})$, the possible relations between their estimated active intervals can be described distinctly by Allen's interval logic, shown in

Fig. 7 [19,4]. If $[EST(a_{ip}), LET(a_{ip})] \cap [EST(a_{js}), LET(a_{js})] \neq \phi$, their estimated active intervals need to be adjusted in order to avoid resource conflict in the concurrent execution. Without the loss of generality, here we suppose $EST(a_{ip}) \leq EST(a_{js})$. Then a_{js} 's estimated active interval should be adjusted in terms of the resource dependency with a_{ip} as follows:

$$EST(a_{js}) = \text{MAX}\{EST(a_{js}), EST(a_{ip}) + d(a_{ip})\},$$

$$LET(a_{js}) = \text{MAX}\{LET(a_{js}), LET(a_{ip}) + D(a_{ip})\}.$$

After the adjustment of a_{js} 's estimated active interval above, given an active interval of a_{ip} such that $[S(a_{ip}), E(a_{ip})] \subseteq [EST(a_{ip}) + S(a_{ik}), LET(a_{ip}) + S(a_{ik})]$ in workflow schedule \mathcal{S} , we can get an active interval $[S(a_{js}), E(a_{js})] \subseteq [EST(a_{js}) + S(a_{jm}), LET(a_{js}) + S(a_{jm})]$ such that $[S(a_{ip}), E(a_{ip})] \cap [S(a_{js}), E(a_{js})] = \phi$ in the workflow schedule.

So far, we have described the adjustment method for the estimated active intervals of two activities with a resource dependency. In fact, the adjustment for the estimated active intervals of all activities with resource dependencies is an iterative process. Given a set of workflow specifications $WS = \{ws_1, ws_2, \dots, ws_m\}$ associated with workflow schedule $\mathcal{S} = \langle W, A_S, <_S \rangle$ and reference point a_{ik} , we present here the steps for calculating the estimated active intervals of some activities in WS as follows:

- Step 1. Select a reference point for each workflow in W . Assume $S(a_{ik}) = t_i$, at that moment, activity a_{jm} is selected as the reference point of w_j , ($j \neq i$), if it is in the running state.
- Step 2. Calculate the estimated active intervals of activities within a single workflow specification. For each workflow $w_j \in W$, ($j = 1, \dots, m$), and the selected reference point a_{jm} , we calculate $[EST(a_{js}), LET(a_{js})]$ for each activity $a_{js} \in Re(a_{jm})$ within workflow w_j using the method described in Section 5.2.
- Step 3. For each workflow $w_j \in W$, ($j \neq i$), we need to adjust the estimated active intervals of activities $a_{js} \in Re(a_{jm})$ because

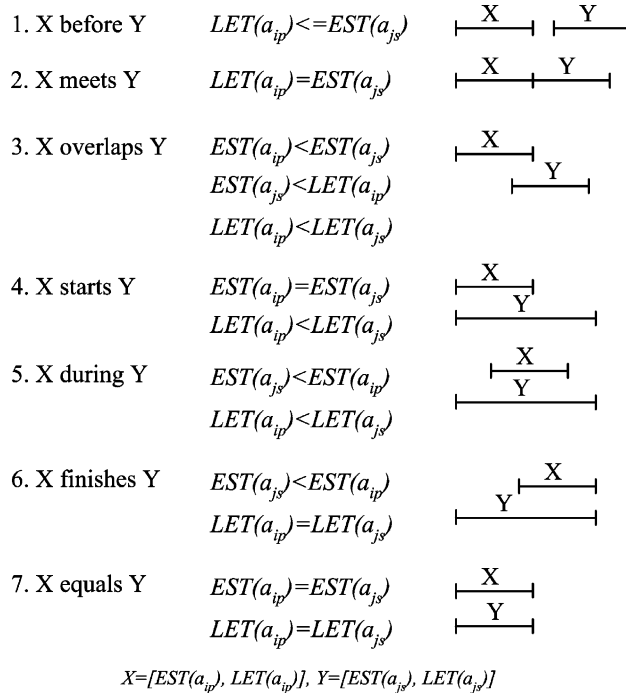


Fig. 7. Temporal relations between two estimated active intervals.

a_{jm} may have started for a while. Suppose $S(a_{jm}) = t_j (t_j < t_i)$, let $\delta = t_i - t_j$. For each activity $a_{js} \in Re(a_{jm})$, the estimated active interval is adjusted as $[EST(a_{js}) - \delta, LET(a_{js}) - \delta]$.

Step 4. All those activities having the estimated active intervals in $\cup AN_i, i = 1, \dots, m$, compose a sequence, Q , according to their EST values with an ascending order. Now, we process this sequence in the following steps until it becomes empty.

Step 4.1. Remove an activity from the head of Q and assume it as a_{ip} .

Step 4.2. Let $Dep(a_{ip})$ be the set of those activities in Q which have resource dependencies with a_{ip} . Similarly, all elements in $Dep(a_{ip})$ are sorted by their EST values with an ascending order.

Step 4.3. If $Dep(a_{ip}) = \phi$, then go to Step 4.1; otherwise remove an activity with the smallest EST value from $Dep(a_{ip})$ and assume it as a_{js} .

Step 4.4. If $[EST(a_{ip}), LET(a_{ip})] \cap [EST(a_{js}), LET(a_{js})] = \phi$, then go to Step 4.3; otherwise go to Step 4.5.

Step 4.5. According to the previous discussion, $EST(a_{js})$ and $LET(a_{js})$ are adjusted as follows:

$$EST(a_{js}) = \text{MAX}\{EST(a_{js}), EST(a_{ip}) + d(a_{ip})\},$$

$$LET(a_{js}) = \text{MAX}\{LET(a_{js}), LET(a_{ip}) + D(a_{js})\}.$$

Then, for each activity $a_{jt} \in Re(a_{js})$ (excluding a_{js}), its estimated active interval is re-calculated within the workflow specification ws_j accordingly. After that, go to Step 4.3. \square

In fact, at last, we get a complete workflow schedule S_c of S . More formally:

Definition 5.6. (Complete workflow schedule) Let $S = \langle W, A_S, \prec_S \rangle$ be a workflow schedule. The complete workflow schedule of S at reference point a_{ik} , S_c , is a 3-tuple $\langle W_{S_c}, A_{S_c}, \prec_{S_c} \rangle$, where (i) $W = W_{S_c}$. (ii) A_{S_c} is a set of activities derived from A_S in the following ways: (a) For each workflow $w_i \in W$,

($i = 1, \dots, m$), if $a_{ip} \in A_i$, then $a_{ip} \in A_{S_c}$. (b) For each workflow $w_i \in W$, ($i = 1, \dots, m$), suppose a_{ik} is a reference point, if $a_{ip} \in Re(a_{ik})$, then $a_{ip} \in A_{S_c}$. (iii) The partial order, \prec_{S_c} is determined as follows: (a) For any two activities $a_{ip}, a_{js} \in A_S$, if $a_{ip} <_S a_{js}$ in S , then $a_{ip} \prec_{S_c} a_{js}$ in S_c . (b) For two activities $a_{ip}, a_{is} \in (A_{S_c} - A_S)$,³ if $Reachable(a_{ip}, a_{is}) = True$, then $a_{ip} \prec_{S_c} a_{is}$ in S_c . (c) For two activities $a_{ip} \in A_S$ and $a_{is} \in (A_{S_c} - A_S)$, we have $a_{ip} \prec_{S_c} a_{is}$ in S_c . (iv) For every workflow $w_i \in W$, ($i = 1, \dots, m$), each activity $a_{ip} \in A_i$ having completed in S has an active interval $[S(a_{ip}), E(a_{ip})]$, and each activity $a_{is} \in Re(a_{ik})$ has an estimated active interval $[EST(a_{is}), LET(a_{is})]$ relative to reference point a_{ik} .

Theorem 2. If workflow schedule $S = \langle W, A_S, <_S \rangle$ is correct, then the complete workflow schedule of S , $S_c = \langle W_{S_c}, A_{S_c}, \prec_{S_c} \rangle$, at reference point a_{ik} is also correct.

Proof. Given any two activities $a_{ip}, a_{js} \in A_{S_c}$ with a resource dependency, we distinguish the relation between them by the following complete cases:

- *Case 1:* $a_{ip}, a_{js} \in A_S$. Since workflow schedule S is correct, from Definition 4.2, we have $([S(a_{ip}), E(a_{ip})] \cap [S(a_{js}), E(a_{js})]) = \phi$.
- *Case 2:* $a_{ip} \in A_S, a_{js} \in (A_{S_c} - A_S)$. We have $a_{ip} \prec_{S_c} a_{js}$ from Definition 5.6, then $([S(a_{ip}), E(a_{ip})] \cap [S(a_{js}), E(a_{js})]) = \phi$.
- *Case 3:* $a_{ip} \in (A_{S_c} - A_S), a_{js} \in A_S$. We have $a_{js} \prec_{S_c} a_{ip}$ also from Definition 5.6, then $([S(a_{ip}), E(a_{ip})] \cap [S(a_{js}), E(a_{js})]) = \phi$.
- *Case 4:* $a_{ip}, a_{js} \in (A_{S_c} - A_S)$. Suppose $EST(a_{ip}) \leq EST(a_{js})$, from the above calculating steps for activities' estimated active intervals, we have: given an active interval $[S(a_{ip}), E(a_{ip})]$ in workflow schedule S , there exists an active interval $[S(a_{js}), E(a_{js})] \subseteq [EST(a_{js}) + S(a_{jm}), LET(a_{js}) + S(a_{jm})]$ (a_{jm} is a selected reference point) such that $[S(a_{ip}), E(a_{ip})] \cap [S(a_{js}), E(a_{js})] = \phi$

From the above discussion, S_c is also correct. \square

³ $A_{S_c} - A_S$ is a set composed of elements, $a \in A_{S_c}$, but $a \notin A_S$.

5.4. Checking process for temporal constraints

As stated earlier, the purpose of this paper is to check if the temporal constraint $D_R(a_{ik}, a_{il}) \leq p$ can be satisfied at reference point a_{ik} in workflow schedule $S = \langle W, A_S, <_S \rangle$. Based on the previous discussion, the checking process is summarised as follows: Firstly, we construct the complete workflow schedule of S at the reference point. Then according to Theorems 1 and 2, the satisfaction of the temporal constraint can be decided by the following three cases:

- *Case 1:* $p < EET(a_{il})$. The temporal constraint can be violated at a_{ik} .
- *Case 2:* $LET(a_{il}) \leq p$. The temporal constraint can be satisfied at a_{ik} .
- *Case 3:* $EET(a_{il}) \leq p < LET(a_{il})$. The temporal constraint cannot be decided at a_{ik} .

Now we can dynamically check if a temporal constraint can be satisfied at a reference point through the above process. The result of satisfaction means that the associated workflow is executing under the normal situation. Otherwise, if the result is uncertain or violated, some measures should be taken to tackle these abnormal cases, for example, triggering exception handling, assigning a higher priority to the workflow or adjusting the temporal constraint. This issue is beyond the scope of this paper and will be addressed in detail elsewhere.

6. Application

The previous section has presented our approach for the verification of temporal constraints for concurrent workflows. In this section, we illustrate an application of this approach through an example in the real world and discuss some issues in its application.

6.1. An example

We take Computer Integrated Manufacturing (CIM) environments as an example to demonstrate the checking process for temporal

constraints in an environment with concurrent workflows. This example is originated from [17] with minor modification.

6.1.1. Scenario

In a CIM environment, it is assumed that productions do not follow mass-production techniques but aim at customising each one of them to deliver. The development and production of a new product are composed of two processes, namely construction process and production process. The construction process contains all developing steps from the design of a new part to the final test and the subsequent technical documentation. The production process includes all manufacturing steps from the ordering of materials to the production floor including the necessary scheduling. In order to bring products to marketplace as soon as possible, the development of a product and its manufacture are strongly tied. Therefore, the construction process and the production process maybe execute concurrently.

These two processes can be specified to be two workflow specifications, respectively, as shown in Fig. 8. Resources and time information are annotated in the figure as well.

A workflow schedule, S_3 , on the workflow specifications is initiated at a time. As shown in Fig. 8, workflow w_1 corresponding to ws_1 starts at time 0, and w_2 corresponding to ws_2 starts at time 7. Now suppose that a_{21} is a reference point, we check if temporal constraint $E(a_{26}) \leq_t 24$ can be satisfied at the reference point.

6.1.2. Verification

Temporal constraint $E(a_{26}) \leq_t 24$ is an absolute temporal constraint. So, it needs to be transformed into its canonical representation, that is, the relative temporal constraint. Since $S(a_{21}) = 7$, according to the discussion in Section 2.3, the canonical representation is calculated to be $D_R(a_{21}, a_{26}) \leq_t p$, where $p = 24 - 7 = 17$.

At the moment when a_{21} is being initiated, workflow schedule S_3 includes three activities, that is, a_{11}, a_{12} and a_{13} . According to Definition 4.2, S_3

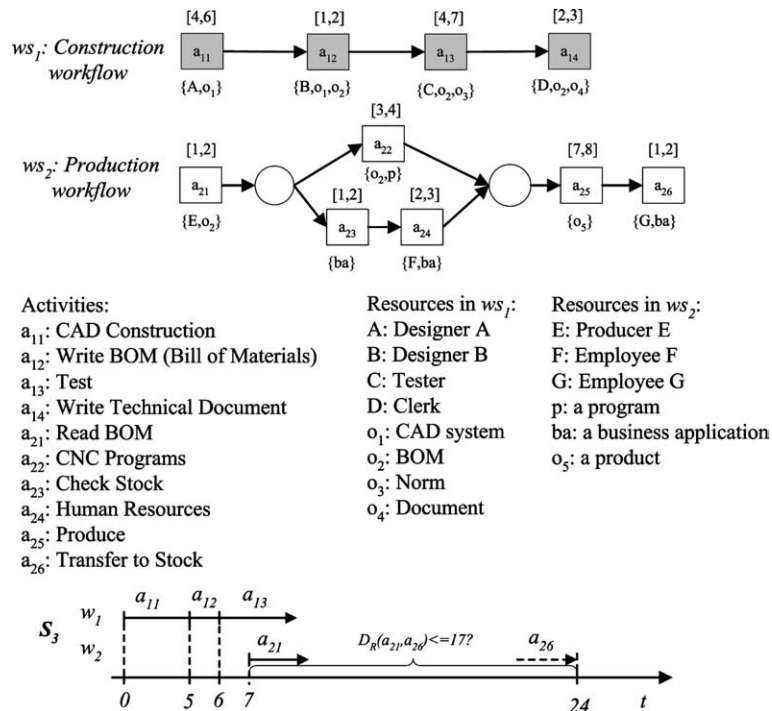


Fig. 8. Example of checking temporal constraints from concurrent workflows.

is a correct workflow schedule. Now we can construct its complete workflow schedule. As shown in Fig. 8, the complete workflow schedule includes all activities in workflow specifications w_{S1} and w_{S2} . Each completed activity has an active interval; and each of the other activities has an estimated active interval. According to the discussion in Section 5.3, the estimated active intervals can be calculated in the following steps:

Step 1. At the start time of a_{21} , a_{13} is in the running state. Therefore, a_{13} is selected as the reference point of w_1 .

Step 2. According to Definition 5.1, we have $Re(a_{13}) = \{a_{13}, a_{14}\}$ and $Re(a_{21}) = \{a_{21}, a_{22}, a_{23}, a_{24}, a_{25}, a_{26}\}$. The estimated active intervals of activities in $Re(a_{13})$ and $Re(a_{21})$ are calculated within w_{S1} and w_{S2} , respectively, as follows:

$a_{13}: [0, 7]$, $a_{14}: [4, 10]$;
 $a_{21}: [0, 2]$, $a_{22}: [1, 6]$, $a_{23}: [1, 4]$, $a_{24}: [2, 7]$, $a_{25}: [4, 15]$,
 $a_{26}: [11, 17]$.

Step 3. Since a_{13} has executed for 1 time unit, we have $\delta = 1$. Then the estimated active intervals of activities in $Re(a_{13})$ are adjusted as:

$a_{13}: [-1, 6]$, $a_{14}: [3, 9]$.

The estimated active intervals of activities in $Re(a_{21})$ remain unchanged, that is:

$a_{21}: [0, 2]$, $a_{22}: [1, 6]$, $a_{23}: [1, 4]$, $a_{24}: [2, 7]$, $a_{25}: [4, 15]$,
 $a_{26}: [11, 17]$.

Step 4. A sequence composed of these activities with respect to their EST values is $Q = \langle a_{13}, a_{21}, a_{22}, a_{23}, a_{24}, a_{14}, a_{25}, a_{26} \rangle$. Now we process this sequence through executing Steps 4.1 to 4.5 as described in Section 5.3. In Step 4.1, remove a_{13} from the head of Q . In Step 4.2, $Dep(a_{13})$ is calculated as $Dep(a_{13}) = \{a_{21}, a_{22}, a_{14}\}$. So, we go to Step 4.3 and remove a_{21} from $Dep(a_{13})$. In Step 4.4, we go to Step 4.5 because of $[EST(a_{13}), LET(a_{13})] \cap [EST(a_{21}), LET(a_{21})] \neq \phi$. In Step 4.5, the estimated active interval of a_{21} is adjusted as $a_{21}: [3, 8]$. Accordingly the estimated active intervals of a_{22} , a_{23} , a_{24} , a_{25} and a_{26} are re-calculated within workflow specification w_{S2} . Then, we have:

$a_{13}: [-1, 6]$, $a_{14}: [3, 9]$;
 $a_{21}: [3, 8]$, $a_{22}: [4, 12]$, $a_{23}: [4, 10]$, $a_{24}: [5, 13]$, $a_{25}: [7, 21]$, $a_{26}: [14, 23]$.

Next we remove a_{22} from $Dep(a_{13})$. Similarly, the estimated active interval of a_{22} is adjusted because of $[EST(a_{13}), LET(a_{13})] \cap [EST(a_{22}),$

$LET(a_{22})] \neq \phi$. Accordingly the estimated active intervals of a_{23} , a_{24} , a_{25} and a_{26} are re-calculated within workflow specification w_{S2} (in this case, the values remain unchanged). Then, we have:

$a_{13}: [-1, 6]$, $a_{14}: [3, 9]$;
 $a_{21}: [3, 8]$, $a_{22}: [4, 12]$, $a_{23}: [4, 10]$, $a_{24}: [5, 13]$, $a_{25}: [7, 21]$, $a_{26}: [14, 23]$.

Although the estimated active interval of a_{14} needs to be adjusted according to Step 4.4, the value remains unchanged.

Repeat the above steps to adjust the estimated active intervals of activities in $Re(a_{13})$ and $Re(a_{21})$. At last, the final values of these estimated active intervals are listed as follows:

$a_{13}: [-1, 6]$, $a_{14}: [7, 15]$;
 $a_{21}: [3, 8]$, $a_{22}: [4, 12]$, $a_{23}: [4, 10]$, $a_{24}: [5, 13]$, $a_{25}: [7, 21]$, $a_{26}: [14, 23]$.

According to Proposition 5.1, $EET(a_{26})$ is calculated as $EET(a_{26}) = EST(a_{26}) + d(a_{26})$, that is, $14 + 1 = 15$ time units. By checking the relation between time value $p = 17$ and the time interval $[15, 23]$, we conclude that the temporal constraint $E(a_{26}) \leq_t 24$ cannot be decided at a_{21} according to Theorem 1. \square

In this example, if we do not consider those resource dependencies between w_1 and w_2 and suppose they execute separately, the value of $LET(a_{26})$ is 17. So, the temporal constraint $E(a_{26}) \leq_t 24$ can be satisfied at a_{21} . However, the checking result becomes uncertain because activity a_{21} might be delayed by a_{13} in the workflow schedule. Furthermore, we point out that a_{26} definitely ends within 23 time units relative to the reference point a_{21} under the normal condition. That is to say that the temporal constraint $E(a_{26}) \leq_t 30$ can be satisfied at a_{21} .

6.2. Discussion

The proposed approach is usually used by workflow enactment components to check temporal constraints at run-time. However, some issues should be closely inspected in practical application:

The estimation of the minimum and maximum durations. It is impossible to get the precise duration for an activity in a workflow specification at build-time. So at build-time each activity is estimated with two time values: the minimum and maximum

durations. It is assumed that the run-time duration is within the interval from the minimum to the maximum duration. The correctness of the verification algorithm depends on the correct estimation of these time values of activities. In the practical application, we can estimate them more accurately.

The selection of reference points. The verification algorithm is invoked by workflow enactment components at a reference point. Some policies can be used to determine a set of reference points for each temporal constraint: (I) at the start time of each activity – in this policy, the temporal inconsistency of a temporal constraint can be identified at the earliest time, but it is not practical given a large number of activities in a workflow; (II) at the start time of an activity succeeding a decision node in a workflow specification – this option is not practical either because we do not know which branch will be taken during workflow execution at all based on a workflow specification [13]; (III) at the time when an activity executes overtime – once an activity's run-time duration is beyond the interval from the minimum to the maximum duration, the algorithm should be invoked to check the temporal constraint; and (IV) user-defined reference points – the temporal constraint can be verified at a user-defined reference point in a workflow specification.

The application of component-based technology. The complexity of the proposed algorithm might be unmanageable when it is used in large-scale workflows. Component-based technology can be introduced to solve this problem. In fact the component-based approach has been introduced in the development of workflow management systems [22]. So the complexity of the verification algorithm used in large-scale workflows can be divided and conquered if the workflow management systems are developed based on component technology.

7. Related work

In this section, we present the previous work on verification of temporal constraints in WfMSs. Generally speaking, previous work on verification of temporal constraints is focused on using static

analysis techniques to predict the run-time state of workflows, while little or no attention has been paid to dynamic checking for temporal constraints on workflows at run-time. Unfortunately, it is not always possible to statically determine that a temporal constraint will be satisfied at all cases of workflow execution.

Pozewaunig et al. [14] uses an extended netdiagram technique PERT (program evaluation and review technique), namely ePERT, to support time management in WfMSs. Each activity in a workflow specification is estimated with a time value for its duration. Other time values (including the earliest best case, the latest best case, the earliest worst case, and the latest worst case) for a state are calculated using the critical path method (CPM) on the basis of the estimated time value and the workflow structure. The time information, as the authors claim, can be used to pro-actively avoid time errors and reactively resolve time failures at run-time. However, this issue is not discussed in depth in that paper. Similar to [14], [5] defines a timed workflow graph by augmenting each activity node with two values: the earliest end time and the latest end time. Based on the definitions, temporal constraints, namely fixed-date constraints, lower-bound constraints and upper-bound constraints, can be calculated by using the modified CPM method at build-time and process instantiation time, and then enforced at run-time. The approaches reported in [14,5] have an assumption with an ideal scenario that each activity occupies a fixed duration, which is not realistic in real-workflow execution.

In [20], Zhao and Stohr develop a framework for temporal workflow management in the context of claim handling. Issues including the prediction of turnaround time, time allocation policy and task prioritisation policy are investigated in that paper. Although some factors, such as the queue time of a task, priority of a claim and probability for selecting a branch, are taken into account in predicting turnaround times and estimating time allocation for tasks, the framework proposed is still based on the same assumption as [14,5]. Additionally, they do not address the technique of dynamic verification for temporal constraints.

[1,13,21] assume that an activity in a workflow specification is estimated with a minimum and

maximum duration (relative time values) at build-time. During workflow execution, a completed activity has a start time and an end time (absolute time values). Using the time information, [13] presents a method for dynamic verification of absolute deadline constraints and relative deadline constraints. The approach presented in [21] differs from that in [13] in three aspects: (1) each flow between activities is also assigned with a minimum and maximum duration at build-time, a start time and an end time at run-time; (2) the time difference is taken into account in distributed execution environments, then each activity is assigned with a time axis; and (3) the proposed consistency checking incorporates the exact run-time duration and the estimated build-time duration. [1] employs Temporal Constraint Petri Nets (TCPN) to specify workflows, and tests the temporal feasibility for a workflow at build-time. The problem in those approaches is that the verification of temporal constraints is limited to a single workflow and does not consider the case with concurrent workflows.

A recent paper [2] investigates temporal constraints and the related reasoning algorithms in workflow systems. At build-time, each flow in a workflow specification is assigned with a set of temporal constraints, each activity with an interval denoting its minimum and maximum duration, respectively. Based on the definition of constraint networks, the temporal consistency can be checked for the workflow specification, and the time frame for the involved activities can also be predicted. At run-time, a temporal support module provides the service of scheduling activities in a workflow specification. To this end, the authors of [2] identify three interesting families of enactment schedules for autonomous agents, namely free schedules, restricted due-time schedules and bounded schedules. The corresponding algorithms are presented in that paper as well. Furthermore, the work is extended to the case of temporal constraints where multiple time granularities are involved. Compared with those approaches in [5,13,21], a more general set of temporal constraints in workflow specifications are considered in that paper. However, the problem in this method is the same as before, that is, limited to a single workflow.

8. Concluding remarks

The ability to dynamically check temporal constraints is crucial to workflow management. The existing approaches are usually limited to a single workflow and employ static techniques to predict the run-time state of workflows. In reality, there are multiple workflows executing concurrently in a workflow management system. So, we believe that those approaches are too weak for verifying temporal constraints in an environment with concurrent workflows.

This paper proposes a new approach to dealing with this problem in the environment with concurrent workflows. Firstly, we unify the representation of absolute temporal constraints and relative temporal constraints, and provide them with a canonical representation. Then, we investigate resource constraints between activities and define concurrent workflow executions. Subsequently, the verification method is unfolded based on these discussions. Compared with the existing approaches, the method presented in this paper takes into account the relationship between concurrent workflows. In addition, it is dynamic and can be used by workflow management systems at run-time.

For future work, the proposed approach needs to be improved in the following aspects:

1. To avoid the violation of temporal constraints, we use a more conservative method to solve this problem. A temporal constraint is checked before the execution of the corresponding activity. Therefore, our approach is based on the estimated time values, namely the minimum and maximum durations, which are defined at build-time. However, a truly dynamic checking of temporal constraints should be based on the actual duration of activities. If the violation of a temporal constraint has happened, some measure can be used to solve the exception. This question needs further research.
2. Our approach does not allow any concurrent execution of two activities if there is a resource dependency between them. We acknowledge this is more restrictive than necessary although this restriction does not affect the correctness

of our checking algorithm in practical applications. For two activities with a resource dependency, it is possible that both of them do not occupy their resources from the start to the end. They are allowed to execute concurrently in some cases. Therefore, the checking algorithm can be improved.

In addition to the above two aspects, the work-item prioritisation and resource allocation policies will be taken into account in the future as well. The resources can be allocated to activities with a higher priority for the purpose of the satisfaction of temporal constraints.

Acknowledgements

The work reported in this paper is supported in part by Swinburne VC Strategic Research Initiative Grant (2002–2004), as well as the National Natural Science Foundation of China under Grant Nos. 60273026 and 60273043.

References

- [1] N. Adam, V. Atluri, W. Huang, Modeling and analysis of workflows using petri nets, *Journal of Intelligent Information Systems: Special Issue on Workflow and Process Management* 10 (2) (1998) 131–158.
- [2] C. Bettini, X. Wang, S. Jajodia, Temporal reasoning in workflow systems, *Distributed and Parallel Databases* 11 (3) (2002) 269–306.
- [3] F. Casati, P. Grefen, B. Pernici, et al. WIDE workflow model and architecture. public documents. <www.dis.sema.es/projects/WIDE/Documents/ase30_4.ps.gz> April, 1996.
- [4] S. Chinn, G. Madey, Temporal representation and reasoning for workflow in engineering design change review, *IEEE Transactions on Engineering Management* 47 (4) (2000) 485–492.
- [5] J. Eder, E. Panagos, M. Rabinovich. Time constraints in workflow systems, in: *Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE'99)*, LNCS vol. 1626, Springer, Germany, 1999, pp. 286–300.
- [6] G. Fakas, B. Karakostas, A workflow management system based on intelligent collaborative objects, *Information and Software Technology* 41 (1999) 907–915.
- [7] D. Georgakopoulos, M. Hornick, A. Sheth, An overview of workflow management: from process modeling to workflow automation infrastructure, *Distributed and Parallel Databases* 3 (1995) 119–153.
- [8] P. Grefen, J. Vonk, P. Apers, Global transaction support for workflow management systems: from formal specification to practical implementation, *The VLDB Journal* 10 (4) (2001) 316–333.
- [9] D. Hollingsworth, Workflow management coalition: the workflow reference model TC00-1003. <www.wfmc.org>, 1995.
- [10] H. Li, Y. Yang, T.Y. Chen, Resource constraints analysis of workflow specifications, *Journal of Systems and Software* 73 (2) (2004) 271–285.
- [11] H. Li, Y. Yang. Verification of temporal constraints for concurrent workflows, in: *Proceedings of the Sixth Asia-Pacific Web Conference (APWeb 2004)*, LNCS vol. 3007, Springer, Hangzhou, China, April 14–17, 2004, pp. 804–813.
- [12] O. Marjanovic, M.E. Orlowska, On modeling and verification of temporal constraints in production workflows, *Knowledge and Information Systems* 1 (2) (1999) 157–192.
- [13] O. Marjanovic, Dynamic verification of temporal constraints in production workflows, in: *Proceedings of the Australian Database Conference*, Canberra, Australia, 2000, pp. 74–81.
- [14] H. Pozawaunig, J. Eder, W. Liebhart, ePERT: extending PERT for workflow management systems, in: *Proceedings of the First East-European Symposium on Advances in Database and Information Systems*, St Petersburg, 1997, pp. 217–224.
- [15] M. Reichert, T. Bauer, P. Dadam. Enterprise-wide and cross-enterprise workflow management: challenges and research issues for adaptive workflows. *Enterprise-wide and Cross-Enterprise Workflow Management: Concepts, Systems, Application*, Germany, 1999.
- [16] W. Sadiq, M.E. Orlowska, Analysing process models using graph reduction techniques, *Information Systems* 25 (2) (2000) 117–134.
- [17] H. Schudt, G. Alonso, C. Beerli, H. Schek, Atomicity and isolation for transactional processes, *ACM Transactions on Database Systems* 27 (1) (2002) 63–116.
- [18] E. Stohr, J. Zhao, Workflow automation: overview and research issues, *Information Systems Frontiers: Special Issue on Workflow Automation and Business Process Integration* 3 (3) (2001) 281–296.
- [19] A. Zaidi, On temporal logic programming using petri nets, *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans* 29 (3) (1999) 245–254.
- [20] J. Zhao, E. Stohr. Temporal workflow management in a claim handling system, in: *Proceedings of Work Activities Coordination and Collaboration (WACC'99)*, San Francisco, CA, USA, 1999, pp. 187–195.
- [21] H. Zhuge, T. Cheung, H. Pung, A timed workflow process model, *The Journal of Systems and Software* 55 (3) (2001) 231–243.
- [22] H. Zhuge, Component-based workflow systems development, *Decision Support Systems* 35 (2003) 517–536.