

Resource constraints analysis of workflow specifications

Hongchen Li ^{a,*}, Yun Yang ^a, T.Y. Chen ^b

^a Centre for Internet Computing and E-Commerce, School of Information Technology, Swinburne University of Technology, P.O. Box 218, Hawthorn VIC 3122, Melbourne 3122, Australia

^b Centre for Software Engineering, School of Information Technology, Swinburne University of Technology, Melbourne 3122, Australia

Received 30 October 2002; received in revised form 3 August 2003; accepted 10 August 2003

Available online 25 December 2003

Abstract

A workflow specification is a formal description of business processes in the real world. Its correctness is critical to the workflow execution and hence the realisation of business objectives. In addition to structural and temporal constraints, resource constraints are also implied in workflow specifications. Therefore, they should be analysed to ensure that the workflow specification is resource consistent at build-time. In this paper, we first identify the problem of resource constraints in a workflow specification. Then we propose an innovative approach with corresponding algorithms to the checking of resource consistency for a workflow specification. Furthermore, we extend our analysis work to timed workflow specifications, where time information is taken into consideration for the checking of the resource consistency of a workflow specification. The work reported in this paper provides a theoretical foundation for workflow modeling and analysis in workflow management.

© 2003 Elsevier Inc. All rights reserved.

Keywords: Workflows; Timed workflows; Workflow specifications; Workflow analysis; Resource constraints

1. Introduction

Workflow management has emerged as an important technology designed to support modeling, redesign and execution of business processes. According to workflow management coalition (WfMC), *workflow* is defined as the computerised facilitation or automation of a business process, in whole or part. A *workflow management system* (WfMS) is a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic (WfMC, 1995). So far, numerous workflow products or prototypes have been developed, with some on the market.

In general, a WfMS consists of two main functional components: a workflow modeling component and a workflow enactment component. The former offers a

build-time environment where workflow specifications can be defined, analysed and managed. In addition, the component also supports the persistent storage for workflow specifications. The latter, however, provides a run-time environment for the creation, execution and management of workflows. In the course of workflow execution, the component possibly interacts with actors or some external applications for the correctness of workflow execution.

In order to support the automation of business processes, they should be abstracted from the real world and specified using a language, namely, *workflow specification language*. The result is called *workflow specification*, which contains information formally describing various aspects of a workflow, for example, the process aspect, information aspect and organisation aspect (Chan et al., 1997). The process aspect describes the structure of a workflow using such entities as activities, subprocesses (a process that is enacted or called from another process or subprocess) (WfMC, 1995), as well as flows (data and control flows) between them. The information aspect addresses what are input and output by activities in a workflow. The organisation aspect defines entities

* Corresponding author. Tel.: +61-3-9214-8934; fax: +61-3-9819-0823.

E-mail addresses: hli@it.swin.edu.au (H. Li), yyang@it.swin.edu.au (Y. Yang), tychen@it.swin.edu.au (T.Y. Chen).

belonging to an organisation or a virtual organisation as well as the relationship between them.

Building workflow specifications is a complex and error-prone process, especially for the large-scale ones. It is likely to introduce inconsistencies or errors to these workflow specifications. For example, “deadlock” or “lack of synchronisation”, as reported in Sadiq and Orlowska (2000), may appear in workflow specifications due to the false representation. Such inconsistencies or errors may lead to incorrect execution of some or all workflow cases. Only a few investigations (Aalst, 1998; Adam et al., 1998; Davulcu et al., 1998; Marjanovic, 2000) have addressed this problem and proposed some approaches for the verification of workflow correctness from structural and temporal aspects. However, activities in a workflow usually access some resources during their executions. So, in addition to structural and temporal constraints, resource constraints are also implied in a workflow specification, which affect greatly the execution of a workflow. In a WfMS, some resources can be shared by 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, the resources are released and can be accessed by other activities. Suppose that a resource constraint between two activities within a workflow specification is not represented correctly, the activities probably compete for the same resources in a workflow, and then result in a conflict. Therefore, the workflow specification should be analysed in terms of resource constraints at build-time rather than only at run-time in order to identify this kind of potential conflicts; and then these potential conflicts should be removed from the workflow specification.

To the best of our knowledge, no work has been done on analysing resource constraints in workflow specifications. Based on the past work on workflow verification, this paper takes this challenge to tackle this problem in workflow community and proposes analysis method for resource constraints in workflow specifications. At first, the problem of resource constraints is identified in workflow specifications. Then, an approach is proposed for the verification of resource consistency for workflow specifications, with the corresponding algorithms developed. Moreover, we extend our work to timed workflow specifications. The method presented in this paper can be used to check a workflow specification at build-time for its resource consistency.

The remainder of this paper is organised as follows. The next section introduces some preliminary knowledge on workflow specifications. Section 3 discusses resource constraints in workflow specifications. After that, we address the analysis on resource constraints in workflow specifications in Section 4, and extend the same analysis to timed workflow specifications in Section 5. In Section 6, we suggest several ways to remove potential resource

conflicts from workflow specifications. Section 7 summarises the related work on workflow analysis. Finally, Section 8 draws concluding remarks.

2. Background

2.1. Workflow specification

Conceptually, a workflow is a collection of activities, together with their 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 (Adam et al., 1998). 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; while a composite activity is an abstract description of another process and can be decomposed into a workflow (WFMC, 1995). 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, that is, sequential, parallel, selective and iterative structures, are defined in the workflow reference model (WFMC, 1995).

Usually, workflows are specified to be workflow specifications according to specific syntax rules. An activity is denoted by a node (called activity node). In order to represent the above control structures, four types of nodes (called control nodes) are introduced, that is, and-split (as), and-join (aj), or-split (os) and or-join (oj). Formally, a workflow specification can be defined as follows:

Definition 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) $F = \{f_i \mid f_i = \langle n_s, n_t \rangle, n_s, n_t \in N\}$ 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

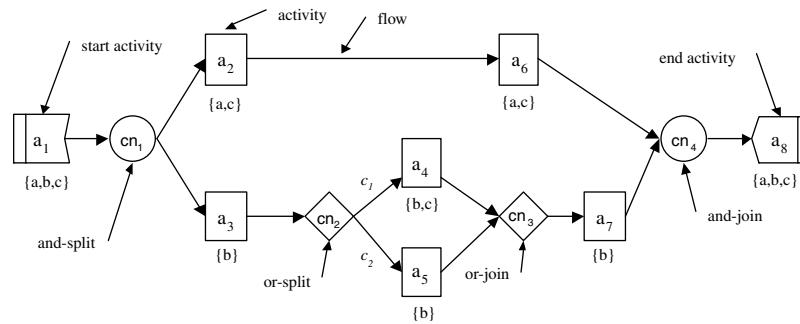


Fig. 1. Graphic representation of a workflow specification.

(see Fig. 1). The meanings of all symbols are annotated in Fig. 1, where small squares represent activities involved in a workflow, small circles and diamonds denoting control nodes in the workflow, arrows denoting flows between 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 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. For example, an iterative structure can be denoted by a composite activity. The resource set accessed by this composite activity is composed of all resources accessed by those activities in the iterative structure. On the other hand, the resource consistency for the iterative structure can be verified separately using the same method.

A workflow, also called *workflow instance*, 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. Without the loss of generality, in the rest of this paper, we consider only one execution case of a workflow specification, and hence identifiers marking the workflow and activities are omitted, using the same symbols as in the workflow specification unless explicitly specified.

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 between them, which prescribe the precedence relationship between those 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 $n_i \prec n_j$. More formally:

Definition 2 (Workflow). A workflow, w , is a 4-tuple $\langle id, ws, A, \prec \rangle$, 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) that contains a subset of activities in ws .
- (iv) The *execution order* $\prec \subseteq (A \times A)$ 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 $a_i \prec a_j$.

Note that the execution order \prec is transitive, that is, let $a_i, a_j, a_k \in A$ be three activities in workflow w , if there exist $a_i \prec a_j$ and $a_j \prec a_k$, then $a_i \prec a_k$.

Not all activities are scheduled in a workflow. Some activities may not be instantiated because of the selective structures in the workflow specification. Given two nodes $n_i, n_j \in N$ in a workflow specification, we use $w(n_i * n_j)$ to denote that there exists such a workflow w that both n_i and n_j are scheduled in it, and $n_i \oplus n_j$ to denote that they will never be scheduled together in a workflow. In this paper, we assume only one branch is selected in executing a selective structure. For example, as shown in Fig. 1, either a_4 or a_5 will be scheduled in a workflow, but they are never scheduled in a workflow together.

2.2. Timed workflow specification

A *timed workflow specification* is constructed by augmenting each activity in a traditional workflow specification (see Definition 1) with two time values, that is, the minimum and maximum durations. So, at build-time, we define $d(a)$ and $D(a)$ as the minimum and maximum durations of activity a in its execution respectively ($d(a) < D(a)$) (Zhuge et al., 2001). Time is expressed in some basic time units, such as minutes, hours, or days. The granularity is selected according to specific workflow applications.

We designate the start time of a workflow as the *reference point* (denoted as P). At run-time, a completed

activity a in the workflow has a start time (denoted as $S(a)$) and an end time (denoted as $E(a)$) relative to the reference point. Activity a is active during the period from its start time to end time. This period is called the *active interval* of a , denoted as $[S(a), E(a)]$. $D_R(a) = E(a) - S(a)$ is defined as its *run-time duration*.

Corollary 1. *Given a timed workflow specification, under the normal condition, we have: (1) $d(a) \leq D_R(a) \leq D(a)$ for any activity a in a workflow; (2) $E(a_i) \leq S(a_j)$, if $\exists f = \langle a_i, a_j \rangle \in F$.*

Proof. Proof of the corollary follows directly from Definition 2 and the above discussion. \square

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. Schuldts et al. (2002) investigate into more depth on addressing compensating activities and their relationship with regular activities. 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.

Temporal constraints are different rules that regulate the time component of a workflow. They are consistent with a workflow specification if and only if they could be satisfied based on the syntax of the workflow specification and expected minimum and maximum durations of activities (Marjanovic, 2000). In other words, all activities are schedulable during workflow execution.

3. Resource constraints in workflow specifications

As stated early, activities in a workflow need to 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 by activities and are only accessed by an activity. So, it is unnecessary to involve the private resources in our analysis because they do not give rise to resource constraints between activities. For those shared resources, we further classify them as two types. One type of resources can be accessed simultaneously by

activities no matter what operations are executed on them; on the contrary, another type of resources is not allowed to do so. In this paper, the former type of resources does not need to be involved in our analysis because they do not result in resource constraints, despite that they can be accessed by different activities at the same time. Finally, we focus on the latter type of resources, which should be treated carefully and are divided into the following two subclasses:

- (I) In this class, whether resources can be accessed simultaneously by activities depends on their access modes. The actions can be allowed if these modes are compatible. However, our analysis work is based on a high-level framework and the concurrency control mechanism for them is in a low-level and can be turned on when more details of a particular application domain are provided. Therefore, they are not involved in our analysis yet.
- (II) The resources in this class are assumed to be occupied exclusively by an activity during its execution, and cannot be accessed by another one until its completion. In this case, we say, there is a resource constraint between these activities.

As the above discussion, resource constraints exist between activities in a workflow, which are implied rules that influence the execution order of those activities and the result of the workflow. In WfMSs, a resource can be modeled as an object (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 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 (Resource dependency). Given two activities a_i, a_j ($i \neq j$) within a workflow specification, we say 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 . Otherwise, a conflict may arise from the competition for the same resources. We call this kind of conflict as resource conflict. It should be noted that a resource conflict in a workflow stems from the erroneous representation of resource constraints between activities (called potential resource conflict) in the associated workflow specification. More formally:

Definition 4 (Potential resource conflict). Given two activities a_i, a_j ($i \neq j$) within a workflow specification, activity a_i has a *potential resource conflict* with a_j if they have a resource dependency and there exists a workflow w such that $([S(a_i), E(a_i)] \cap [S(a_j), E(a_j)]) \neq \phi$.

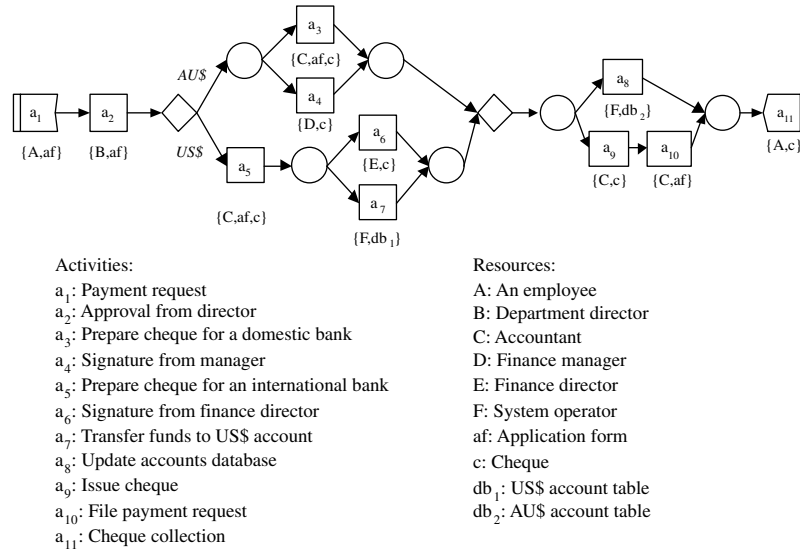


Fig. 2. Example of a workflow specification with a potential resource conflict.

An example illustrating resource constraints between activities is given below.

Example 1. Fig. 2 is the graphic representation of a workflow specification originated from Sadiq and Orłowska (2000) with minor modification. It depicts the processing of payment requests in an Australian organisation. According to the verification approach proposed in Sadiq and Orłowska (2000), this workflow specification is structurally correct, without structural conflicts. As shown in Fig. 2, a_1 and a_2 have a resource dependency because they access the same resource af; and a_3 and a_4 also have a resource dependency due to resource c. Since there is a flow $\langle a_1, a_2 \rangle$ in the workflow specification, no potential resource conflict exists between a_1 and a_2 . However, suppose that a_3 and a_4 execute simultaneously in a workflow, they will compete for the same resource c. This results in a resource conflict. Therefore, a potential resource conflict exists between a_3 and a_4 .

A potential resource conflict in a workflow specification should be closely inspected, as they may help to reveal the following cases:

(I) *Incorrect workflows:* Two activities with a potential resource conflict may execute in any order in a workflow. So, the result of the workflow is affected by the execution order of these activities. For instance, consider the resource dependency between activities a_3 and a_4 in Example 1. As shown in Fig. 2, no control or data flow exists between them. So, suppose a_4 starts to execute in a workflow prior to activity a_3 , a semantic error happens because at that moment the cheque to be signed by finance manager has not yet been created.

(II) *The delay of activities' execution:* For two activities with a resource dependency, if no control or data flow exists between them, one activity may be delayed by another in a workflow because of the unavailability of the required resources. In some circumstances, this results in the violation of temporal constraints on the workflow. For instance, reconsider Example 1. Suppose activities a_3 and a_4 are initiated at the same time in a workflow and a_3 obtains resource c, then activity a_4 will be postponed for its execution due to the resource unavailability. The temporal constraint, if it exists, on the activity is probably violated.

Traditionally, workflow specifications primarily reflect control and data flows between activities, neglecting resource constraints between them. A correct workflow specification guarantees that the resource constraints be consistent with the structural constraints, and further with the temporal constraints.

Definition 5 (Resource consistency). A workflow specification is resource consistent if and only if for any two activities a_i and a_j with a resource dependency, we have: (i) $a_i \prec a_j$ or $a_j \prec a_i$; or (ii) $a_i \oplus a_j$.

Definition 5 can be explained as that for any two activities a_i and a_j with a resource dependency, either one precedes another in a workflow, or they will never be scheduled in a workflow together. These cases can avoid the competition for the same resources between them. From Definition 5, we conclude that no potential resource conflicts exist in a workflow specification with resource consistency.

4. Verification of resource consistency for workflow specifications

As discussed earlier, resource conflicts are caused by the erroneous representation of resource constraints between activities in a workflow specification. Therefore, these potential resource conflicts should be identified and removed from the workflow specification. However, these errors cannot be identified by only analysing structural constraints in the workflow specification. It is needed to analyse resource constraints between activities. In this section, we present our approach with the corresponding algorithm for verifying the resource consistency of a workflow specification.

Given a workflow specification ws as Definition 1, we have the following definitions.

Definition 6 (*Path and acyclic path*). Let $p = \langle n_0, n_1, \dots, n_t \rangle$, where $n_i \in N$, $i = 0, \dots, t$, be a sequence. If $\langle n_i, n_{i+1} \rangle \in F$, where $n_i, n_{i+1} \in N$, $i = 0, \dots, t-1$, then $p = \langle n_0, n_1, \dots, n_t \rangle$ is called a path on ws . The length of p is t , denoted as $|p| = t$. If $n_i \neq n_j$ for $i \neq j$, where $i, j = 0, \dots, t$, then p is regarded as an acyclic path on ws .

Definition 7 (*Reachability*). Node n_j is reachable from node n_i if there is an acyclic path $p = \langle n_i, \dots, n_j \rangle$ on ws .

Let $\text{Reachable}(n_i, n_j)$ be a Boolean function to denote the reachability from node n_i to n_j such that

$$\text{Reachable}(n_i, n_j) = \begin{cases} \text{True}, & \text{if } (\exists p = \langle n_i, \dots, n_j \rangle), \\ \text{False}, & \text{otherwise.} \end{cases}$$

Here, we assume the relation of reachability is reflexive, that is, $\text{Reachable}(n_i, n_i) = \text{True}$. From Definition 7, the following corollary can be derived.

Corollary 2. Given $\text{Reachable}(n_i, n_j) = \text{True}$ in workflow specification ws , we have: (i) There exists a workflow w such that $w(n_i * n_j)$; (ii) $n_i \prec n_j$ in the workflow.

Proof. Given $\text{Reachable}(n_i, n_j) = \text{True}$, there must be a path $p = \langle n_i, n_{i+1}, \dots, n_j \rangle$ on ws according to Definition 7. Hence, there exists such a workflow w where all nodes on this path are scheduled. So, we have $w(n_i * n_j)$ and $n_i \prec n_j$ according to Definition 2. \square

Algorithm 1 checks the reachability from n_i to n_j . In Algorithm 1, the expression $n_k = n_j$ means that they have the same name or id. The algorithm returns *TRUE* indicating that node n_j is reachable from n_i , else returns *FALSE*.

Algorithm 1. $\text{Reachable}(n_i, n_j)$ —Check the reachability from n_i to n_j

1. Initialisation:
let Q be an empty queue; add all out-flows of n_i to Q .
2. Repeat the following steps until $Q = \text{NULL}$
 - 2.1 remove flow f from the head of Q , and denote the sink of f as n_k ;
 - 2.2 if $n_k = n_j$, then return *TRUE*; otherwise append all out-flows of n_k to Q ;
3. Return *FALSE*;

Our method in verifying the resource consistency of a workflow specification is through the analysis on resource constraints between activities. Here, we pay more attention to activities with resource dependencies. Our idea is that for any two activities with a resource dependency, we first check their reachability. If an activity is reachable from another, their execution order is predefined in the workflow specification. Otherwise, we further check whether they might be scheduled in a workflow together. If so, a potential resource conflict exists between these two activities.

However, in order to decide whether two activities may be scheduled in a workflow together, we should analyse the structural relation between them. Given two nodes n_i and n_j in a workflow specification, suppose $\text{Reachable}(n_i, n_j) = \text{False}$ and $\text{Reachable}(n_j, n_i) = \text{False}$, then there must exist such a node (denoted as n_{ca}) that $\text{Reachable}(n_{ca}, n_i) = \text{True}$ and $\text{Reachable}(n_{ca}, n_j) = \text{True}$. Here, node n_{ca} is called the *common ancestor* of these two nodes. For two nodes in a workflow specification, probably there exists more than one common ancestor from which they can be reached. There is a node among them called the *nearest common ancestor*, which is defined next.

Definition 8 (*Distance*). The distance between two nodes n_i and n_j in a workflow specification can be computed as follows:

$$\text{Distance}(n_i, n_j) = \begin{cases} \text{MIN}\{|p_s| \mid p_s = \langle n_i, \dots, n_j \rangle, s = 1, \dots, m\}, & \text{if } (\text{Reachable}(n_i, n_j) = \text{True}), \\ \text{MIN}\{|p_s| \mid p_s = \langle n_j, \dots, n_i \rangle, s = 1, \dots, m\}, & \text{if } (\text{Reachable}(n_j, n_i) = \text{True}), \\ +\infty, & \text{otherwise,} \end{cases}$$

where p_s represents all paths between n_i and n_j .

In Definition 8, the distance between n_i and n_j is computed in considering three cases. If $\text{Reachable}(n_i, n_j) = \text{True}$ or $\text{Reachable}(n_j, n_i) = \text{True}$, the distance is the length of the shortest path between them. Or else, the distance is assumed to be $+\infty$.

Definition 9 (*Nearest common ancestor*). Given two nodes n_i and n_j in a workflow specification, their nearest common ancestor is the node, which is a common ancestor and has the shortest distances to them, denoted as n_{nca} .

For every common ancestor, n_{ca} , of nodes n_i and n_j in workflow specification ws , we have $Distance(n_{ca}, n_i) \leq Distance(n_{ca}, n_j)$ and $Distance(n_{ca}, n_j) \leq Distance(n_{ca}, n_i)$. From the above definitions, if there is $Reachable(n_i, n_j) = True$ in ws , n_i is n_j 's nearest common ancestor. Similarly, if there is $Reachable(n_j, n_i) = True$ in ws , n_j is regarded as n_i 's nearest common ancestor.

Theorem 1. For two nodes n_i and n_j in a structurally correct workflow specification, if both n_k and n_l , $k \neq l$, are their nearest common ancestors, then n_k and n_l have the same type, that is, and-split or or-split.

Proof. If n_i and n_j have two nearest common ancestors, we have $Reachable(n_i, n_j) = False$ and $Reachable(n_j, n_i) = False$. Suppose n_k and n_l have different types, for example, n_k is an or-split node, and n_l is an and-split node (see Fig. 3). From the above definitions, we have $Distance(n_k, n_i) = Distance(n_l, n_i)$ and $Distance(n_k, n_j) = Distance(n_l, n_j)$. At last, the two branches from n_i and n_j , respectively, will merge into one at a control node n_p . If n_p is an and-join node, as shown in Fig. 3(a), a workflow case only via n_k would deadlock at n_p . Or else, if n_p is an or-join node, as shown in Fig. 3(b), a workflow case via n_l would create multiple instances after n_p since the two branches are not synchronised before the merge structure. These two incorrect situations caused by structural conflicts have been addressed in Sadiq and Orlowska (2000). \square

For two nodes in a workflow specification, we can design an algorithm to calculate their nearest common ancestor (see Algorithm 2 below). The algorithm takes the nodes as input and returns their nearest common ancestor. If multiple nearest common ancestors exist, the algorithm finds one of them because they have the same type according to Theorem 1. Algorithm 2 first checks the reachability between n_i and n_j . If there is $Reachable(n_i, n_j) = True$ or $Reachable(n_j, n_i) = True$, n_i or n_j is returned as the nearest common ancestor. Otherwise, from n_i along the reverse direction of flows, the algorithm repeatedly invokes Algorithm 1 to check if there exists a node n_k such that $Reachable(n_k, n_j) = True$. If so, n_k is returned as the nearest common ancestor. From Definition 1, we conclude that n_k must exist in the workflow specification.

Algorithm 2. Calculate the nearest common ancestor of n_i and n_j

1. Initialisation:
 - let Q be an empty queue; add all in-flows of n_i to Q ;
2. If $Reachable(n_i, n_j) = True$, return n_i ;
3. If $Reachable(n_j, n_i) = True$, return n_j ;
4. Repeat the following steps until $Q = NULL$
 - 4.1 remove flow f from the head of Q . denote the source of f as n_k ;
 - 4.2 if $Reachable(n_k, n_j) = True$, return n_k ; else append all in-flows of n_k to Q ;

Theorem 2. For two activities a_i and a_j with a resource dependency in a workflow specification, if their nearest common ancestor n_{nca} is an and-split node, a potential resource conflict exists between them; on the other hand, if n_{nca} is an or-split node, no potential resource conflict exists between them.

Proof. According to Definition 9 and Theorem 1, there exists no such a common ancestor, n_{ca} , of a_i and a_j that $Distance(n_{ca}, n_i) < Distance(n_{nca}, n_i)$ and $Distance(n_{ca}, n_j) < Distance(n_{nca}, n_j)$. So, If n_{nca} is an and-split node, there must exist a workflow w such that $w(a_i * a_j)$. Furthermore, suppose $([S(a_i), E(a_i)] \cap [S(a_j), E(a_j)]) \neq \phi$ in the workflow, a resource conflict is raised in this case. On the other hand, if n_{nca} is an or-split node, we have $n_i \oplus n_j$. Then they will never be scheduled together in a workflow. \square

So far we have presented the algorithms to check the reachability and to calculate the nearest common ancestor of two activities in a workflow specification. Now, we present the algorithm fulfilling the functionality to check the resource consistency of a workflow specification, shown in Algorithm 3. It takes the workflow specification as input, and returns *TRUE* indicating that the workflow specification is resource consistent, otherwise returns *FALSE* and prints all potential resource conflicts in it.

Algorithm 3. Check the resource consistency of a workflow specification

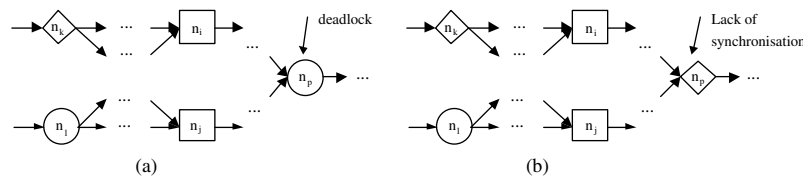


Fig. 3. Incorrect situations of two nearest common ancestors in a workflow specification.

1. Initialisation:
 - 1.1 let S be a set of unchecked activities. S is initialised with all activities in AN;
 - 1.2 $b = TRUE$ is a variable;
2. Repeat the following steps until $S = NULL$
 - 2.1 remove an element from S , denoted as a_i ;
 - 2.2 for each element a_j in S , execute the following steps:
 - 2.2.1 if $\mathfrak{R}(a_i) \cap \mathfrak{R}(a_j) = \phi$, skip to the next iteration;
 - 2.2.2 if $Reachable(a_i, a_j) = True$ (by executing Algorithm 1), skip to the next iteration;
 - 2.2.3 if $Reachable(a_j, a_i) = True$ (by executing Algorithm 1), skip to the next iteration;
 - 2.2.4 invoke Algorithm 2 to compute the nearest common ancestor of a_i and a_j , n_{nca} ;
 - 2.2.5 if n_{nca} is an or-split node, skip to the next iteration;
 - 2.2.6 if n_{nca} is an and-split node, then

/* the detailed information on resource conflicts can be derived. */

 - 2.2.6.1 print (“Potential resource conflict between:”, a_i , “and”, a_j);
 - 2.2.6.2 $b = FALSE$;
3. Return b ;

Algorithm 3 checks the resource constraints between activities one by one. For two activities a_i and a_j with a resource dependency, if there is $Reachable(a_i, a_j) = True$ or $Reachable(a_j, a_i) = True$, we have $a_i \prec a_j$ or $a_j \prec a_i$. Otherwise, the algorithm computes their nearest common ancestor n_{nca} . According to Theorem 2, if n_{nca} is an or-split node, no potential resource conflict exists between them; or else, n_{nca} is an and-split node, then a potential resource conflict exists between a_i and a_j .

5. Verification for timed workflow specifications

In the preceding section, we present the algorithm for checking the resource consistency of a workflow specification. If Algorithm 3 returns *FALSE*, we conclude that the workflow specification has potential resource conflict(s). However, Algorithm 3 does not take into account time information in the workflow specification. So, some consistent resource dependencies are wrongly classified as potential resource conflicts.

Example 2. Reconsider the potential resource conflict between a_3 and a_4 in Example 1. Suppose that the possible active intervals of activity a_3 in all workflows are within the 4th to 6th time unit (relative to the start time of the workflows), and the possible active intervals of activity a_4 are within the 7th to 10th time unit. Therefore, the active intervals of a_3 and a_4 do not intersect in all workflows. In this circumstance, although accessing

the same resource (see Fig. 2), they have no resource conflicts in all workflow cases. Their execution order is implied in the workflow specification, that is, $a_3 \prec a_4$.

Therefore, in order to identify the potential resource conflicts more accurately, we further extend the verification to timed workflow specifications in this section.

5.1. Estimated active interval

Suppose that a potential resource conflict between two activities a_i and a_j is identified by Algorithm 3 in a workflow specification, we may check further whether their active intervals intersect in a workflow. If $[S(a_i), E(a_i)] \cap [S(a_j), E(a_j)] = \phi$ in all workflows, we can claim that no potential resource conflict exists between them because they do not compete for the same resources in all workflow cases.

Note that an activity’s active interval can be determined only if it has finished its execution. Given two activities with a resource dependency, if they have finished in a workflow, we can conclude whether their active intervals intersect in the workflow. However, when these two activities have committed and a resource conflict is identified in the workflow, then it is too late for any preventive or corrective action. Therefore, it is necessary to determine the potential resource conflicts in a workflow specification at build-time. To this end, we should estimate each activity’s active interval in all workflows. Given a workflow specification ws as Definition 1 with time information, we have:

Definition 10 (Earliest start time). The earliest start time of activity a , $EST(a)$, is its start time relative to reference point P (defined in Section 2.2) under the condition where $(\forall a_i \in AN)$ such that $D_R(a_i) = d(a_i)$.

Definition 11 (Latest end time). The latest end time of activity a , $LET(a)$, is its end time relative to P under the condition where $(\forall a_i \in AN)$ such that $D_R(a_i) = D(a_i)$.

Definition 12 (Estimated active interval). The estimated active interval of activity a is the period from $EST(a)$ to $LET(a)$, where $EST(a) < LET(a)$, denoted as $[EST(a), LET(a)]$.

From the above definitions, we conclude that:

Corollary 3. For any activity $a \in AN$, the active interval $[S(a), E(a)]$ is within the estimated active interval $[EST(a), LET(a)]$ in any workflow, that is, $[S(a), E(a)] \subseteq [EST(a), LET(a)]$.

Proof. The corollary can be derived directly from Corollary 1 and Definitions 10–12. \square

Corollary 4. A workflow specification ws is resource consistent if, for any two activities $a_i, a_j \in AN$ with a resource dependency, their estimated active intervals do not intersect, that is, $([EST(a_i), LET(a_i)] \cap [EST(a_j), LET(a_j)]) = \phi$.

Proof. Given any two activities $a_i, a_j \in AN$ with a resource dependency in ws , if $[EST(a_i), LET(a_i)] \cap [EST(a_j), LET(a_j)] = \phi$, we then have $([S(a_i), E(a_i)] \cap [S(a_j), E(a_j)]) = \phi$ from Corollary 3. Since a_i, a_j are any two activities in ws , from Definitions 4 and 5, the corollary holds. \square

5.2. Calculation of estimated active intervals

According to the above definitions, we have $EST(a_s) = 0$ and $LET(a_s) = D(a_s)$ (a_s is the start activity of a workflow specification). Given the ESTs and LETs of activities a_i and a_k , the EST and LET of a_j 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 a sequence. In a timed workflow specification, if there is a sequential connection between activities a_i and a_j (see Fig. 4(a), denoted as $a_i \cdot a_j$), $EST(a_j)$ and $LET(a_j)$ are calculated as follows:

$$EST(a_j) = EST(a_i) + d(a_i);$$

$$LET(a_j) = LET(a_i) + D(a_j).$$

An and-split connection is defined as a single thread of control splitting into two or more parallel activities. As shown in Fig. 4(b), activities a_j and a_k are the and-split successors of activity a_i , denoted as $a_i \cdot (a_j \wedge a_k)$. The four time values can be calculated below:

$$EST(a_j) = EST(a_i) + d(a_i); \quad EST(a_k) = EST(a_i) + d(a_i);$$

$$LET(a_j) = LET(a_i) + D(a_j); \quad LET(a_k) = LET(a_i) + D(a_k).$$

An and-join connection is defined as two or more parallel executing activities converging into a single common thread of control. Fig. 4(c) shows an and-join connection, where activity a_j executes after the completion of activities a_i and a_k , denoted as $(a_i \wedge a_k) \cdot a_j$. $EST(a_j)$ and $LET(a_j)$ are calculated as follows:

$$EST(a_j) = \text{MAX}\{EST(a_i) + d(a_i), EST(a_k) + d(a_k)\};$$

$$LET(a_j) = \text{MAX}\{LET(a_i), LET(a_k)\} + D(a_j).$$

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. 4(d), activities a_j and a_k are the or-split successors of activity a_i , denoted as $a_i \cdot (a_j \vee a_k)$. The four time values are calculated as follows:

$$EST(a_j) = EST(a_i) + d(a_i); \quad EST(a_k) = EST(a_i) + d(a_i);$$

$$LET(a_j) = LET(a_i) + D(a_j); \quad LET(a_k) = LET(a_i) + D(a_k).$$

An or-join connection is defined as two or more activities workflow branches re-converging into a single thread of control without any synchronisation. As shown in Fig. 4(e), where activity a_j executes after the completion of activity a_i or a_k , denoted as $(a_i \vee a_k) \cdot a_j$. $EST(a_j)$ and $LET(a_j)$ are calculated as follows:

$$EST(a_j) = \text{MIN}\{EST(a_i) + d(a_i), EST(a_k) + d(a_k)\};$$

$$LET(a_j) = \text{MAX}\{LET(a_i), LET(a_k)\} + D(a_j).$$

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. 5(a), activities a_j and a_k execute in parallel, denoted as $(a_j || a_k)$. $EST(a_j || a_k)$ and $LET(a_j || a_k)$ are calculated as follows:

$$EST(a_j || a_k) = EST(a_i) + d(a_i);$$

$$LET(a_j || a_k) = \text{MAX}\{LET(a_i) + D(a_j), LET(a_i) + D(a_k)\}.$$

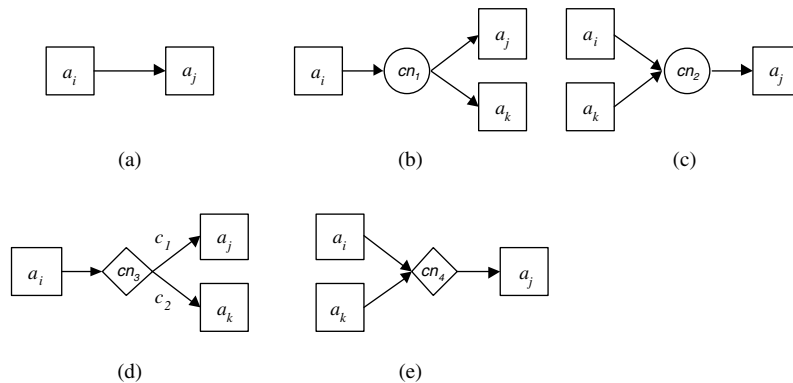


Fig. 4. Basic control structures in workflow specifications. (a) Sequential connection, (b) and-split connection, (c) and-join connection, (d) or-split connection and (e) or-join connection.

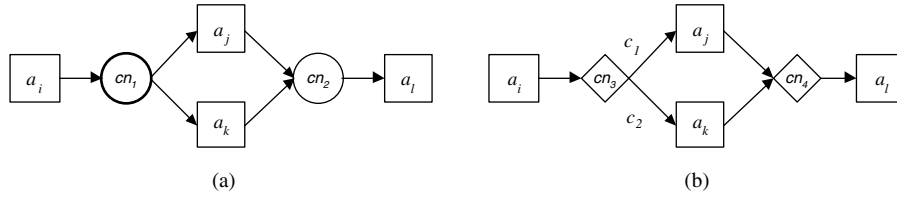


Fig. 5. Parallel and selective connections in workflow specifications. (a) Parallel connection and (b) selective connection.

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. 5(b), either activity a_j or a_k will be executed in a workflow, denoted as $(a_j|a_k)$. $EST(a_j|a_k)$ and $LET(a_j|a_k)$ are calculated as follows:

$$EST(a_j|a_k) = EST(a_i) + d(a_i);$$

$$LET(a_j|a_k) = \text{MAX}\{LET(a_i) + D(a_j), LET(a_i) + D(a_k)\}.$$

Given a timed workflow specification composed of the above control structures, we can design an algorithm to calculate the EST and LET of each activity at build-time. Here, the algorithm, being relatively straightforward, is omitted due to the space limit.

5.3. Relationship between estimated active intervals

From Corollary 3, we know that an activity is instantiated and executed over its estimated active

interval (assume the lengths of all intervals are non-zero). Considering a single time line, relations between two estimated active intervals can be formally described by Allen’s interval logic, which is composed of seven interval expressions (Zaidi, 1999; Chinn and Madey, 2000). As shown in Fig. 6, each is related to a primitive in $R = \{\text{before, meets, overlaps, starts, during, finishes, equals}\}$. The temporal relations between two estimated active intervals depicted in Fig. 6, are mutually exclusive and exhaustive, that is, given any two intervals, there exists a unique primitive in R to depict the relation between them (Zaidi, 1999).

5.4. Verification algorithm in timed workflow specifications

Now that an activity’s active interval cannot be obtained at build-time, we might use the estimated active interval to verify the resource consistency of a timed workflow specification. From Fig. 6, we know that:

1. X before Y	$LET(a_i) \leq EST(a_j)$	
2. X meets Y	$LET(a_i) = EST(a_j)$	
3. X overlaps Y	$EST(a_i) < EST(a_j)$ $EST(a_j) < LET(a_i)$ $LET(a_i) < LET(a_j)$	
4. X starts Y	$EST(a_i) = EST(a_j)$ $LET(a_i) < LET(a_j)$	
5. X during Y	$EST(a_j) < EST(a_i)$ $LET(a_i) < LET(a_j)$	
6. X finishes Y	$EST(a_j) < EST(a_i)$ $LET(a_i) = LET(a_j)$	
7. X equals Y	$EST(a_i) = EST(a_j)$ $LET(a_i) = LET(a_j)$	

$$X = [EST(a_i), LET(a_i)], Y = [EST(a_j), LET(a_j)]$$

Fig. 6. Allen’s temporal relations for estimated active intervals.

given two activities a_i and a_j , if the relation between their estimated active intervals is overlaps, starts, during, finishes, or equals in R , we say $[\text{EST}(a_i), \text{LET}(a_i)] \cap [\text{EST}(a_j), \text{LET}(a_j)] \neq \phi$. Therefore, if these two activities have a resource dependency, a potential resource conflict exists between them.

Example 3. Reconsider the potential resource conflict between a_3 and a_4 in Example 1. Suppose their estimated active intervals are $[4, 7]$ and $[5, 9]$ respectively, the relation between them can be described by *overlaps* in R (see Fig. 6). It is possible that there is a workflow in which the active intervals of a_3 and a_4 are $[5, 7]$ and $[6, 8]$ respectively. Then under this situation, they will compete for the same resource (see Fig. 2).

Therefore, in order to verify the resource consistency of a timed workflow specification, we should firstly calculate all activities' estimated active intervals. Then for each pair of activities with a resource dependency, we decide if a potential resource conflict exists between them through comparing their estimated active intervals. Algorithm 4 below is designed by revising Algorithm 3 to fulfil the functionality of verifying the resource consistency of a timed workflow specification. Similar to Algorithm 3, the input is a timed workflow specification. The algorithm returns *TRUE* indicating that the workflow specification is resource consistent, otherwise returns *FALSE* and prints all potential resource conflicts in it.

In Algorithm 4, if n_{nca} is an and-split node, then there is a workflow w such that $w(a_i * a_j)$. We distinguish the following two cases:

- (1) $[\text{EST}(a_i), \text{LET}(a_i)] \cap [\text{EST}(a_j), \text{LET}(a_j)] \neq \phi$: In this case, a potential resource conflict exists between a_i and a_j , corresponding to the scenario in Example 3.
- (2) $[\text{EST}(a_i), \text{LET}(a_i)] \cap [\text{EST}(a_j), \text{LET}(a_j)] = \phi$: According to Definition 4 and Corollary 3, no potential resource conflict exists between a_i and a_j , corresponding to the scenario in Example 2.

In a workflow specification, for any two activities a_i and a_j with a resource dependency, if $([\text{EST}(a_i), \text{LET}(a_i)] \cap [\text{EST}(a_j), \text{LET}(a_j)]) = \phi$, according to Corollary 4, we can conclude that this workflow specification is resource consistent.

Algorithm 4. Check the resource consistency of a timed workflow specification

1. Initialisation:
 - 1.1 let S be a set of unchecked activities. S is initialised with all activities in AN;
 - 1.2 calculate all activities' ESTs and LETs;
 - 1.3 $b = \text{TRUE}$ is a variable;

2. Repeat the following steps until $S = \text{NULL}$
 - 2.1 remove an element from S , denoted as a_i ;
 - 2.2 for each element a_j in S , execute the following steps:
 - 2.2.1 if $\mathfrak{R}(a_i) \cap \mathfrak{R}(a_j) = \phi$, skip to the next iteration;
 - 2.2.2 if $\text{Reachable}(a_i, a_j) = \text{True}$ (by executing Algorithm 1), skip to the next iteration;
 - 2.2.3 if $\text{Reachable}(a_j, a_i) = \text{True}$ (by executing Algorithm 1), skip to the next iteration;
 - 2.2.4 invoke Algorithm 2 to compute the nearest common ancestor of a_i and a_j , n_{nca} ;
 - 2.2.5 if n_{nca} is an or-split node, skip to the next iteration;
 - 2.2.6 if n_{nca} is an and-split node and $[\text{EST}(a_i), \text{LET}(a_i)] \cap [\text{EST}(a_j), \text{LET}(a_j)] \neq \phi$, then /*the detailed information on resource conflict can be derived.*/
 - 2.2.6.1 print("Potential resource conflict between:", a_i , "and", a_j);
 - 2.2.6.2 $b = \text{FALSE}$;
3. Return b ;

6. Removal of potential resource conflicts from workflow specifications

As stated previously, for a workflow specification, Algorithm 3 or 4 returns *FALSE* indicating that it contains potential resource conflict(s). Then these potential resource conflicts should be removed from the workflow specification. In this section, we address this problem by suggesting several ways to solve potential resource conflicts in a workflow specification as follows:

6.1. Remove conflicted resources from activities

As discussed earlier, a resource conflict is caused by the competition for the same resources between two activities. Therefore, removal of those resources from one of or all the activities can resolve the potential resource conflict in the workflow specification. For example, as shown in Fig. 2, a potential resource conflict exists between activities a_3 and a_4 due to accessing the same resource c . If c is removed from a_4 , the potential resource conflict is thereby eliminated from the workflow specification.

It should be noted that although a potential resource conflict can be resolved by using this method, it probably makes the workflow semantics incorrect. For example, as shown in Fig. 2, after c is removed from a_4 , the activity would lose its original semantics. Therefore, sometimes we cannot solve a potential resource conflict by simply removing some resources from activities.

6.2. Adjust time information of activities

In a timed workflow specification, a potential resource conflict can be solved through adjusting some activities' minimum and maximum durations. Such an adjustment ensures that the estimated active intervals do not intersect for those activities with a potential resource conflict. After the adjustment, their active intervals will not overlap in all workflow cases. Then the execution order is implicitly predefined in the workflow specification. For example, Fig. 7(a) shows a graphic representation of a timed workflow specification, where two numbers in a pair of square brackets above an activity denote the minimum (left) and maximum (right) durations respectively. As discussed in Section 5.2, the estimated active intervals of a_2 and a_4 are calculated as [2, 10] and [5, 12] respectively. Then, a potential resource conflict exists between them. Now, we can solve this potential resource conflict by adjusting time information in the workflow specification. The result is shown in Fig. 7(b), where the minimum and maximum durations of a_2 and a_3 are adjusted as [3, 4] and [6, 8] respectively. Hence, the estimated active intervals of a_2 and a_4 change accordingly to be [2, 8] and [8, 15] respectively. According to Definition 4 and Corollary 3, the potential resource conflict no longer exists because the execution order between a_2 and a_4 is implied in the workflow specification, that is, $a_2 < a_4$.

This method by adjusting the time information cannot resolve all potential resource conflicts in the workflow specification. Similar to the above, it probably makes the activities losing their original semantics.

6.3. Add null activities to the workflow specification

A null activity does not execute any operation at runtime, but occupies a period of time. So, we can add null activities to a timed workflow specification to resolve potential resource conflicts between activities. These operations are added to make sure that the estimated active intervals of those activities with potential resource conflicts no longer intersect. Then their active intervals will not overlap in all workflow cases. The execution orders between these activities are also implicitly predefined in the workflow specification. For example,

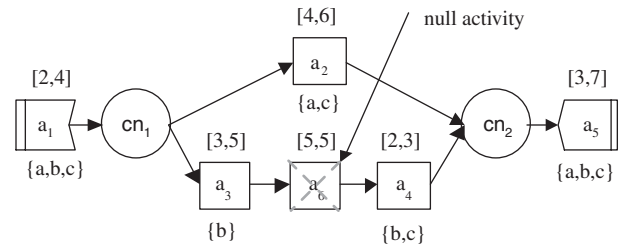


Fig. 8. Adding a null activity to a timed workflow specification.

reconsidering the timed workflow specification depicted in Fig. 7(a), a potential resource conflict exists between activities a_2 and a_4 . Now, we can add a null activity (see a_6 in Fig. 8) to the workflow specification. The minimum and maximum durations of a_6 are the same, that is, five time units. Then the estimated active interval of a_4 changes accordingly to be [10, 17]; and the estimated active interval of a_2 remains the same value, that is, [2, 10]. According to Definition 4 and Corollary 3, the potential resource conflict no longer exists between them.

6.4. Modify some flows in the workflow specification

In this way, a potential resource conflict between two activities is resolved through modifying some flows in the workflow specification. Such modification makes that these activities do not execute concurrently in all workflow cases. Then, the potential resource conflict no longer exists due to the precedence relationship between them. For example, to remove the potential resource conflicts between activities a_2 and a_4 , a_4 and a_6 in Fig. 1, we can change some flows in the workflow specification to ensure that they do not execute concurrently in all workflow cases. The modified workflow specification is graphically presented in Fig. 9. After the modification, we have $a_2 < a_4$ and $a_4 < a_6$ in all workflow cases.

7. Related work

The work on workflow analysis mainly includes three aspects: workflow verification (Hofstede and Orłowska, 1999), workflow simulation (Tarumi et al., 1999), and

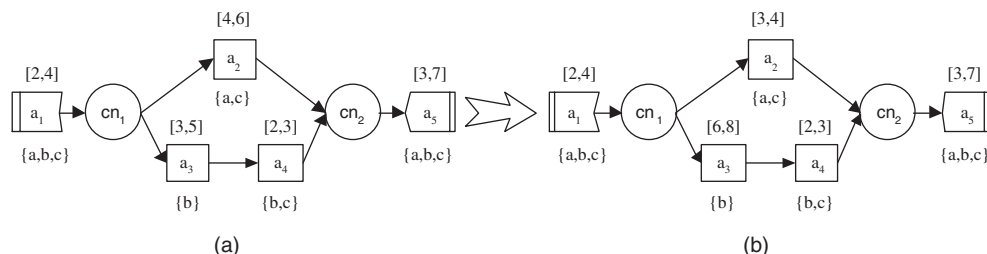


Fig. 7. Adjusting time information in a timed workflow specification.

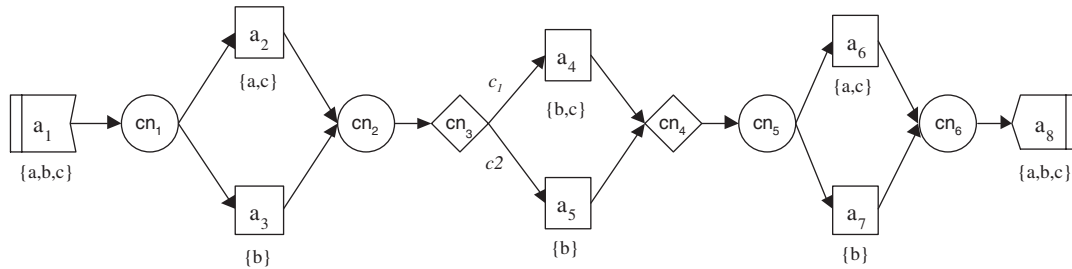


Fig. 9. Representation of the modified workflow specification.

performance analysis of workflows (Kim and Ellis, 2001). Resource constraints analysis discussed in this paper is fallen into the workflow verification. Due to the space limit, this section only reviews closely related work on workflow verification.

The workflow verification aims to establish the correctness of workflow specifications. Most references found in our literature review are about the analysis on process structure of a workflow specification, while others about the verification of temporal constraints.

7.1. Structural constraints analysis

The structure of a workflow specification defines the order by which activities would be scheduled in a workflow. It is the primary and most important aspect of the workflow specification, building a foundation for capturing other aspects of the workflow specification. The purpose of structural constraints analysis tries to verify the structural correctness of a workflow specification. Such verification is usually based on an extension of a kind of formal method, for example, directed graph, petri net, algebra, or temporal logic.

The authors in Sadiq and Orlowska (2000) and Onoda et al. (1999) employ directed graphs to specify workflows. In a directed graph, a variety of symbols represent entities in a workflow specification, for example, nodes and arcs representing activities and control flows, respectively. The structural correctness of a workflow specification can be verified through analysing this directed graph. Some structural conflicts can be identified in the workflow specification, such as deadlock, lack of synchronisation.

Petri nets can be used to specify workflows due to their formal semantics. In Aalst (1998) and Adam et al. (1998), a workflow is mapped onto a petri net, where activities are mapped by transitions, and dependencies between activities are mapped by places and arcs. Then, the structural correctness of the workflow specification can be verified through analysing the petri net.

The algebra can also be employed to verify workflows (Hofstede and Orlowska, 1999; Karamanolis et al., 1999; Singh, 1997). Hofstede and Orlowska (1999) introduces algebra of communicating processes with the

empty action (ACP_ϵ) to specify workflows; Karamanolis et al. (1999) exploits Labelled Transition Systems (LTS: a kind of process algebra) to model workflows; and Singh (1997) introduces event algebra to model workflows. In a workflow specification, activities are marked by symbols, and flows between these activities by operators. Similarly, the structural properties of the workflow specification are verified through analysing these algebra expressions. In addition, the temporal logic is used to describe workflows in Davulcu et al. (1998). Like the previous models, workflows are mapped onto logic expressions. The verification on the structural correctness of a workflow specification is through analysing these logic expressions.

7.2. Temporal constraints analysis

In reality, activities execute along the time dimension. Therefore, temporal constraint is an important attribute of workflow specifications; and time management is a critical component in WfMSs. The analysis on temporal constraints is to verify the temporal consistency of a workflow specification.

Eder et al. (1999) defines a timed workflow graph through augmenting each activity node with two values: the earliest end time and the latest end time. Temporal constraints (including fixed-date constraints, lower-bound constraints and upper-bound constraints) can be calculated using the modified critical path method (CPM) method at build-time and process instantiation time, and then enforced at run-time. In Zhao and Stohr (1999), Zhao and Stohr develop a framework for temporal workflow management in the context of a claim handling system. Based on the framework, issues including the prediction of turnaround time, time allocation policy and task prioritisation policy are discussed in that paper.

Marjanovic (2000) assumes 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, the authors presents a method for dynamic verification of absolute deadline constraints

and relative deadline constraints. Zhuge et al. (2001) differs from the previous work 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. In addition, Adam et al. (1998) employs temporal constraint petri nets (TCPN) to specify workflows, and then tests the temporal feasibility for a workflow at build-time.

As far as the workflow verification is concerned, the structural correctness is essential to workflow specifications. Analyses on temporal constraints and resource constraints are based on the structural correctness because these constraints are implied in the semantics of workflow specifications.

8. Conclusions and future work

Before a workflow specification is deployed and put into operation, it should be verified for its correctness. However, a workflow specification contains information for representing various aspects of a workflow, which makes the workflow verification being a complex process in workflow management. The previous work on workflow analysis is mainly about the verification of structural correctness and temporal consistency. However, resource constraints are implied in the semantics of a workflow specification and may influence the state or the result of some or all workflows. A correct workflow specification should have resource consistency.

This paper discusses the analysis on resource constraints of a workflow specification and presents a new approach with corresponding algorithms for the resource consistency of the workflow specification. The main contribution of this paper is threefold. First, the problem of resource constraints in a workflow specification is identified. We use a practical example to illustrate the resource constraints and the erroneous consequences caused by them. Second, the checking method and its corresponding algorithms for resource consistency of a workflow specification are presented. Third, we extend our analysis work to timed workflow specifications, where the checking method for the resource consistency is revised with respect to time information in a workflow specification.

However, this paper presents a static analysis technique on workflow specifications at build-time, not considering dynamic analysis on workflows at run-time. In fact, in a workflow system, there are multiple workflows executing concurrently. Activities belonging to different workflows may access the same resources as

well. A resource conflict occurs when these activities execute over the same time interval. So, our future work is to analyse resource constraints between concurrent workflows and check dynamically the resource and temporal consistency in an environment with concurrent workflows.

Acknowledgement

The work reported in this paper has been supported in part by Swinburne Vice Chancellor's Strategic Research Initiative Grant (2002–2004).

References

- Aalst, W.M.P., 1998. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers* 8 (1), 21–66.
- Adam, N., Atluri, V., Huang, W., 1998. Modeling and analysis of workflows using petri nets. *Journal of Intelligent Information Systems, Special Issue on Workflow and Process Management* 10 (2), 131–158.
- Chan, D., Vonk, J., Sanchez, G., Grefen, P., Apers, P., 1997. A specification language for the WIDE workflow model. In: *Proceedings of the 1998 ACM symposium on Applied Computing*, Atlanta, USA, pp. 197–199.
- Chinn, S., Madey, G., 2000. Temporal representation and reasoning for workflow in engineering design change review. *IEEE Transactions on Engineering Management* 47 (4), 485–492.
- Davulcu, H., Kifer, M., Ramakrishnan, C., Ramakrishnan, I., 1998. Logic based modeling and analysis of workflows. In: *Proceedings of ACM/SIGMOD Symposium on Principles of Database Systems (PODS'98)*, Seattle, WA, USA, pp. 25–33.
- Eder, J., Panagos, E., Rabinovich, M., 1999. Time constraints in workflow systems. In: *Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE'99)*. Springer Verlag, LNCS 1626, Germany, pp. 286–300.
- Hofstede, A., Orłowska, M., E., 1999. On the complexity of some verification problems in process control specifications. *The Computer Journal* 42 (5), 349–359.
- Karamanolis, C., Giannakopoulou, D., Magee, J., Wheeler, S., 1999. Modeling and analysis of workflow processes. Technical Report 99/2, Department of Computing, Imperial College.
- Kim, K., Ellis, C.A., 2001. Performance analytic models and analyses for workflow architectures. *Journal of Information Systems Frontiers* 3 (3), 339–355.
- Marjanovic, O., 2000. Dynamic verification of temporal constraints in production workflows. In: *Proceedings of the Australian Database Conference*. IEEE Press, Canberra, Australia, pp. 74–81.
- Onoda, S., Ikkai, Y., Kobayashi, T., Komoda, N., 1999. Definition of deadlock patterns for business processes workflow models. In: *Proceedings of the 32nd Hawaii International Conference on System Sciences*, pp. 1–11.
- Sadiq, W., Orłowska, M.E., 2000. Analysing process models using graph reduction techniques. *Information Systems* 25 (2), 117–134.
- Schuldt, H., Alonso, G., Beeri, C., Schek, H., 2002. Atomicity and isolation for transactional processes. *ACM Transactions on Database Systems* 27 (1), 63–116.
- Singh, M.P., 1997. Formal aspects of workflow management, Part 1: Semantics. Technical Report, Department of Computer Science, North Carolina State University.

- Tarumi, H., Matsuyama, T., Kamabayashi, Y., 1999. Evolution of business processes and a process simulation tool. In: *Proceedings of Asia Pacific Software Engineering Conference (APSEC'99)*, pp. 180–187.
- WFMC, 1995. Workflow management coalition: the workflow reference model. TC00-1003. Available from <<http://www.wfmc.org>>.
- Zaidi, A., 1999. On temporal logic programming using petri nets. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans* 29 (3), 245–254.
- Zhao, J., Stohr, E., 1999. Temporal workflow management in a claim handling system. In: *Proceedings of Work Activities Coordination and Collaboration (WACC'99)*, San Francisco, CA, USA, pp. 187–195.
- Zhuge, H., Cheung, T., Pung, H., 2001. A timed workflow process model. *The Journal of Systems and Software* 55 (3), 231–243.

Hongchen Li received the B.E. and M.E. degrees from Henan University and Northern Jiaotong University in 1992 and 1997 respectively, and the PhD degree from Tsinghua University, China in 2002. He is presently a post-doctoral scholar in the School of Information Technology, Swinburne University of Technology, Australia. His re-

search interests include CSCW (computer supported cooperative work), workflow systems, distributed object computing and advanced transaction management.

Yun Yang is an Associate Professor and foundation director of the CICEC at Swinburne University of Technology, Melbourne, Australia. He is also the deputy head (research) of School of Information Technology at Swinburne University of Technology. He received a Ph.D. degree in Computer Science from the University of Queensland, Brisbane, Australia. His current research areas include software technology, workflow systems, Internet computing applications, CSCW and e-business processes. He has edited one book and co-authored about seventy papers on international journals and conferences. He is now a member of IEEE and IEEE Computer Society.

T.Y. Chen received the BSc and MPhil degrees from the University of Hong Kong, MSc degree and DIC from the Imperial College of Science and Technology, and the PhD degree from the University of Melbourne. He is currently the Professor of Software Engineering in the School of Information Technology, Swinburne University of Technology, Australia. His research interests include software testing, debugging, software maintenance, and software design. He is a member of the Editorial Board of *Software Testing, Verification and Reliability*.