Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 81 (2008) 1770–1783
The Journal of Systems and Software
www.elsevier.com/locate/jss

# An incremental analysis for resource conflicts to workflow specifications

Hwai-jung Hsu *, Feng-jian Wang

*Institute of Computer Science and Engineering, College of Computer Science, National Chiao Tung University, Room 510, EC Building, 1001 Ta-Hsueh Road, Hsinchu City, Taiwan, ROC*

Received 2 May 2006; received in revised form 7 December 2007; accepted 27 December 2007
Available online 10 January 2008

**Abstract**

Workflow management technology helps modulizing and controlling complex business processes within an enterprise. Generally speaking, a workflow management system (WfMS) is composed of two primary components, a design environment and a run-time system. Structural, timing and resource verifications of a workflow specification are required to assure the correctness of the specified system. In this paper, an incremental methodology is constructed to analyze resource consistency and temporal constraints after each edit unit defined on a workflow specification. The methodology introduces several algorithms for general and temporal analyses. The output returned right away can improve the judgment and thus the speed and quality on designing.
© 2008 Elsevier Inc. All rights reserved.

*Keywords:* Workflow; Workflow specification; Resource consistency; Temporal constraints; Incremental methodology

## 1. Introduction

Electronic workflow integrates business rules and staffs inside an enterprise into an automatic information system. Inside a flow, the processes (activities) and information flow between them are specified according to the business rules which accomplish specific tasks (WfMC, 1993; Hollingsworth, 1995). Furthermore, workflow specifications which record tasks and schedules are enacted by the workflow management system (WfMS) to coordinate human resources and information (Hollingsworth, 2004). In a modern WfMS, the environment for workflow design and enactment are usually supported at the same time.

To assure the correctness of executing a workflow specification, analyses on structural integrity, temporal correctness, and resource conflicts are required. Aalst gives effective Petri-net based techniques in structural analysis (van der Aalst et al., 1999; van der Aalst and ter Hofstede, 2000; van der Aalst, 1998). Eder focuses on temporal analysis in workflow models (Eder et al., 1999a,b). Reveliotis constructs a Petri-based model with consideration of resource allocation, and uses the model in structural and deadlock analysis in workflow application (Reveliotis, 2003; Park and Reveliotis, 2001). There are still various methodologies for structural and temporal analysis of workflow system specifications which have been developed and proved effective (Sadiq and Orlowska, 2000; Fleurke et al., 2003; Adam et al., 1998; Onoda et al., 1999; Singh, 1997; Li et al., 2004b; Eder et al., 1999a,b; Had et al., 2005).

In Li et al. (2004b), Reveliotis (2003), Park and Reveliotis (2001), Sun et al. (2006), Tang et al. (2004) and Russell et al. (2004), the analysis of resource conflicts in workflow is widely discussed. In Li et al. (2004a), Li and Yang (2005), Hsu et al. (2005) and Zhong and Song (2005), the methodologies to analyze both resource and temporal constraints are considered. In Li et al. (2004a), a model of analyzing resource and temporal constraints in workflow specification has been constructed. In Hsu et al. (2005) and Zhong and Song (2005), the model is applied with Petri-net based workflow and DAG (Directed Acyclic Graph). In Zhong

* Corresponding author. Tel.: +886 3 571 2121x54718; fax: +886 3 5724176.
*E-mail addresses:* hjhsu@cs.nctu.edu.tw, hjhsu@csie.nctu.edu.tw (H.-j. Hsu), fjwang@cs.nctu.edu.tw (F.-j. Wang).

and Song (2005), the model is extended for dynamic analysis. The analysis discussed in Zhong and Song (2005) is used for workflow enactment systems and the analysis discussed in Li et al. (2004a), Li and Yang (2005) and Hsu et al. (2005) is used for workflow design environments.

Among all the researches above, the relationships between methodologies and edit operations are not clearly defined and discussed. Some methodologies, e.g. the algorithms in Hsu et al. (2005) are also required to be refined. More discussions are necessary to clarify the information required for workflow designer in editing or maintaining a workflow.

In this paper, the incremental methodology for analysis of resource constraints in well structuralized workflow specifications mentioned in Hsu et al. (2005) is refined or rebuilt. First, the edit operations made by workflow designer are discussed. The workflow model for analyzing resource conflicts with temporal considerations is then defined completely. Finally, the analysis algorithms are constructed; their correctness and time complexities are carefully discussed.

This paper is organized as following: The workflow model and its notations are defined in Section 2. In Section 3, the definition of resource conflicts for analysis is depicted. The edit operations for workflow design are described in Section 4. The incremental algorithms for analyzing the resource conflicts are represented in Section 5. Proofs and time complexity discussions of the algorithms are also included in this section. In Section 6, the relationships between the operations and the algorithms are discussed. Several examples about how the algorithms work are described in Section 7 and the conclusions are made in Section 8.

## 2. Definitions and notations

In this paper, a workflow specification is a graph-based model. The workflow specification is formally defined as Definition 1.

**Definition 1** (*Workflow specification*).

A workflow specification $ws = (N, F, R, C, S, E)$.
$N$ is the set of processes, $F$ is the set of flows, $R$ is the set of resources, $C$ is the set of control blocks, $S$ is the start process and $E$ is the end process.
$S,\ E \in N,\ S.type = \text{START},\ E.type = \text{END},\ S.cb = E.cb = \text{ROOT}$.

The nodes represent processes in a workflow specification, and the directed arcs for flows. The processes can be classified as activity or control processes. Control processes are further categorized as AND-SPLIT, AND-JOIN, XOR-SPLIT, and XOR-JOIN WfMC (1999). A control block records the processes quoted by a pair of split and join processes. The start/end processes (Kim, 2003) and the resources are externally modeled.

**Definition 2** (Processes, flows and resources).

$\forall$ process $n \in N$, $n = (type, ref, cb)$
  $n.type = \{\text{ACT}, \text{AS}, \text{AJ}, \text{OS}, \text{OJ}, \text{START}, \text{END}\}$
  $n.ref = \{r | r \in R$ and $n \in r.use\}$
  $n.cb = \{\text{ROOT},\ c |\ c \in C$ and $n \in c.p\_set\}$
$\forall$ flow $f \in F$
  $f = (n_i, n_j),\ n_i, n_j \in N$
  $f$ is an **in-flow**of $n_j$ and an **out-flow** of $n_i$
  $n_i$ is the source process of the $f$, and $n_j$ is the sinking process of $f$
$\forall$ resource $r \in R$
  $r.use = \{n | n \in N$ and $r \in n.ref\}$

Definition 2 describes the properties of processes, flows and resources in more detail. An activity process is simply denoted as type "ACT", and-split as "AS", and-join as "AJ", xor-split as "OS", xor-join as "OJ", start as "START", and end as "END". The control block which a process $n$ belongs to is recorded as $n.cb$. A process must belong to some control block or is on the path which is never split or has been totally joined. $n.cb$ is recorded as "ROOT" if process $n$ belonging to no control blocks. Such processes are said to be on the root path in this paper. The resources referenced by some process $n$ are recorded in set $n.ref$ and the processes accessing some resource $r$ are also recorded in property $r.use$.

**Definition 3** (Control blocks).

$\forall$ control block $c \in C$, $c = (start, end, p\_set)$
  $c.begin = n_s, n_s.type = \{\text{AS}, \text{OS}\}$
  $c.end = n_e, n_e.type = \text{AJ}$ if and only if $n_s.type = \text{AS}$, $n_e.type = \text{OJ}$ if and only if $n_s.type = \text{OS}$
  $c.p\_set = \{n | n \in N,\ Reachable\ (n_s, n) = \text{true},\ Reachable(n, n_e) = \text{true},\ n.cb = c$ and $\forall\ c' \notin C$ and $\forall\ c' \in c,\ n \notin c'.p\_set\}$
  $Reachable(c.start, c.end) = \text{true}$

The properties of a control block are defined in Definition 3. A control block starts from a split process and ends at a join process. The processes directly contained in some block $c$ are included in $c.p\_set$(process set).

In this paper, not only constraints between resources and workflow but also temporal factors are considered. Designers/Maintainers of the workflow specification can set the maximal/minimal working duration (Li et al., 2003, 2004a,b; Marjanovic, 2000; Li and Yang, 2005; Hsu et al., 2005; Zhong and Song, 2005; Ling and Schmidt, 2000; Zaidi, 1999) of each activity process for timing control. The notations and constraints of working durations are defined in Definition 4.

**Definition 4** (Working durations).

$\forall$ process $n \in N$
  $d(n)$ is the minimal working duration of $n$
  $D(n)$ is the maximal working duration of $n$
  If $n.type \neq \{\text{ACT}\}, d(n) = D(n) = 0$

Through the working durations, the earliest start time (EST) and the latest end time (LET) (Li et al., 2004a; Li and Yang, 2005; Hsu et al., 2005; Zhong and Song, 2005) of a process can be calculated. The calculation of EST and LET for each process is conceptually shown in Fig. 1 and is formally defined in Definitions 5 and 6.

**Definition 5** (*Earliest start time*).

$\forall$process $n \in N$
If $n.type \neq \{\mathrm{AJ}, \mathrm{OJ}, \mathrm{START}\}$, there must exist a flow $(n', n)$ such that
$EST(n) = EST(n') + d(n')$
If $n.type = \mathrm{START}$, $EST(n) = 0$
If $n.type = \mathrm{AJ}$, there must exist one or more flows $(n_1, n), \ldots, (n_k, n)$ such that
$EST(n) = \mathrm{MAX}(\{EST(n_i) + d(n_i) | i = 1..k\})$
If $n.type = \mathrm{OJ}$, there must exist one or more flows $(n_1, n), \ldots, (n_k, n)$ such that
$EST(n) = \mathrm{MIN}(\{EST(n_i) + d(n_i) | i = 1..k\})$

MAX( ) is a function which will return the element with the largest value among the input set; on the contrary, and MIN( ) returns the minimal one.

**Definition 6** (Latest end time).

$\forall$ process $n \in N$
If $n.type \neq \{\mathrm{AJ}, \mathrm{OJ}, \mathrm{START}\}$, there must exist a flow $(n', n)$ such that
$LET(n) = LET(n') + D(n')$
If $n.type = \mathrm{START}$, $LET(n) = 0$
If $n.type = \{\mathrm{AJ}, \mathrm{OJ}\}$ there must exist one or more flows $(n_1, n), \ldots, (n_k, n)$
such that
$LET(n) = \mathrm{MAX}(\{LET(n_i) + D(n) | i = 1..k\})$

To some process $n$, the time interval between $EST(n)$ and $LET(n)$ is the estimated active interval (EAI) of $n$ (Li et al., 2004a; Li and Yang, 2005; Hsu et al., 2005; Zhong and Song, 2005). In Definitions 7 and 8, the definition of time intervals and the notation of EAI are depicted.

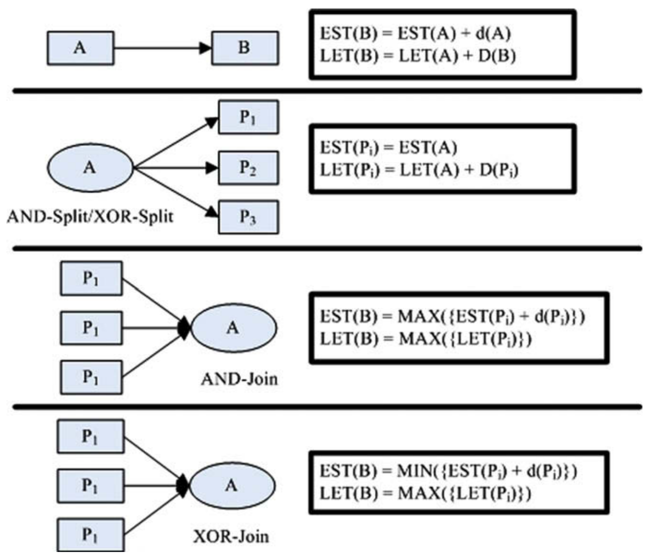


Fig. 1. How to calculate EST and LET for processes.

**Definition 7.** (Time Intervals)

A time interval from $t_a$ to $t_b$ is denoted as $[t_a, t_b]$, in which $t_b \leqslant t_a$.
The operator $\otimes$ for two time intervals is defined as follows:
$[t_{a1}, t_{a2}] \otimes [t_{b1}, t_{b2}] = \mathrm{MIN}(\{t_{a2}, t_{b2}\}) - \mathrm{MAX}(\{t_{a1}, t_{b1}\})$
Relations between time intervals:

(1) two time intervals are overlapped if and only if $[t_{a1}, t_{a2}] \otimes [t_{b1}, t_{b2}] > 0$;
(2) two time intervals are met if and only if $[t_{a1}, t_{a2}] \otimes [t_{b1}, t_{b2}] = 0$;
(3) two time intervals are exclusive if and only if $[t_{a1}, t_{a2}] \otimes [t_{b1}, t_{b2}] < 0$.

The relationships between time intervals are concluded by Allen's temporal reasoning relations (Allen, 1983) as Fig. 2 shows. Allen's temporal reasoning relations have been applied for the analysis of temporal factors in many researches (Li and Yang, 2005; Chen et al., 2004; Zhong and Song, 2005; Chinn and Madey, 2000; Ling and Schmidt, 2000; Zaidi, 1999).

In this paper, relation 1 in Allan's chart is equal to relation (3) in Definition 7, relation 2 in Allan's chart is equal to relation (2), and relations 3 to 7 in Allan's chart are equal to relation (1).

With Definition 7, definition and notation of EAI is represented as following:

**Definition 8** (*Estimated active interval*).

Estimated active interval of some process $n$, is the time interval from $EST(n)$ to $LET(n)$, the statement is formally described as follows:
$\forall$ process $n \in \mathrm{N}$, $EAI(n) = [EST(n), LET(n)]$

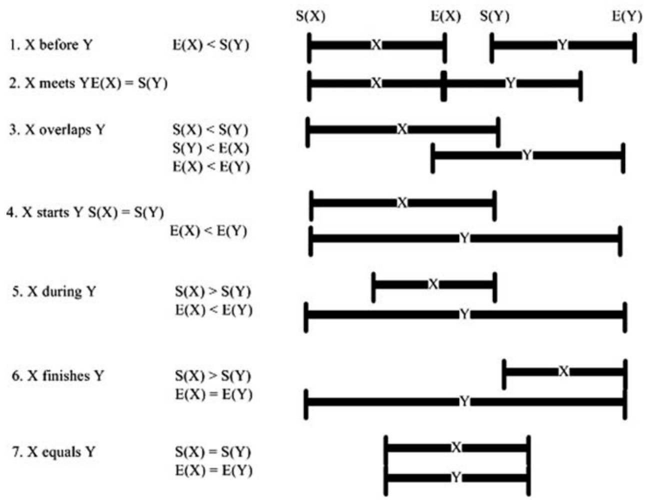


Fig. 2. Allen's temporal reasoning relations for intervals (Allen, 1983).

To clear the description of our algorithm, paths, reachability, distance, and ancestors (Li et al., 2004a; Li and Yang, 2005; Hsu et al., 2005; Zhong and Song, 2005) are defined in following definitions.

**Definition 9** (*Paths*). A path $p = (n_1, \ldots, n_k)$ in which $k \geqslant 2$, $i = 1..k$, $n_i \in N$; $j = 1..k-1$, $(n_j, n_{j+1})$ $F, p$ can be denoted as $Path(n_1, n_k)$ in brief. The size of path $p$, denoted as $|p| = k - 1$

**Definition 10** (Reachability).

$\forall$ processes $n_i$, $n_j \in N$
  $Reachable(n_i, n_j)$ is a Boolean function. $Reachable(n_i, n_j) =$ true if and only if $Path(n_i, n_j)$ exists
$Reachable(n, n) =$ true

**Definition 11** (*Distance*).

$\forall$ processes $n$, $n' \in N$
  if $Reachable(n, n') =$ false, $Dist(n, n') = \infty$
  otherwise, $Dist\ (n, n') =$ MIN ({size of path $p | p$ is a path from $n$ to $n'$})
$Dist(n, n) = 0$

**Definition 12** (Ancestors and common ancestors).

$\forall$ processes $n_i, n_j \in N$
  $n_i$ is an **ancestor** of $n_j$ if and only if $Reachable(n_i, n_j) =$ true
$\forall$ processes $n, n_1, \ldots, n_k \in N$, $k \geqslant 2$
  $n$ is a **common ancestor** of $n_1, n_2, \ldots, n_k$ if and only if $Reachable(n, n_i) =$ true, for $1 \leqslant i \leqslant k$

**Definition 13** (The nearest common ancestor).

$\forall$ processes $n$, $n' \in N$, $n.type =$ ACT and $n'.type =$ ACT
  $CA = \{n_1, \ldots, n_k\}$ is the set of all the **common ancestors** of $n$ and $n'$,
  $n_i \in CA$ is the **nearest common ancestor** of $n$ and $n'$ if and only if
  $\forall n_j \in CA$, $n_i \neq n_j$,
  MIN($\{Dist(n_i, n), Dist(n_i, n')\}$) <
  MIN($\{Dist(n_j, n), Dist(n_j, n')\}$)
The statement, $n_i$ is the **nearest common ancestor** of $n$ and $n'$, is briefly denoted as $NCA(n, n') = n_i$

To simplify our analysis, the workflow specifications discussed contain no cycles; i.e. they are directed acyclic graphs (DAG). DAG is widely adopted for the analysis of control flows or data flows in a workflow schema (Li et al., 2004a; Sadiq and Orlowska, 2000; Marjanovic, 2000; Reichert and Dadam, 1998; Sadiq et al., 2003). Besides, the control blocks in our workflow specification are **totally contained** or **exclusive** to each other. All the workflow specifications discussed in this paper are well-formed DAG. Axiom 1 gives a formal definition to the constraints which keeps a workflow well-formed and therefore the axiom is hold through all statements in this paper.

**Axiom 1** (Well-formed workflow specification). A workflow specification $ws = (N, F, R, C, S, E)$ is a **well-formed** DAG if and only if all the following rules hold:

(1) $Reachable(S, E) =$ true
(2) $S$ has no in-flows and $E$ has no out-flows
(3) $\forall$ process $n \in N$
  $n$ has exactly one in-flow if and only if $n.type \neq \{\text{AJ}, \text{OJ}\}$
  $n$ has exactly one out-flow if and only if $n.type \neq \{\text{AS}, \text{OS}\}$
(4) $\forall$ process $n \in N$-$\{S, E\}$
  $Reachable(S, n) =$ true and $Reachable(n, E) =$ true
(5) $\forall$ processes $n_i$, $n_j \in N$
  if $Reachable(n_i, n_j) =$ true, $Reachable(n_j, n_i) =$ false
(6) $\forall$ process $n \in N$, $n.type = \{\text{AS}, \text{OS}\}$ if and only if there exists one and only one control block $c \in C$ and $c.start = n$
(7) $\forall$ process $n \in N$, $n.type = \{\text{AJ}, \text{OJ}\}$ if and only if there exists one and only one control block $c \in C$ and $c.end = n$
(8) $\forall$ control blocks $c_i$, $c_j \in C$
  $Reachable(c_i.start, c_j.start) =$ true if and only if $Reachable(c_j.end, c_i.end) =$ true or $Reachable(c_i.end, c_j.start) =$ true

Points (1)–(5) give basic constraints of a well-formed workflow; among them, points (4), (5) keep the workflow acyclic. Points (6)–(8) limit the construction of control blocks and control processes. Point (8) keeps all control blocks in a well-formed workflow totally contained or exclusive to each other.

## 3. Resource conflicts in a WfMS

In this section, resource conflicts discussed in this paper are described.

In a workflow, one resource might be referenced by two or more processes. When a resource is referenced by a process, it indicates that the process might depend on the resource. Such a phenomenon is formally defined as **resource dependency** in Definition 14.

**Definition 14** (Resource dependency).

$\forall$ process $n \in N$, resource $r \in R$
  $n$ and $n'$ have **resource dependency** on $r$ if and only if $n \in r.use$ and $r \in n.ref$

In a workflow, the processes are possibly active spontaneously when they are in different paths split from an and-split control process and have overlapped estimated active interval. Such a condition is called **potentially concurrent execution** in this paper, and it is formally defined in Definition 15.

**Definition 15** (Potentially concurrent execution).

$\forall$ process $n$, $n' \in N$
$n$ and $n'$ have **potentially concurrent execution** if and only if the following conditions hold:

(1) $Reachable(n,n') = \text{false}$ and $Reachable(n',n) = \text{false}$
(2) $NCA(n,n').type = \text{AS}$
(3) $EAI(n) \otimes EAI(n') > 0$

A resource conflict occurs when a pair of processes might concurrently access the same resource. Therefore, a resource conflict can be depicted formally in Definition 16.

**Definition 16** (Resource conflict).

$\forall$ process $n$, $n' \in N$
$n$ and $n'$ have **resource conflict** if and only if the following conditions all hold:

(1) $n$ and $n'$ have **resource dependency** on a resource
(2) $n$ and $n'$ have **potentially concurrent execution**

The resource conflict above can be denoted as $(r,n,n')$

For algorithm usage, all resource conflicts in a workflow specification are collected in set $RCT$ defined below.

**Definition 17** (*Resource conflict table*).

$(r,n,n')$ $RCT$ if and only if $n$ and $n'$ have **resource conflict** on resource $r$.

From description of Definition 17, $(r,n,n')$ and $(r,n',n)$ are equivalent. i.e. $(r,n,n')$ belongs to $RCT$ if and only if $(r,n',n)$ belongs to $RCT$. Any addition or removal of resource conflicts in $RCT$ should be warned to the workflow designer.

## 4. Edit operations for workflow specification

Our approach records the information about resources and processes affected by each edit operation. The information about resource conflicts is recorded in table $RCT$, and reported to designers immediately. All the edit operations are asked to follow constraints from axiom 1 for the integrity of a well-formed workflow. It is a more popular facility to provide the information, created or to be applied, after user enters an edit operation in current editors of programming languages. Similarly, our methodology calculates the influences after each edit operation to improve the interaction between designers and the environment. With the methodology, the designers can obtain information after each move they made, and respond to any conflicts immediately.

Generally speaking, insertion, modification and deletion of the resources and processes are the operations made by the designers. To simplify our discussion, the following limitations to the edit operations are made. First, an activity process can be directly inserted into/removed from a workflow. A pair of control processes is added/removed through inserting/removing a control block into/from the workflow. In modifying properties of a process, only adding/removing resource references and altering minimal or maximal working durations are discussed, and only properties of activity processes can be modified. Before the edit operations are introduced in detail, the basic model for workflow specifications, processes, and control blocks are defined in Definition 18. Editing must be operated on a newly initialized basic workflow specification or on an existent one.

**Definition 18** (Basic models).

A basic workflow specification $ws = (\{S,E\},\{(S,E)\},\{\},\{\},S,E)$ in which
$D(S) = d(S) = D\,(E) = d\,(E) = 0$
A basic process $n$: $n.type = \{\text{ACT},\text{AS},\text{OS},\text{AJ},\text{OJ}\}$,$n.ref = \phi$, $d(n) = D(n) = 0$
A basic control block $c$: $c.start = n_s$, $c.end = n_e$, $(n_s,n_e) \in F$, $c.p\_set = \phi$

In this paper, the following edit operations are discussed:

(1) Inserting a basic activity process $n$ into an existent flow $(n_i,n_j)$:
Type of $n$ is initialized as ACT. $(n_i,n_j)$ is removed from $F$; and $(n_i,n)$, $(n,n_j)$ are added into $F$. If $n_i$ is not a split process, set $n.cb$ to $n_i.cb$, or set $n.cb$ to the control block which $n_i$ starts. If $n.cb$ is not "ROOT", add $n$ to $n.cb.p\_set$.
(2) Inserting a basic control block $c$ into an existent flow $(n_i,n_j)$:
$(\underline{n_i},n_j)$ is removed from $F$ and $(n_i,c.start)$ and $(n,c.end)$ are added into $F$. If $n_i$ is not a split process, set both $c.start.cb$ and $c.end.cb$ to $n_i.cb$, or set them to the control block which $n_i$ starts. Let $c'$ be $c.start.\ cb$. Add $c.start$ and $c.end$ to $c'.p\_set$ if $c'$ is not "ROOT".
(3) Adding a split path to a control block:
For a control block $c$, if $(c.start,c.end) \notin F$, add $(c.start,c.end)$ to $F$.
(4) Removing a split path from a control block:
For a control block $c$, if $(c.start,c.end) \in F$ and $c.p\_set \neq \phi$, remove $(c.start,c.end)$ from $F$.
(5) Adding/removing a resource $r$ to a workflow:
Add/Remove some resource $r$ into/from $R$, respectively. The resource $r$ may be a document, some human resource, or external data, etc.
(6) Adding a resource reference $r$ to an activity process $n$:
Add $r$ to $n.ref$ and add $n$ to $r.use$.
(7) Removing a resource reference $r$ from an activity process $n$:
Remove $r$ from $n.ref$ and remove $n$ from $r.use$.
(8) Setting minimal or maximal working durations of an activity process $n$:
Set $d(n)/D(n)$ to wish value.
(9) Removing an existing activity process $n$:
For any resource reference $r$in $n.ref$, remove $n$ from $r.use$, remove inflow $(n_i,n)$ and out-flow $(n,n_j)$ from

$F$, and add $(n_i, n_j)$ to $F$. Then, if $n.cb$ is not "ROOT", remove $n$ from $n.cb.p\_set$. Finally, remove $n$ from $N$.

(10) Removing a control block $c$ from the workflow: Remove all the control blocks and activity processes in $c$, remove $(n_i, c.start)$, $(c.end, n_j)$, and $(c.start, c.end)$ from $F$, and add $(n_i, n_j)$ to $F$. Then, remove $c.start$ and $c.end$ from $N$

Among the operations above, operations (1), (2), (6)–(10) may generate or eliminate resource conflicts. Operation (5) may eliminate resource conflicts. The designers modify working duration with operation (8) and therefore, operation (8) is also called duration modification operation in this paper. Operation (6), (7), and (8) may cause a series of changes in EAIs and therefore is called temporal related operations. Combination between the operations and our approaches is discussed in Section 5.5.

## 5. An incremental algorithm detecting resource conflicts in workflow specifications

In this section, several algorithms for detecting different properties of a resource conflict are introduced in Sections 5.1–5.4. For each algorithm, the correctness of output conditions is proved and the time complexity is discussed. At Section 5.5, the relations between our algorithm and edit operations are discussed.

### 5.1. Algorithm for detecting resource dependency

With the target process $n$ which has resource dependency on resource $r$, Algorithm 1 simply collects all the other processes which have resource dependency on $r$. The algorithm is primarily used after a new resource reference is added to a process.

**Algorithm 1** (*Check resource dependency*).

```
Input: $r \in R$, $n \in N$, $n.type = \mathrm{ACT}$, $r \in n.ref$
Output: A process set named $CRD$which contains all the
processes that have resource dependency on $r$. $n \notin CRD$
Algorithm:
    Process set $CRD$ {
    1. $CRD = r.use - \{n\}$
}
```

By Definitions 2 and 14, the proof to the correctness of the algorithm is intuitive and therefore omitted.

The time complexity of Algorithm 1 is constant, since the data required are recorded during editing.

### 5.2. Algorithm for detecting potentially concurrent execution

In this section, the algorithm to detect potentially concurrent execution is constructed. Before the algorithm is introduced, some lemmas and an additional definition helping the discussion of the algorithm are presented.

To detect whether two mutually unreachable processes have potentially concurrent execution, the type of their nearest common ancestor is checked, and their active intervals are compared.

First, the nearest common ancestor of any two mutually unreachable activity processes is a split process. This feature is formally described in Lemma 1 and proved as follows.

**Lemma 1**

$\forall processes\ n, n' \in N,\ n.type = n'.type = ACT$
*When* $Reachable(n, n') = false$ *and* $Reachable(n', n) = false$, $NCA(n, n').type = \{AS, OS\}$

**Proof.** Assume that $n' = NCA(n, n')$ and $n''.type \notin \{\mathrm{AS}, \mathrm{OS}\}$. With Axiom 1, it is known that $n''$ has only one out-flow denoted as $(n'', n_a)$. Since $(n'', n_a)$ is the only out-flow of $n''$, the path between $n''$ and $n$ is $(n'', n_a, \ldots, n)$. On the other hand, the path between $n''$ and $n'$ is also $(n'', n_a, \ldots, n')$. Therefore, $n_a$ has smaller distance to both $n$ and $n'$ than $n''$ does. $n''$ can not be $NCA(n, n')$. It is a contradiction. □

With Lemma 1, the nearest common ancestor can be easily obtained from the split typed ancestors. Algorithm 2 is designed to obtain the list of all split typed ancestors between an activity process and the root path. The nearest common ancestor of the target processes can be located by comparing the sets obtained by the algorithm.

Since our purpose is to find the nearest common ancestor, the output set of Algorithm 2 is produced as path ordered, i.e. the processes in the output set is ordered by distance from the target process. To clearly elaborate this concept, a **path ordered process set** is defined as following.

**Definition 19** (*Path ordered process set*).

A process set $S$ is said to be **path ordered** if the following conditions hold:

(1) if $|S| = 0$, $S = \phi$
(2) if $|S| = 1$, $S = \{n | n \in N\}$
(3) if $|S| > 1$, $S = \{n_1, \ldots, n_k | n_i \in N\}$, $\forall n_j \in S$, $1 < j \leqslant k$, $Reachable(n_j, n_{j-1}) = \mathrm{true}$

In our algorithm, it is assumed that the order of elements in any subset of a path ordered set does not change, and therefore the subset(s) are still kept path ordered. With this assumption, intersection of two **path ordered** sets is also **path ordered**.

Algorithm 2 tracks back from the target input activity process $n$, and finds the split typed ancestors which start the control block containing $n$. In other words, if $n'$ collected by Algorithm 2 is the start process of a control block $c$, $Reachable(c.start, n)$ and $Reachable(n, c.end)$ are both true. The output set of Algorithm 2 is path ordered as defined in Definition 19.

**Algorithm 2** (*Find split ancestor(s) belonging to*).

```
   Input: $n \in N$, $n.type = \text{ACT}$
   Output: a **path ordered** process set named $FANB$
      (1) $\forall$ process $n_i \in FANB$ $n_i$ is an **ancestor** of $n$
      (2) $\forall$ process $n_i \in FANB$ $n_i.type = \{\text{AS}, \text{OS}\}$
   Algorithm:
      Path ordered process set $FANB$ {
1. $c = n.cb$
2. loop {
3. if($c == \text{ROOT}$ ) break the loop
4. $tp = c.start$
5. add $tp$ into $FANB$
6. $c = tp.cb$
7. }

   }
```

**Proof.** Let us assume that the result set $FANB = \{p_1, p_2, \ldots, p_k\}$. With the loop structure of the algorithm, $p_{i+1}$ is the start process of the control block containing $p_i$, i.e. $p_{i+1}$ is $p_i.cb.start$. Since the loop halts when the process is on the root path, with Definition 3, *Reachable*$(p_{i+1}, p_i)$ = true holds. $FANB$ is path ordered. $p_1$ is $n.cb.start$ if $n.cb$ is not ROOT. *Reachable*$(p_1, n)$ = true. Therefore for any process $p_i$ in $FANB$, *Reachable*$(p_i, n)$ = true, $p_i$ is an ancestor of $n$. The output condition (1) holds.

All the processes collected into $FANB$ are start process of a control block. By Definition 3 again, the start process of any control blocks is either AS or OS. The output condition (2) also holds. □

The time complexity of Algorithm 2 is decided by the loop part. The number of loops required is decided by the number of nested control blocks containing the input process, and the start processes of each of these control blocks are collected by the algorithm. To simplify our discussion, without loss of generality (W.L.O.G), it is assumed that each split process is followed with at least two split paths. With the definition about well-formed DAG in Axiom 1, control blocks are totally contained or exclusive to each other like what Fig. 3 shows. In real cases, the number of split paths following each split processes may vary; however, it does not affect the upper-bound of the time complexity of the algorithm. Due to the path growth of splitting, the time complexity of Algorithm 2 is O(log|$N$|). In Fig. 3, the total number of processes is 10, and the number of split processes from root path to any activity process is 2. This example indicates the execution of Algorithm 2 needs 2 loops only, much less than $\log_2 10$, i.e. $\log_2|N|$, loops.

Algorithm 2 is the core part to detect if two activity processes are in different paths split from some process. With the algorithm, Algorithm 3 is constructed to detect potentially concurrent execution among input processes as following:

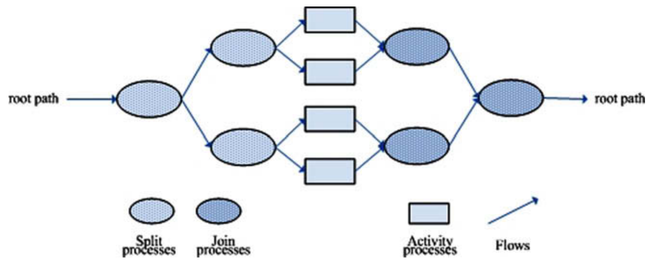


Fig. 3. Sample structure of control blocks.

**Algorithm 3** (*Check potentially concurrent execution*).

```
   Input: $n \in N$, $n.type = \text{ACT}$, $S = \{n_i | n_i \in N, n_i.type = \text{ACT}\}$
   Output: a process set named $CPCE$
      (1) $CPCE \subseteq S$
      (2) $\forall$ process $n_i \in CPCE$, $n_i$ and $n$ have **potentially concurrent execution**
   Algorithm:
   Process set $CPCE$ {
1. $\forall$ process $n_i \in S$
2. if($Reachable(n, n_i)$ == true or $Reachable(n_i, n)$ == true)
   remove $n_i$ from $S$
3. $\forall$ process $n_i \in S$ {
4. Let $U = FANB(n) \cap FANB(n_i) = \{n_{a1}, \ldots, n_{ak}\}$
5.    if($U \neq \phi$ and $n_{a1}.type$ == AS and $EAI(n) \otimes EAI(n_i) > 0$)
6.       add $n_i$ to $CPCE$
7. }
   }
```

To prove the correctness of output conditions of Algorithm 3, the following lemma is shown.

**Lemma 2.**

*$\forall$processes $n$, $n' \in N$, $n.type = ACT$ and $n'.type = ACT$*
*If $Reachable(n, n')$ and $Reachable(n', n)$ are both false, $NCA(n, n')$ is the first element in $FANB(n) \cap FANB(n')$.*

Lemma 2 indicates that the nearest common ancestor of any two mutually unreachable processes is in the intersection of sets obtained by Algorithm 2 for both processes. To prove Lemmas 2–4 are introduced.

Based on Lemma 1 and Definition 3, the nearest common ancestor of any two mutually unreachable processes is known as a split typed process. Lemma 3 shows that two mutually unreachable processes are contained in the control block started by their nearest common ancestor.

**Lemma 3**

*$\forall$processes $n$, $n' \in N$, $n.type = ACT$ and $n'.type = ACT$*
*When $Reachable(n, n') = false$ and $Reachable(n', n) = false$ and $NCA(n, n')$ is $c.start$ for some control block $c$,*
*$Reachable(n, c.end) = Reachable(n', c.end) = true$*

**Proof.** By way of contradiction (B.W.O.C), if one of $Reachable(n, c.end)$ and $Reachable(n', c.end)$ is false. W.L.O.G, it is assumed that $Reachable(n, c.end) = \text{false}$ and $Reachable(n', c.end) = \text{true}$. If $Reachable(c.end, n) = \text{true}$, $Reachable(n', n) = \text{true}$. It is a contradiction. On the other hand, let's discuss the condition about $Reachable(c.end, n) = \text{false}$. With Definition 3, it is known that $Reachable(c.start, c.end) = \text{true}$. Since $Reachable(c.end, n)$, $Reachable(n, c.end)$ and $Reachable(n, c.end)$ are all false, $c.start$ can not be an ancestor of $n$. It is a contradiction. □

Lemma 4 shows that for any activity process $n$, Algorithm 2 collects the start process(es) of the control block(s) containing $n$. In other words, there exist paths from $n$ to the corresponding join processes of these split processes.

**Lemma 4**

*$\forall n \in N$, $n.type = ACT$ and $c \in C$, $Reachable(c.start, n) = true$*
*$c.start \in FANB(n)$ if and only if $Reachable(n, c.end) = true$*

**Proof**

(1) If $Reachable(n, c.end) = \text{true}$, $c.start \in FANB(n)$B.W.O.C, it is assumed that $n' \notin FANB(n)$. Since $Reachable(n, c.end)$ and $Reachable(c.start, n)$ are both true, $n$ is contained in $c$. With the Algorithm 2 and the edit operations defined, it is known that $c.start$ is contained in $FANB(n)$. It is a contradiction.
(2) If $c.start \in FANB(n)$, $Reachable(n, c.end) = \text{true}$.

B.W.O.C, it is assumed that $Reachable(n, c.end) = \text{false}$. Since $Reachable(c.start, c.end)$ and $Reachable(c.start, n)$ are both true, $Reachable(c.end, n)$ is true. With the Algorithm 2 and the edit operations defined, it is known that $c.start$ can not be in $FANB(n)$ if $c.start, c.end$ and $n$ are on the same path in such a sequence. It is a contradiction.

With all proofs above, Lemma 4 holds. □

With Lemmas 3 and 4, Lemma 2 is proved as follows.

**Proof of Lemma 2.** By Definition 13, $NCA(n, n')$ is an ancestor of both $n$ and $n'$. With Lemmas 1 and 3, $NCA(n, n')$ is known as a split process and is the start process of some control block $c$. Processes $n$ and $n'$ are quoted by $c.start$ and $c.end$. With Lemma 4 and output conditions of Algorithm 2, $NCA(n, n') \in FANB(n) \cap FANB(n')$. By definition, both $FANB(n)$ and $FANB(n')$ are path ordered so that $FANB(n) \cap FANB(n')$ is **path ordered**. Thus, the first element in $FANB(n) \cap FANB(n')$ is the nearest one to $n$ and $n'$ among all the processes in the set. In other words, $NCA(n, n')$ is the first element in $FANB(n) \cap FANB(n')$. Lemma 2 holds. □

With Lemma 2, the correctness of the output conditions of Algorithm 3 is now proved.

**Proof of the output conditions of Algorithm 3.** Output condition (1) holds simply because of line 1, 3, and 6 of the algorithm.

$\forall$ process $n_i \in CPCE$, it is known that $Reachable(n, n_i)$ and $Reachable(n_i, n)$ are both false due to line 2 of the algorithm. With Lemma 2 it is known that $NCA(n, n_i)$ is equal to $n_{a1}$ which is the first element of $FANB(n) \cap FANB(n_i)$. From line 5 of the algorithm, $NCA(n, n_i).type = \text{AS}$ and $EAI(n) \otimes EAI(n_i) > 0$ holds. Therefore, $n$ and $n_i$ are potentially concurrently executed and the output condition (2) holds. □

The time complexity of Algorithm 3 is decided by function $Reachable(\,)$. With Axiom 1, for $p, q \in P$, whether $Reachable(p, q) = \text{true}$ can be decided by a simple process tracking methodology started from $p$. The methodology traces the process one by one in BFS (Breadth First Search) by flows in $F$ until the end process or the target process $q$ is reached. This methodology provides time complexity $O(|N|)$ to the function. There might exist some more talented idea which make $Reachable(\,)$ faster; however to discuss how to improve the efficiency of $Reachable(\,)$ is out of the scope of this paper. With a normal and reasonable function $Reachable(\,)$ and the time complexity discussed about Algorithm 2, the time complexity of Algorithm 3 is known as $O(|N|^2)$.

### 5.3. Algorithm to calculate estimated active intervals

In Definitions 5 and 6, the formula to calculate EAIs of processes is defined. EAI change ripples to descendent processes until a join processes or the end process is met. Algorithm 4 works after any working duration is modified or an activity process/control block is removed. All the affected processes are touched by the algorithm and their estimated active intervals are correctly updated. In the algorithm, $EST'(n)$ and $LET'(n)$ are records to original $EAI(n)$ before the algorithm is invoked. In the following paragraphs, $EAI'(n)$ represents $[EST'(n), LET'(n)]$. $EAI'(n)$ is used to decide if the $EAI(n)$ is updated in the calculation.

**Algorithm 4** (*Calculate EAI*).

```
Input: $n \in N$, $n.type = ACT$
Output: a process set $CEAI$ containing all processes
which $EAI$ is changed in this calculation
Algorithm:
Process set $CEAI$ {
 1. initialize a process queue $Q$
 2. $Q.enqueue(n)$
 3. $\forall\ n_i \in N$, $(n, n_i) \in F$
 4.     $Q.enqueue(n_i)$
 5. while($Q$ is not empty) {
 6.     $p = Q.dequeue$
 7.     $EST'(p) = EST(p)$, $LET'(p) = LET(p)$
 8.     if($p.type == \text{AJ}$) {
 9.         $EST(p) = \text{MAX}(\{EST(n_i) + d(n_i) | (n_i, p) \in F\})$
10.         $LET(n) = \text{MAX}(\{LET(n_i) | (n_i, p) \in F\})$
11.     }
12.   else if($p.type$ == OJ) {
13.      $EST(p) = \mathrm{MIN}(\{EST(n_i) + d(n_i) | (n_i, p) \in F\})$
14.      $LET(p) = \mathrm{MAX}(\{LET(n_i) | (n_i, p) \in F\})$
15.   else {
16.      there exists $(n_i, p) \in F$ such that
17.      $EST(p) = EST(n_i) + d(n_i)$;
18.      $LET(p) = LET(n_i) + D(p)$;
19.   }
20.   if($EST'(p) \notin EST(p)$ or $LET'(p) \notin LET(p)$) {
21.      add $p$ to $CEAI$
22.      $n_i \in N$, $(p, n_i) \in F$
23.         if($n_i \notin Q$) $Q.enqueue(n_i)$
24.   }
25. }
   }
```

**Proof.** To prove the correctness of the algorithm, first, it is shown that after a duration modification operation, for any process, its EAI is changed if and only if it is in *CEAI*. Any EAI change is accomplished from line 8 to line 19 in the algorithm only. Therefore when any EAI change occurs, the process with altered EAI is found in line 20 and is put into *CEAI* at line 21. On the other hand, with the algorithm, it is known that the process with no EAI change is not put into *CEAI*. □

Second, it is shown that for processes whose EAI should be altered, they are in *CEAI*. By Definitions 4–6, for a process, its EAI is changed only when (1) its $D$ value is modified, (2) the $d$ value of its precedent process is modified, or (3) EAI of its precedent process is changed. When $d(n)$ or $D(n)$ is modified, $n$ and its descendent process are enqueued into $Q$ at line 2 and line 4. EAI change originated from condition (1) or (2) is calculated from line 8 to line 19 and the process with altered EAI is put into *CEAI* at line 21. For any $(q', q) \in F$ If $EAI(q')$ is changed, $q$ is enqueued into $Q$ at line 23, and therefore $EAI(q)$ is calculated from line 8 to line 19 and is put into *CEAI* at line 21 if $EAI(q)$ is changed. The process with EAI change originated from condition (3) is also included in *CEAI*.

Last, according to line 20 to 24, a process is put into *CEAI* if its EAI has been changed. Since the EAI is calculated according to the formula defined in Definitions 5 and 6, any process that should have no EAI change is not put in *CEAI*.

With above statements, the correctness of Algorithm 4 holds. □

The time complexity of Algorithm 4 is simply $O(|N|)$ since the operation affect only reachable processes of $n$ which are at most all processes in the workflow.

### *5.4. Algorithm for checking resource conflicts after temporal related operations*

After a temporal related operation, the *RCT* set may need update. The following lemma describes the influences created by temporal related modification on a workflow specification. Based on the lemma, Algorithm 5 which updates *RCT* after temporal related operations is constructed.

**Lemma 5**

*∀processes $n \in N$*
*The shrink of $EAI(n)$ can not create new resource conflicts, and the expansion of $EAI(n)$ cannot eliminate any resource conflict either*

**Proof.** The shrinks or expansions of $EAI(n)$ do not affect the structure or the resource references of the workflow. Therefore, the creation or elimination of resource conflicts by temporal related operations is only resulted from the change(s) of EAI(s). □

The discussions are made as following.

(1) The shrink of $EAI(n)$ can not create new resource conflicts.
B.W.O.C, it is assumed that $(r, n, n')$ is a new resource conflict created due to $EAI(n) \otimes EAI(n') > 0$ after the shrink. This case indicates first, $EAI'(n) \otimes EAI(n') < 0$, i.e. $\mathrm{MIN}(\{LET'(n), LET(n')\}) - \mathrm{MAX}(\{EST'(n), EST(n')\}) < 0$, and second, $LET'(n) > LET(n)$ or $EST'(n) < EST(n)$. The above statement induces that $\mathrm{MIN}(\{LET(n), LET(n')\}) - \mathrm{MAX}(\{EST(n), EST(n')\}) < 0$, i.e. $EAI(n) \otimes EAI(n') < 0$. It is a contradiction.

(2) The expansion of $EAI(n)$ can not eliminate any resource conflict.
B.W.O.C, it is assumed that $(r, n, n')$ is a resource conflict eliminated due to $EAI(n) \otimes EAI(n') < 0$ after the expansion. This case indicates first, $EAI'(n) \otimes EAI(n') > 0$, i.e. $\mathrm{MIN}(\{LET'(n), LET(n')\}) - \mathrm{MAX}(\{EST'(n), EST(n')\}) > 0$, and second, $LET'(n) < LET(n)$ or $EST'(n) > EST(n)$. The above statement induces that $\mathrm{MIN}(\{LET(n), LET(n')\}) - \mathrm{MAX}(\{EST(n), EST(n')\}) > 0$, i.e. $EAI(n) \otimes EAI(n') > 0$. It is a contradiction.

By (1), (2), the lemma is proved. □

Algorithm 5 checks the affected processes calculated by Algorithm 4 according to the properties in Lemma 5.

**Algorithm 5** (*Check resource conflict after duration modification*).

```
Input: process set $S = \{n | n \in N, n.type = \mathrm{ACT}, EAI'(n) \neq EAI(n)\}$
Output:
Algorithm:
   CRCDM {
1. ∀ $n \in S$ {
2. if($EST(n) > EST'(n)$ or $LET(n) < LET'(n)$) {
3.    $\forall r \in n.ref$
4.       $\forall\ (r, n, n') \in RCT$
5.         if($EAI(n) \otimes EAI(n') < 0$)  remove  $(r, n, n')$  from
   $RCT$
6. }
7. if($EST(n) < EST'(n)$ or $LET(n) > LET'(n)$) {
8.    $\forall\ r \in n.ref$
9.       $\forall\ n' \in CPCE(n, CRD(r, n))$
10.          add $(r, n, n')$ to $RCT$
11. }
12. }
   }
```

In Algorithm 5, the activities from line 2 to line 6 check the existent resource conflicts and the remove the resource conflicts which does not exist further. The activities from line 7 to line 12 check if there is any new resource conflict created because of the edit operation. The resources referenced by the target process are extracted at line 8. At line 9, with Algorithms 1 and 3, the processes which have resource conflicts to the input processes are picked.

The time complexity of the first part (line 2 to line 6) is $O(|R|^*|N|)$ because there are at most $|R|$ resources referenced by a process and $|N| - 1$ other processes might have resource conflict with $n$. The time complexity for the second part (line 7 to line 11) of the algorithm is $O(|R|^*|N|^2)$ since the time complexity for function $CPCE()$ (Algorithm 3) is $O(|N|^2)$. With the loop quotes from line 1 to line 12, the time complexity for Algorithm 5 is $O(|R|^*|N|^3)$.

### 5.5. Combining the algorithms with the operations

In Section 4, 10 edit operations for workflow specifications are introduced, and seven of them may result in resource conflicts. The algorithms described in Section 5 can be used to detect resource conflicts caused by the seven operations. In following paragraphs, the combinations between the algorithms and each operation are described.

(1) Inserting a basic activity process $n$ into an existent flow $f$.
$CEAI(n)$ is called to calculate $EAI(n)$. According to Definition 18, $D(n)$ and $d(n)$ are both initialized as zero, and $n.ref$ is empty. Therefore, no resource conflicts are created or eliminated, and it is not necessary to call $CRCDM()$ in this operation.

(2) Inserting a basic control block $c$ into an existent flow $f$.
Similar to (1), $CEAI(c.start)$ is called.

(3) Removing a resource $r$ to a workflow:
For any process $n \in r.use$, remove $r$ from $n.ref$. For any processes $n_i, n_j \in N$, $(r, n_i, n_j) \in RCT$, remove $(r, n_i, n_j)$ from $RCT$.

(4) Adding a resource reference $r$ to an activity process $n$.
$\forall n' \in CPCE(n, CRD(r, n))$. $(r, n, n')$ is put in $RCT$.

(5) Removing a resource reference $r$ from an activity process $n$.
$RCT$ is checked to remove resource conflicts related to both $r$ and $n$. In other words, for any resource conflict $(r, n, n_i) \in RCT$ where $n_i \in N$, $(r, n, n_i)$ is removed from $RCT$.

(6) Setting minimal or maximal working duration of an activity process $nCRCDM(CEAI(n))$ is called.

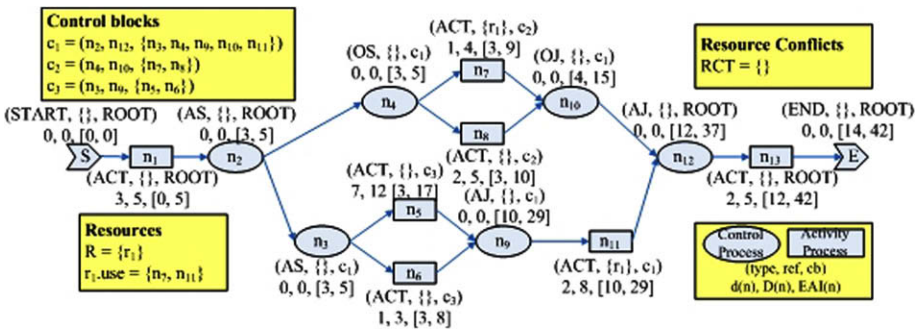
(7) Removing an existing activity process $n$.
Resource conflicts related to $n$ are removed from $RCT$. Before $n$ is removed from the specification, $d(n)$ and $D(n)$ is set to 0, and For $(n_i, n)$ and $(n, n_j) \in F, (n_i, n_j)$ is added to $F$ after the removal. $n_j$'s precedent process is changed from $n$ *to* $n_i$, and therefore $CRCDM(CEAI(n_j))$ is called to calculate the effect of EAI change.

(8) Removing a control block $c$ from the workflow.
The resource conflicts related to the activity processes in $c$ are removed from $RCT$. For $(n_i, c.start)$ and $(c.end, n_j) \in F, (n_i, n_j)$ is added to $F$ after the removal. $n_j$'s precedent process is changed from $c.end$ to $n_i$, and therefore $CRCDM(CEAI(n_j))$ is called to calculate the effect of EAI change.

## 6. Examples

To demonstrate our algorithms, three examples are constructed in this section. First, how to detect the resource conflict generated by resource assignment is shown; second, the affect by change of working durations is represented; and the last example shows the influence when an activity process is removed.

The first example is initialized with the workflow specification in Fig. 4 and the corresponding algorithms are applied step by step in the following sections.


Control blocks
c₁ = (n₂, n₁₂, {n₃, n₄, n₉, n₁₀, n₁₁})
c₂ = (n₄, n₁₀, {n₇, n₈})
c₃ = (n₃, n₉, {n₅, n₆})
Resources
R = {r₁}
r₁.use = {n₇, n₁₁}
Resource Conflicts
RCT = {}
(START, {}, ROOT) 0, 0, [0, 0]
S
n₁
(ACT, {}, ROOT) 3, 5, [0, 5]
(AS, {}, ROOT) 0, 0, [3, 5]
n₂
(OS, {}, c₁) 0, 0, [3, 5]
n₄
(ACT, {r₁}, c₂) 1, 4, [3, 9]
n₇
n₈
(ACT, {}, c₂) 2, 5, [3, 10]
(OJ, {}, c₁) 0, 0, [4, 15]
n₁₀
n₃
(AS, {}, c₁) 0, 0, [3, 5]
(ACT, {}, c₃) 7, 12 [3, 17]
n₅
n₆
(ACT, {}, c₃) 1, 3, [3, 8]
(AJ, {}, c₁) 0, 0, [10, 29]
n₉
n₁₁
(ACT, {r₁}, c₁) 2, 8, [10, 29]
(AJ, {}, ROOT) 0, 0, [12, 37]
n₁₂
n₁₃
(ACT, {}, ROOT) 2, 5, [12, 42]
(END, {}, ROOT) 0, 0, [14, 42]
E
Control Process
Activity Process
(type, ref, cb)
d(n), D(n), EAI(n)
Fig. 4. The sample workflow specification.

### 6.1. Example 1: adding a resource reference

With the sample example in Fig. 4, the designer adds resource reference $r_1$ to process $n_5$. With discussions in Section 3, $n_5.ref = \{r_1\}$ and $r_1.use = \{n_5, n_7, n_{11}\}$.$CPCE(n_5, CRD(r_1, n_5))$ is called.

$CRD(r_1, n_5)$ simply returns $\{n_7, n_{11}\}$. Now let's trace how $CPCE(n_5, \{n_7, n_{11}\})$ works. Since $Reachable(n_5, n_{11}) =$ true,$n_{11}$ is neglected. By following the definition of path ordered set, $FANB(n_5) = \{n_3, n_2\}$ because $Dist(n_3, n_5) = 1$ and $Dist(n_2, n_5) = 2$. At the same time, $FANB(n_7) = \{n_4, n_2\}$, and $FANB(n_5) \cap FANB(n_7) = \{n_2\}$. Since $n_2$ is the first element of the intersection and $n_2.type =$ AS, $n_5$ and $n_7$ have a nearest common ancestor which is typed and-split.
$EAI(n_5) \otimes EAI(n_7) = \mathrm{MIN}(\{LET(n_5), LET(n_7)\}) - \mathrm{MAX}(\{EST(n_5), EST(n_7)\}) = \mathrm{MIN}(\{17, 9\}) - \mathrm{MAX}(\{3, 3\}) = 9 - 3 = 6 > 0$. Therefore, $CPCE(n_5, \{n_7, n_{11}\}) = \{n_7\}$. Since $(r_1, n_5, n_7) \notin RCT$, it is a new generated resource conflict. The designer is warned when it is put into $RCT$.

After the operation, the workflow specification is updated as in Fig. 5. The changed parts are marked with different colors and under-scopes.

### 6.2. Example 2: modifying working durations of a process

Now, the influences of the modification on working durations are shown. Let the designer change $d(n_5)$ in Fig. 5 from 7 to 4. $CRCDM(CEAI(n_5))$ is called. With Algorithm 4 it is known that $CEAI(n_5)$ returns $\{n_5, n_9, n_{11}, n_{12}, n_{13}, E\}$.

$CRCDM(\{n_5, n_9, n_{11}, n_{12}, n_{13}, E\})$ checks the processes in the target set to validate if there is any change about resource conflicts. To simplify the discussion, the example only discusses how the algorithms dealing with $n_5$ and $n_{11}$ while the other processes in the input set have no resource references. $CPCE(n_5, CRD(r_1, n_5))$ returns $\{n_7\}$. However resource conflict $(r_1, n_5, n_7)$ has already be in $RCT$, and therefore it makes no differences. $CPCE(n_{11}, CRD(r_1, n_{11}))$ is also $\{n_7\}$. Since $(r_1, n_7, n_{11}) \notin RCT$, $(r_1, n_7, n_{11})$ is put into $RCT$. After the operation, $RCT = \{(r_1, n_5, n_7), (r_1, n_7, n_{11})\}$ and the designer is warned of the new generated resource conflict. The workflow specification after the operation is as in Fig. 6, and the altering part is also marked in different colors and under-scopes.

### 6.3. Example 3: removing an activity process from the workflow

In this section, the influence of removal of an activity process named $n_{11}$ from the workflow specification represented in Fig. 6 is shown.

$(n_9, n_{11})$ and $(n_{11}, n_{12})$ are removed from $F$, and $(n_9, n_{12})$ is added to $F$. Since $r_1 \in n_{11}.ref$, $n_{11}$ is removed from $r_1.use$. Therefore, $(r_1, n_7, n_{11})$ is removed from $RCT$, and the removal of the resource conflict is immediately informed to the designer. Than, $CRCDM(CEAI(n_{12}))$ is called. $CEAI(n_{12})$ returns $\{n_{12}, n_{13}, E\}$. Since there is no resource

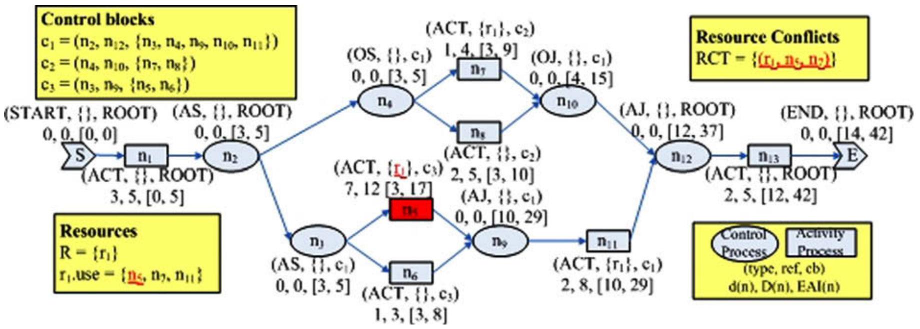

Control blocks
c₁ = (n₂, n₁₂, {n₃, n₄, n₉, n₁₀, n₁₁})
c₂ = (n₄, n₁₀, {n₇, n₈})
c₃ = (n₃, n₉, {n₅, n₆})
Resource Conflicts
RCT = {(r₁, n₅, n₇)}
Resources
R = {r₁}
r₁.use = {n₅, n₇, n₁₁}
(START, {}, ROOT) 0, 0, [0, 0]
(ACT, {}, ROOT) 3, 5, [0, 5]
(AS, {}, ROOT) 0, 0, [3, 5]
(OS, {}, c₁) 0, 0, [3, 5]
(ACT, {r₁}, c₂) 1, 4, [3, 9]
(ACT, {}, c₂) 2, 5, [3, 10]
(OJ, {}, c₁) 0, 0, [4, 15]
(AS, {}, c₁) 0, 0, [3, 5]
(ACT, {r₁}, c₃) 7, 12 [3, 17]
(ACT, {}, c₃) 1, 3, [3, 8]
(AJ, {}, c₁) 0, 0, [10, 29]
(ACT, {r₁}, c₁) 2, 8, [10, 29]
(AJ, {}, ROOT) 0, 0, [12, 37]
(ACT, {}, ROOT) 2, 5, [12, 42]
(END, {}, ROOT) 0, 0, [14, 42]
S
n₁
n₂
n₃
n₄
n₅
n₆
n₇
n₈
n₉
n₁₀
n₁₁
n₁₂
n₁₃
E
Control Process
Activity Process
(type, ref, cb)
d(n), D(n), EAI(n)

Fig. 5. The sample workflow specification after modification on resource references.

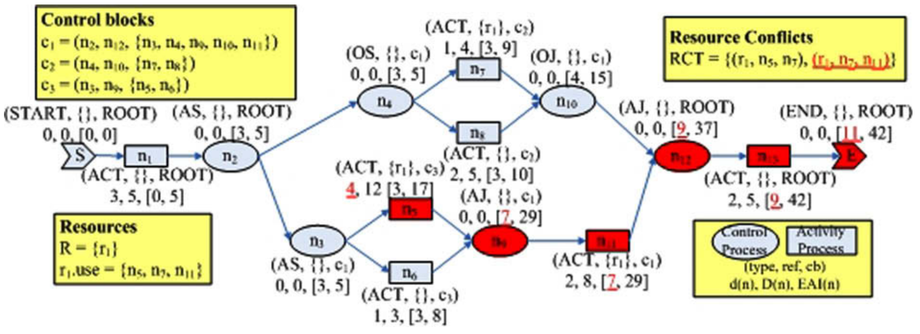

Control blocks
c₁ = (n₂, n₁₂, {n₃, n₄, n₉, n₁₀, n₁₁})
c₂ = (n₄, n₁₀, {n₇, n₈})
c₃ = (n₃, n₉, {n₅, n₆})
Resource Conflicts
RCT = {(r₁, n₅, n₇), (r₁, n₇, n₁₁)}
Resources
R = {r₁}
r₁.use = {n₅, n₇, n₁₁}
(START, {}, ROOT) 0, 0, [0, 0]
(ACT, {}, ROOT) 3, 5, [0, 5]
(AS, {}, ROOT) 0, 0, [3, 5]
(OS, {}, c₁) 0, 0, [3, 5]
(ACT, {r₁}, c₂) 1, 4, [3, 9]
(ACT, {}, c₂) 2, 5, [3, 10]
(OJ, {}, c₁) 0, 0, [4, 15]
(AS, {}, c₁) 0, 0, [3, 5]
(ACT, {r₁}, c₃) 4, 12 [3, 17]
(ACT, {}, c₃) 1, 3, [3, 8]
(AJ, {}, c₁) 0, 0, [7, 29]
(ACT, {r₁}, c₁) 2, 8, [7, 29]
(AJ, {}, ROOT) 0, 0, [9, 37]
(ACT, {}, ROOT) 2, 5, [9, 42]
(END, {}, ROOT) 0, 0, [11, 42]
S
n₁
n₂
n₃
n₄
n₅
n₆
n₇
n₈
n₉
n₁₀
n₁₁
n₁₂
n₁₃
E
Control Process
Activity Process
(type, ref, cb)
d(n), D(n), EAI(n)

Fig. 6. The sample workflow specification after the duration modification.

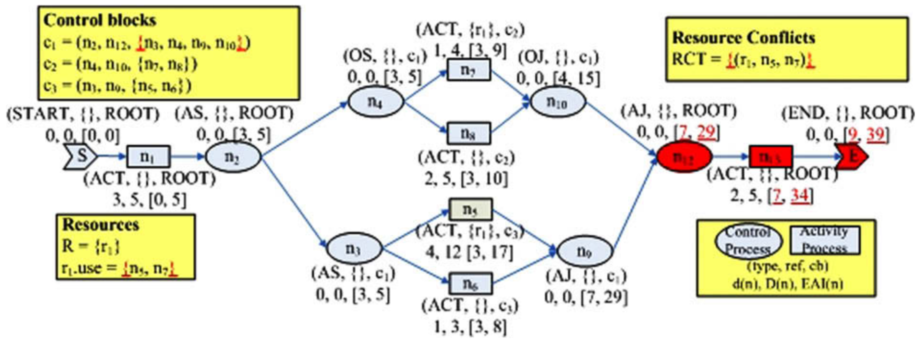

Control blocks
c1 = (n2, n12, {n3, n4, n9, n10})
c2 = (n4, n10, {n7, n8})
c3 = (n3, n9, {n5, n6})
Resources
R = {r1}
r1.use = {n5, n7}
Resource Conflicts
RCT = {(r1, n5, n7)}
(START, {}, ROOT) 0, 0, [0, 0]
(ACT, {}, ROOT) 3, 5, [0, 5]
(AS, {}, ROOT) 0, 0, [3, 5]
(OS, {}, c1) 0, 0, [3, 5]
(ACT, {r1}, c2) 1, 4, [3, 9]
(OJ, {}, c1) 0, 0, [4, 15]
(ACT, {}, c2) 2, 5, [3, 10]
(ACT, {r1}, c3) 4, 12 [3, 17]
(AS, {}, c1) 0, 0, [3, 5]
(ACT, {}, c3) 1, 3, [3, 8]
(AJ, {}, c1) 0, 0, [7, 29]
(AJ, {}, ROOT) 0, 0, [7, 29]
(ACT, {}, ROOT) 2, 5, [7, 34]
(END, {}, ROOT) 0, 0, [9, 39]
S
n1
n2
n3
n4
n5
n6
n7
n8
n9
n10
n12
n13
E
Control Process
Activity Process
(type, ref, cb)
d(n), D(n), EAI(n)

Fig. 7. The sample workflow specification after the removal of an activity process.

referenced by $n_{12}$, $n_{13}$, and $E$, no further change in $RCT$ is made. After the removal, the workflow specification is as in Fig. 7, and the altering part is marked in different colors and under-scopes.

## 7. Related work

The analysis of structural aspects of workflow specification was widely discussed. Bajaj defined a well-formed workflow model for structural analysis (Bajaj and Ram, 2002). Sadiq introduced directed graph (Sadiq and Orlowska, 2000; Sadiq et al., 2003) to analyze process model in structure correctness, such as deadlock or inconsistencies in synchronization. Effective algorithm was built for analysis in Sadiq's research. Aalst transformed workflow model onto Petri-nets (van der Aalst et al., 1999; van der Aalst and ter Hofstede, 2000; van der Aalst, 1998). Petri-nets are suitable for the analysis of parallel behavior in a workflow. The existent tools or methodologies for Petri-nets can be adopted to analyze workflows through Aalst's transformation. Adam constructed a workflow model with temporal consideration based on Petri-net (Adam et al., 1998) for analysis, simulation and validation.

Combining timing constraints into the analysis of workflow models becomes popular after many researches about structural aspects of a workflow are accomplished. Adam et al. (1998) took timing constraints as external conditions of a workflow to analyze structural correctness of a Petri-net based workflow model. Li et al. (2004b) added timing constraints into workflow nets (WF-net, Petri-net based workflow model) to make a timing workflow net (TWF-net) for performance analysis. Workload and some other timing constraints are discussed in Li's research. Eder discussed timing constraints in Eder et al. (1999a,b). Based on a graph-based workflow model, Eder developed a timed graph which shows the working duration with the earliest and the latest finish time. Several time constraints such as fixed-date, lower-bound, and upper-bound constraints are discussed in addition. The algorithms to build the timed graph and to detect violation of timing constraints are constructed in Eder et al. (1999b). Marjanovic verified a production workflow model with temporal constraints in a dynamic way (Marjanovic, 2000). In Marjanovic (2000), the timing model is built based on duration and instantiation space. Absolute and relative deadline constraints are modeled and the dynamic methodologies verifying such constraints are thus built. The time modeling raised in Marjanovic (2000) can be also used for monitoring the execution of workflows, reasoning about the deadlines, and managing the workloads of the tasks.

Resource allocation is another topic for analysis of workflow models. Tang et al. (2004) extended a Petri-net based workflow model for modeling compositions of web-services and resources. Reveliotis (2003), Park and Reveliotis (2001) integrated resource allocation systems into a workflow model in order to analyze deadlock and synchronization problems. Sun et al. (2006) extended Li's research (Li et al., 2004b) with additional resource constraints to analyze performance of workflows. Russel concluded 43 resource allocation patterns in Russell et al. (2004). Coordination among workflow, human resources and external resources are discussed in detail and are well categorized. However in Russel's research, resources are more like the participants of workflows, and is different from the resources discussed in our work in definitions and usages. Various representations and utilization of resources in workflows are captured as patterns (Russell et al., 2004).

Li et al. (2004a) modeled resources and temporal constraints into workflow specification for analysis. Li focused on addition and removal of resource conflicts in a timed workflow specification. In Li et al. (2004a), the notations of features used in resource analysis in a timed workflow are defined. Li's work gave a solid foundation for further work. Therefore, Zhong and Song (2005) applied it on a Petri-net based workflow model and Hsu et al. (2005) applied it on a DAG based model. In Zhong and Song (2005), Zhong applied Li's timing model to form Petri-net based timed workflow conceptually similar to the model described in Hsu et al. (2005). In Hsu et al. (2005), a complete and detailed algorithm for the analysis of resource conflicts in a workflow specification was raised. Hsu focused on providing information to the workflow designers about resource conflicts in a workflow specification during design time. With an incremental algorithm, the information created during designing a workflow is

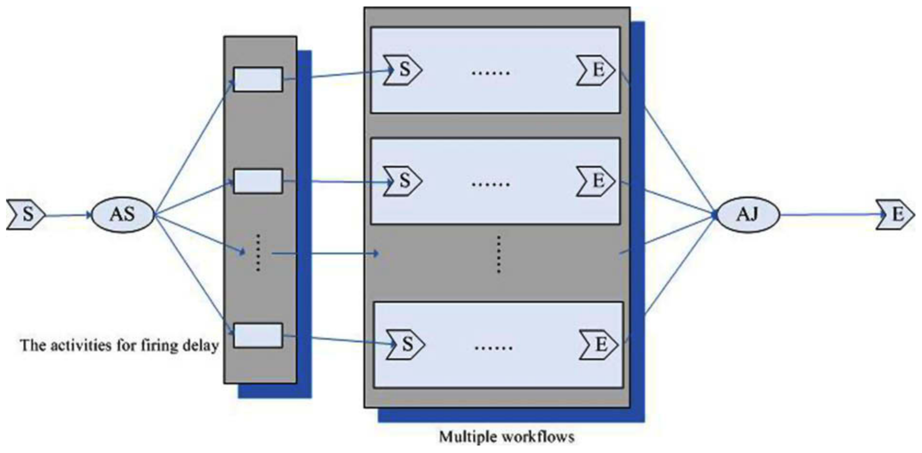


Fig. 8. The super workflow concept model for analysis of multiple workflows.

kept, and is used to calculate resource conflicts for each designer's edit operation. In order to support design process of workflows, relationships between edit operations and analysis methodologies need to be clarified; however, none of above work provide such discussion. In this paper, the works based on Hsu et al. (2005) are carefully rebuilt and refined, and in order to support design environment, the edit operations used in resource and temporal constraints are defined. The combination between edit operations and analyzing methodologies are described and discussed. In this paper, more features for workflow design are considered than in Li et al. (2004a), and more compact algorithms are constructed than in Hsu et al. (2005).

Based on the former research of static timing management of workflow specifications, H.C. Li continued his own research by dynamically analyzing the resource and temporal constraints between distinct workflow instances (Li and Yang, 2005). In Li and Yang (2005), the concept of reference points is introduced to show the relative timing constraints between the activities in different workflow instances. According to a pre-specified reference point in each workflow, all the timing constraints and EAIs are calculated. If any resource conflict exists, the EAIs of the processes involving the conflict are adjusted.

## 8. Conclusion and future work

Workflow specification is a formal description of workflow applications. A proper environment for verification of structural, resource, and timing constraints can help designers produce workflow applications of high quality. There're various effective approaches for verification of structural and temporal correctness. However there is little discussion for the analysis during specification process. This paper describes an incremental algorithm verifying resource conflicts and temporal constraints in a workflow specification after every edit operation. The relationship between each operation and the handling method is presented in detail. These methods are clarified also.

For future work, our methodology might be extended for multiple instances of the same resource type or multiple workflow specifications. To extend for multiple instances of the same resource type, a counter can be added to the resource model to indicate the number of instances of each resource type. For a resource type, when the number of references exceeds that of its counter, a resource conflict might happen.

Our methodology focuses on detecting resource conflicts in a workflow specification during design time. Sometimes, there are multiple workflow specifications involving to one single design work, and the detection of potential resource conflicts among the workflows is required. The workflows can be conceptually integrated into a super workflow to adopt our methodology for resource conflict detection. For example, as Fig. 8 shows, an and-split process follows the start process of the super workflow to simulate concurrent execution, and for each split path of the and-split, an activity process follows the and-split process for simulation of firing delay to the corresponding participated workflow. Each participated workflow starts after its firing delay, and is joined to an and-join process after its end. With the concept of the super workflow, multiple workflows can be viewed as one single workflow specification, and our methodology can almost be directly used for resource conflict detection among the participated workflows.

## References

Adam, N., Atluri, V., Huang, W., 1998. Modeling and analysis of workflows using petri-nets. Journal of Intelligent Information System 10, 131–158.

Allen, J.F., 1983. Maintaining knowledge about temporal intervals. Communication of the ACM 26 (11), 832–843.

Bajaj, A., Ram, S., 2002. SEAM: a state-entity-activity-model for a well defined workflow development methodology. IEEE Transactions on Knowledge and Data Engineering 14 (2), 415–432.

Chen, J., Yang, Y., Chen, T.Y., 2004. Dynamic Verification of temporal constraints on-the-fly for workflow systems. In: Proceedings of the 11th Asia-Pacific Software Engineering Conference, pp. 30–37.

Chinn, S.J., Madey, G.R., 2000. Temporal representation and reasoning for workflow in engineering design change review. IEEE Transactions on Engineering Management 47 (4), 485–493.

Eder, J., Panagos, E., Pozewaunig, H., Rabinovich, M., 1999a. Time management in workflow systems. In: Proceedings of International Conference Business Information Systems, pp. 266–280.

Eder, J., Panagos, E., Rabinovich, M., 1999b. Time constraints in workflow systems. In: Proceedings of 11th International Conferenc on Adv. Inf. Systems Engineering, Lecture Notes in Computer Science, vol. 1626, pp. 286–300.

Fleurke, M., Purvis, Ehrler L., 2003. JBees – an adaptive and distributed agent-based workflow system. In: Proceedings of the International Workshop on Collaboration Agents: Autonomous Agents for Collaborative Environments (COLA 2003), Halifax, Canada.

Had, Y., Jiang, C., Luo, X., 2005. Resource scheduling model for grid computing based on sharing synthesis of Petri-net, In: Proceedings of the Ninth International Conference on Computer Supported Cooperative Work in Design, pp. 367–372.

Hollingsworth, David, 1995. The Workflow Reference Model.

Hollingsworth, David, 2004. The Workflow Reference Model: 10 Years On.

Hsu, H.J., Yang, D.L., Wang, F.J., 2005. An incremental analysis to workflow specifications. In: Proceedings of the 12th Asia-Pacific Software Engineering Conference, pp. 122–129.

Kim, K. 2003 Workflow dependency analysis and its implications on distributed workflow systems, In: Proceedings of the 17th International Conference on Advanced Information Networking and Applications, pp. 677–683.

Li, H., Yang, Y., 2005. Dynamic checking of temporal constraints for concurrent workflows. Electronic Commerce Research and Applications 4, 124–142.

Li, J., Fan, Y., Zhou, M., 2003. Timing constraints workflow nets for workflow analysis. IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans 33 (2), 79–194.

Li, Hongchen, Yang, Yun, Chen, T.Y., 2004a. Resource constraints analysis of workflow specifications. Journal of System and Software 73 (2), 271–285.

Li, J., Fan, Y., Zhou, M., 2004b. Performance modeling and analysis of workflow. IEEE Transaction on Systems, Man, and Cybernetics – Part A: Systems and Humans 34 (2), 229–242.

Ling, S., Schmidt, H., 2000. Time Petri nets for workflow modelling and analysis. In: Proceedings of IEEE International Conference on Systems, Man and Cybernetics, pp. 3039–3044.

Marjanovic, O., 2000. Dynamic verification of temporal constraints in production workflows. In: Proceedings of the 11th Australian Database Conference, Canberra, Austalia, 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.

Park, J., Reveliotis, S., 2001. Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. IEEE Transaction on Automatic Control 46, 1572–1583.

Reichert, M., Dadam, P., 1998. ADEPT-supporting dynamic changes of workflows without losing control. Journal of Intelligent Information System 10 (2).

Reveliotis, S., 2003. Structural analysis of resource allocation systems with synchronization constraints. In: Proceedings of the 2003 IEEE International Conference on Robotics and Automation, pp. 1045–1049.

Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P., 2004. Workflow resource patterns, BETA Working Paper Series, WP 127, Eindhoeven University of Technology, Eindhoven, 2004.

Sadiq, W., Orlowska, M.E., 2000. Analyzing process models using graph reduction techniques. Information System 25 (2), 117–134.

Sadiq, S., Orlowska, M.E., Sadiq, W., Foulger, C., 2003. Data flow and validation in workflow modeling. In: Proceedings of Conferences in Research and Practice in Information Technology, vol. 27.

Singh, M.P., 1997. Formal aspects of workflow management, Part 1: Semantics, Technical Report, Department of Computer Science, North Carolina State University.

Sun, P., Wang, J., Li, X., Jiang, C., 2006. Performance analysis of workflow model with resource constraints, In: Proceedings of the First International Multi Symposiums on Computer and Computational Sciences, vol. 1, pp. 397–401.

Tang, Y., Chen, L., He, K., Jing, N., 2004. SRN: an extended Petri-net-based workflow model for Web service composition. In: Proceedings of IEEE International Conference on Web Services, pp. 591–599.

van der Aalst, W.M.P., 1998. The application of Petri nets to workflow management. Journal of Circuits, Systems, and Computers 8 (1), 21–46.

van der Aalst, W.M.P., ter Hofstede, A.H.M., 2000. Verification of workflow task structures: a petri-net-based approach. Information System 25 (1), 43–69.

van der Aalst, W.M.P., van Hee, K.M., van der Toorn, R.A., 1999. Adaptive workflow: an approach based on inheritance. In: Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business, Stockholm, Sweden.

WfMC, 1999. Management Coalition Terminology and Glossary, Document Number WFMC-TC-1011.

WfMC, 1993. Workflow Management Coalition, <http://www.wfmc.org/>.

Zaidi, A.K., 1999. On temporal logic programming using Petri nets. IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans 29 (3), 245–254.

Zhong, J., Song, B., 2005. Verification of resource constraints for concurrent workflows, In: Proceedings of the Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, pp. 253–261.