

Dynamic Verification of Temporal Constraints on-the-fly for Workflow Systems

Jinjun Chen¹, Yun Yang¹, T.Y. Chen²

¹ Centre for Internet Computing and E-Commerce (CICEC)

² Centre for Software Engineering (CSE)

School of Information Technology

Swinburne University of Technology

PO Box 218, Hawthorn, Melbourne, Australia 3122

{jchen, yyang, tchen}@it.swin.edu.au

Abstract

Temporal verification is an important method to check the temporal correctness of workflow management systems (WfMSs). However, the current temporal verification is relatively independent of the workflow system environments. It does not pay sufficient attention to the interrelationship between the temporal verification at different stages, the mutual dependency between some temporal constraints, and the run-time checkpoint selection strategy, which hence hinders the incorporation and consistency between the temporal verification and the workflow system environments. In this paper, we effectively integrate the temporal verification at different stages and explore the dependency between some temporal constraints. In addition, we present a new effective run-time checkpoint selection strategy which dynamically selects appropriate checkpoints based on the activity completion duration. Furthermore, based on these analyses, we develop some new methods for more efficient temporal verification. These analyses and new methods help to eliminate the gap between the temporal verification and the workflow system environments.

1. Introduction

Workflow technology has emerged as one of the important technologies designed to support modeling, redesign and execution of the business processes. According to the Workflow Management Coalition (WfMC), *workflow* is defined as the computerised facilitation or automation of a business process, in whole or part [13, 14]. Conceptually, a workflow consists of some activities and the dependencies between them. The dependencies form four basic control structures: sequential, parallel, selective and iterative, and four types of control nodes are introduced to represent parallel and selective structures, that is, and-split(*as*), and-join(*aj*), or-split(*os*) and or-join(*oj*) [13, 14]. 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 [13, 14]. Generally, a WfMS consists of two parts: build-time environment and run-time environment. The former defines, analyses and manages workflow specifications. The latter is responsible for the creation, execution and overall management of the workflow instances, and can be further divided into two stages: instantiation stage and execution stage. At the instantiation stage, workflow instances are created, and at the execution stage, the workflow instances are executed.

To control the temporal correctness of the workflow execution, designers often set some explicit temporal constraints when they define a workflow at build time. Then, the temporal verification is conducted to check if all temporal constraints are consistent. Corresponding to the different parts and stages of WfMSs, the temporal verification can be divided into three stages: build-time stage, run-time instantiation stage and run-time execution stage. Among these stages, the execution stage temporal verification is more complicated because the activity completion duration is not certain at this stage. To conduct it efficiently, we should select some appropriate activity points as checkpoints for carrying out the verification computation because normally it is unnecessary to do so at each activity point.

Temporal verification should be linked to the workflow system environments. In the environments, the temporal verification at different stages is interrelated and some temporal constraints have strong mutual dependency which affects the consistency of temporal constraints. Especially, the run-time checkpoint selection strategy should dynamically reflect the uncertainty of the activity completion duration so that the run-time temporal verification is more consistent with the run-time workflow system environments. However, the current temporal verification in practice does not pay sufficient attention to these linkage factors, which hinders the consistency and

incorporation between the temporal verification and the workflow system environments. [8, 9, 10] propose some algorithms to adjust activity deadlines and estimate the escalations and assign the predictive deadlines. [4] uses the modified Critical Path Method (CPM) to calculate temporal constraints and is one of the very few attempts that consider temporal constraint reasoning at both build-time and run-time. At run-time, it takes every activity as a checkpoint. [7] presents a method for dynamic verification of the absolute deadline constraints and relative deadline constraints, and takes the beginning point of a workflow instance as a checkpoint and adds a checkpoint after each decision node is executed. [15] proposes a timed workflow process model with the consideration of the flow time, the time difference in a distributed execution environment and consistency checking combining the exact run-time duration and the estimated build-time duration, and sets checkpoints at the start time and end time of each activity and each flow. However, all these works and some others such as [1, 2, 3, 5, 6, 11] do not effectively integrate the temporal verification at different stages and neglects the dependency between some temporal constraints. In addition, some of them have no strategies for checkpoint selection. For those which have checkpoint selection, [4, 15] may lead to a lot of unnecessary temporal verification, whilst [7] and another popular checkpoint selection strategy: user-defined checkpoints may lead to some unnecessary and omitted temporal verification. In our paper, we borrow some concepts from [4, 7, 15] such as temporal constraints, the minimum and maximum durations as the basis for the workflow representation.

Based on these problems of the current temporal verification, this paper integrates the temporal verification at different stages, and investigates how the dependency between some temporal constraints affects the temporal consistency, and presents a new run-time checkpoint selection strategy which shows how to dynamically select appropriate checkpoints based on the activity completion duration along the workflow execution. At the same time, based on these analyses, this paper develops some new methods for more efficient temporal verification. These analyses and new methods make the temporal verification more consistent with the real-world workflow system environments.

The remainder of the paper is organised as follows. Section 2 describes preliminaries for the workflow representation and temporal constraints. Section 3 summarises the build-time temporal constraint verification and dependency. Section 4 details the deadline constraint dependency, our checkpoint selection strategy and run-time temporal constraint verification at the instantiation and execution stages. Section 5 shows the benefits of our work through comparison and discussion. Finally, section 6 concludes our contributions and points out the future work.

2. Preliminaries

2.1. Workflow representation

Before we conduct temporal verification, we must represent workflow with some time attributes which are expressed in some basic time units, such as minutes, hours, or days. However, the detailed discussion of new timed workflow models is beyond the scope of this paper. Rather, we describe a generic timed workflow representation for presenting our work. Based on directed graph, a workflow can be represented by a workflow graph, where nodes correspond to activities and edges correspond to dependencies between activities, called flows. Here, we assume that execution dependencies between activities form an acyclic directed graph because there is logically no repeated execution in terms of temporal logic, and we also assume the workflow is well structured [4]. In addition, from the time perspective, an activity and a flow can be treated in the same way. So, in the remainder of this paper, we use term ‘activity’ to refer to the real activity and the flow and we will not distinguish them if not explicitly mentioned. We denote the i^{th} activity of a workflow as a_i . For each a_i , we denote the maximum duration, the minimum duration, the run-time start time, and the run-time end time as $D(a_i)$, $d(a_i)$, $S(a_i)$ and $E(a_i)$ respectively. If there is a path from a_i to a_j ($j \geq i$), we denote the maximum duration, the minimum duration and the run-time real completion duration between them as $D(a_i, a_j)$, $d(a_i, a_j)$ and $Rcd(a_i, a_j)$ respectively. For the convenience of the discussion, in the remainder of this paper, we only consider one execution path in the workflow and assume that all activities which we will address are on this execution path. For every other execution path, the discussion is very similar. In addition, we assume that the activity numbering on the path is $1, 2, 3, \dots, i, i+1, i+2, \dots, i+(j-i-1), j, j+1, j+2, \dots$. For the computation of $D(a_i, a_j)$ and $d(a_i, a_j)$, if there are no parallel or selective structures between a_i and a_j , we can use the following formula to compute them.

$$D(a_i, a_j) = \sum_{k=i}^j D(a_k), \quad d(a_i, a_j) = \sum_{k=i}^j d(a_k)$$

If there are parallel or selective structures between a_i and a_j , for each structure, we use the largest of the maximum durations of all branches as the maximum duration of the structure, and use the largest of the minimum durations of all branches as the minimum duration of the structure. Then, if we see every structure as an activity in terms of time, we are still able to use the above formula to compute $D(a_i, a_j)$ and $d(a_i, a_j)$. Moreover, we have: $Rcd(a_i, a_j) = E(a_j) - S(a_i)$.

In addition, activity a_i or workflow systems should keep the following time attributes for fast computation:

- a) Run-time completion duration for completing activity a_i , denoted as $Rcd(a_i) = E(a_i) - S(a_i)$
- b) Accumulative extra time added by the possible temporal adjustments due to the violation of former temporal constraints, denoted as $Acc(a_i)$
- c) Run-time execution duration including the possible queuing time and synchronisation waiting time and other time except the accumulative extra time, denoted as $Ex(a_i)$

According to the above denotations, we have: $D(a_i, a_i) = D(a_i)$, $d(a_i, a_i) = d(a_i)$, $Rcd(a_i, a_i) = Rcd(a_i)$, $Rcd(a_i) = Ex(a_i) + Mt(a_i)$. Also, we define: $D(a_m, a_i) = 0$ ($m > i$), $d(a_m, a_i) = 0$ ($m > i$), $Rcd(a_m, a_i) = 0$ ($m > i$). Normally, $d(a_i) \leq Rcd(a_i) \leq D(a_i)$, $d(a_i, a_j) \leq Rcd(a_i, a_j) \leq D(a_i, a_j)$.

2.2. Temporal constraints

According to [4, 14], the main explicit temporal constraints include:

Upper Bound Constraint: The duration between a_i and a_j must be less than or equal to a relative time value. We denote this value as $upb(a_i, a_j)$.

Lower Bound Constraint: The duration between a_i and a_j must be bigger than or equal to a relative time value. We denote this value as $lob(a_i, a_j)$.

Deadline Constraint: A certain activity, say a_i , must be completed by a certain time. We denote this certain time as $deadline(a_i)$.

The upper bound and the lower bound constraints need to be verified at the build-time stage, the run-time instantiation and execution stages. At build-time, the designers may address deadlines, but only until the instantiation stage, are deadlines set to specific absolute time values. So, to be consistent with the workflow systems, in this paper, deadline constraints are verified at the instantiation and execution stages. This also can support the situations where there may have some changes on the deadline constraint setting at the instantiation and execution stages. The temporal verification is conducted step by step. Firstly, the temporal verification is conducted at the build-time stage, then at the instantiation stage, and finally at the execution stage. When the temporal verification reaches a certain stage, we can assume that the temporal constraints at former stage(s) be consistent. If at a certain stage, one of the temporal constraints is violated, we may use some methods to adjust the time scheduling to make it consistent. So, at the execution state, when the workflow execution has reached activity a_j , we can assume that temporal constraints before the execution of activity a_j be consistent. In addition, we assume the adjustment methods themselves do not result in new violations of temporal constraints. Otherwise, they will affect the consistency of the temporal constraints and hence cannot be used. Moreover, in the real-world workflow system environments, an activity normally at most has one deadline and two activities at most have one upper bound

constraint or lower bound constraint. Therefore, we just assume so in this paper. Otherwise, the temporal verification will probably not be consistent with the real-world workflow system environments.

3. Summary of build-time temporal verification

At build-time, the existing verification methods [7, 15] can be reused. Because we will integrate temporal verification at different stages, we simply give a summary in this section.

Definition 1. The upper bound constraint between a_i and a_j ($j > i$) is consistent at build-time if and only if $D(a_i, a_j) \leq upb(a_i, a_j)$, and the lower bound constraint between a_i and a_j ($j > i$) is consistent if and only if $d(a_i, a_j) \geq lob(a_i, a_j)$.

According to definition 1, if the upper bound and lower bound constraints are consistent individually, they do not affect one another's temporal consistency. So, we do not consider the dependency between the upper bound constraints or lower bound constraints.

4. Run-time temporal verification

4.1. Instantiation stage

At this stage, workflow instances are created and the deadlines of some activities are set to specific absolute time values. So, we will be able to get two kinds of absolute times which are the start time of the workflow instance (i.e. $S(a_i)$) and deadlines. So, we can verify the deadline constraints. However, we do not have absolute times of specific activities because they are not being executed. So, the verification of the upper bound and lower bound constraints is the same as the build-time verification. We just use the verification results of build-time. About the consistency of the deadline constraints, we have:

Definition 2. The deadline constraint at a_i at the instantiation stage is consistent if and only if $D(a_i, a_i) \leq deadline(a_i) - S(a_i)$.

Based on definition 2, for every activity a_i which has a deadline constraint, we calculate $D(a_i, a_i)$ according to the computing formula described in section 2.1. Then, we compare this sum with the difference between $deadline(a_i)$ and $S(a_i)$. If the compared result meets definition 2, the deadline constraint at a_i is consistent. If not, we should reschedule the deadline assignment before the workflow instance is submitted to execute.

When the deadlines are assigned or rescheduled or the deadline constraint verification is conducted, we must consider the dependency between deadlines although this will increase the complexity of the temporal verification. The reason is detailed as follows. Suppose there are two

activities $a_i, a_j (j>i)$ which both have deadlines. Obviously, we should conduct the deadline constraint verification. We suppose the verification results show that the deadline constraints at a_i and a_j are consistent. That is to say, the following two inequations hold:

$$D(a_i, a_j) \leq \text{deadline}(a_j) - S(a_i) \quad (1)$$

$$D(a_i, a_j) \leq \text{deadline}(a_j) - S(a_i) \quad (2)$$

Then, if we omit the dependency between deadlines this workflow instance will be submitted to execute. However, considering the extreme case, when the workflow arrives at $a_i, E(a_i)=\text{deadline}(a_i)$. In this situation, to ensure the deadline at a_j is met, at the instantiation stage, we must make sure the following inequation holds.

$$D(a_{i+1}, a_j) \leq \text{deadline}(a_j) - \text{deadline}(a_i) \quad (3)$$

If we take a careful look at (1)(2)(3), it is not difficult for us to find the following equations:

$$D(a_{i+1}, a_j) = D(a_i, a_j) - D(a_i, a_i) \quad (4)$$

$$\text{deadline}(a_j) - \text{deadline}(a_i) = [\text{deadline}(a_j) - S(a_i)] - [\text{deadline}(a_i) - S(a_i)] \quad (5)$$

(4) shows that the left-hand side of (3) is equal to the difference between the left-hand sides of (2) and (1). (5) shows that the right-hand side of (3) is equal to the difference between the right-hand sides of (2) and (1).

The problem is that we cannot derive (3) from (1) and (2) that we only have. The reason is simple. Suppose (1) is $2<6.5$ and (2) is $6<9$. We will have: $4=6-2$ and $2.5=9-6.5$. Obviously, $4>2.5$, not $4<2.5$ that we want for (3). So, even if the deadline constraints at a_i and a_j are consistent according to definition 2, we still need to consider the dependency between them. Otherwise, they may not be consistent in the overall term though they are consistent individually. So, we have:

Definition 3. The dependency between two adjacent deadline constraints at a_i and $a_j (j>i)$ is consistent if and only if (3) holds.

Definition 3 ensures the dependency consistency between two adjacent deadlines. However, we still need to consider the dependency consistency between any two non-adjacent deadlines. The following theorem solves this problem:

Theorem 1. If the dependency between any two adjacent deadline constraints is consistent, the dependency between any two non-adjacent deadline constraints must be consistent.

Proof: Suppose each of a_i, a_j and $a_l (l>j>i)$ has a deadline constraint, and the dependency between a_i and a_j is consistent, and also, the dependency between a_j and a_l is consistent. That is to say, (3) and the following inequation hold:

$$D(a_{j+1}, a_l) \leq \text{deadline}(a_l) - \text{deadline}(a_j) \quad (6)$$

Now, we prove that the dependency between a_i and a_l is also consistent. According to definition 3, we must prove the following inequation holds:

$$D(a_{i+1}, a_l) \leq \text{deadline}(a_l) - \text{deadline}(a_i) \quad (7)$$

If we add (3) and (6) together, we will derive (7). So, Theorem 1 holds.

Based on the above discussion, at the instantiation stage, on one hand, we conduct the deadline temporal verification based on definition 2; on the other hand, we still need to verify the dependency between adjacent deadline constraints based on definition 3. Based on theorem 1, we need not consider the dependency between non-adjacent deadline constraints. This will save some verification computations.

4.2. Execution stage

At the execution stage, the activity completion duration is not certain, which may affect the consistency of the temporal constraints which are consistent at the build-time and instantiation stages. So we need to re-verify the temporal constraints. However, to be efficient, we should conduct the temporal verification only at the selected checkpoints.

Based on the definitions of temporal constraint consistency at the build-time and instantiation stages, we first give definitions of temporal constraint consistency at the execution stage.

Definition 4. The upper bound constraint between a_i and $a_l (l>i)$ is consistent at a checkpoint before a_i and a_l at the execution stage if and only if $D(a_i, a_l) \leq \text{upb}(a_i, a_l)$, and the lower bound constraint between activities a_i and $a_l (l>i)$ at a checkpoint before a_i and a_l at the execution stage is consistent if and only if $d(a_i, a_l) \geq \text{lob}(a_i, a_l)$.

Definition 5. The upper bound constraint between a_i and a_l is consistent at checkpoint a_j between a_i and $a_l (l>j, j>i)$ at the execution stage if and only if $Rcd(a_i, a_l) + D(a_{j+1}, a_l) \leq \text{upb}(a_i, a_l)$, and the lower bound constraint between a_i and a_l is consistent at checkpoint $a_j (l>j, j>i)$ between a_i and a_l at the execution stage if and only if $Rcd(a_i, a_j) + d(a_{j+1}, a_l) \geq \text{lob}(a_i, a_l)$.

Definition 6. The upper bound constraint between a_i and $a_l (l>i)$ is consistent at a checkpoint after a_i and a_l at the execution stage if and only if $Rcd(a_i, a_l) \leq \text{upb}(a_i, a_l)$, and the lower bound constraint between a_i and $a_l (l>i)$ is consistent at a checkpoint after a_i and a_l at the execution stage if and only if $Rcd(a_i, a_l) \geq \text{lob}(a_i, a_l)$.

Definition 7. The deadline constraint at a_n is consistent at checkpoint a_j by $a_n (j\leq n)$ at the execution stage if and only if $Rcd(a_i, a_j) + D(a_{j+1}, a_n) \leq \text{deadline}(a_n) - S(a_i)$.

Definition 8. The deadline constraint at a_n is consistent at checkpoint a_j after $a_n (j>n)$ at the execution stage if and only if $Rcd(a_i, a_n) \leq \text{deadline}(a_n) - S(a_i)$.

4.2.1. Checkpoint selection. Based on the problems of the current popular checkpoint selection strategies in practice described in section 1, we provide a new checkpoint

selection strategy which dynamically selects appropriate checkpoints based on the activity completion duration for conducting the temporal verification.

Our checkpoint selection strategy in general is that along the execution of a workflow instance, when the execution reaches activity a_j , (1) if $d(a_j) \leq Ex(a_j) + Acc(a_j) \leq D(a_j)$, we do not take a_j as a checkpoint; (2) if $d(a_j) > Ex(a_j) + Acc(a_j)$, we take a_j as a checkpoint for the lower bound constraints which include a_j ; and (3) if $Ex(a_j) + Acc(a_j) > D(a_j)$, we take a_j as a checkpoint for the upper bound and deadline constraints including a_j .

According to definitions 4,5,6,7,8, it is not difficult to understand our strategy. However, to be clear, we simply explain its rationale. At activity a_j , for those temporal constraints which do not include a_j , we need not take a_j as a checkpoint because the execution of a_j has nothing to do with their consistency. Now, we consider the temporal constraints which include a_j .

If $d(a_j) \leq Ex(a_j) + Acc(a_j) \leq D(a_j)$, suppose there is an upper bound constraint between a_i and a_l ($l \geq j, j \geq i$). Based on definition 5, before the execution of a_j , we have $Rcd(a_i, a_{j-1}) + D(a_j, a_l) \leq upb(a_i, a_l)$. On the other hand, $Rcd(a_i, a_{j-1}) + D(a_j, a_l) = Rcd(a_i, a_{j-1}) + D(a_j) + D(a_{j+1}, a_l) \geq Rcd(a_i, a_{j-1}) + Ex(a_j) + Acc(a_j) + D(a_{j+1}, a_l) = Rcd(a_i, a_j) + D(a_{j+1}, a_l)$. So, we have $Rcd(a_i, a_j) + D(a_{j+1}, a_l) \leq upb(a_i, a_l)$. According to definition 5, the upper bound constraint between a_i and a_l is consistent. Similarly, we can prove that the lower bound constraints and the deadline constraints are also consistent. So, we need not take a_j as a checkpoint.

If $d(a_j) > Ex(a_j) + Acc(a_j)$, obviously, the execution of a_j will not affect the consistency of the upper bound constraints and the deadline constraints. Now, we consider the lower bound constraints. Suppose there is a lower bound constraint between a_i and a_l ($l \geq j, j \geq i$). According to definition 5, before execution of a_j , we have $Rcd(a_i, a_{j-1}) + d(a_j, a_l) \geq lob(a_i, a_l)$. On the other hand, $Rcd(a_i, a_{j-1}) + d(a_j, a_l) = Rcd(a_i, a_{j-1}) + d(a_j) + d(a_{j+1}, a_l) > Rcd(a_i, a_{j-1}) + Ex(a_j) + Acc(a_j) + d(a_{j+1}, a_l) = Rcd(a_i, a_j) + d(a_{j+1}, a_l)$. However, based on these two conditions which we only have, we cannot definitely derive $Rcd(a_i, a_j) + d(a_{j+1}, a_l) \geq lob(a_i, a_l)$ which, according to definition 5, is a must for the consistency of the lower bound constraint. So, we need to take a_j as a checkpoint.

If $Ex(a_j) + Acc(a_j) > D(a_j)$, similar to the situation of $d(a_j) > Ex(a_j) + Acc(a_j)$, we are able to prove that we need to take a_j as a checkpoint for the upper bound constraints and the deadline constraints.

We call a_j a *consistent activity* if $d(a_j) \leq Ex(a_j) + Acc(a_j) \leq D(a_j)$. And we call the activity taken as a checkpoint by our strategy *inconsistent activity*.

From the above discussion, we can see that our strategy dynamically selects checkpoints during the execution of a workflow instance based on the completion duration of each activity which reflects the state change of

the workflow system. Also, the above discussion shows that our strategy is complete and effective for the temporal verification because only at these checkpoints the execution of a workflow instance possibly violates some temporal constraints but does not violate temporal constraints at all other points.

4.2.2. Temporal verification at execution stage. On the fly with the execution of workflow, at every checkpoint selected based on our selection strategy, we conduct the temporal verification. However, based on the above discussion, we only need to re-consider the temporal constraints which include the checkpoint.

Along the execution of each workflow instance, when the execution reaches an activity, say, a_j , we calculate the sum of $Ex(a_j)$ and $Acc(a_j)$. Before the execution of a_j , there may be some temporal adjustments which allocate some time deficit to a_j to make previously violated temporal constraints consistent again. So, we must take $Acc(a_j)$ into consideration. And then, we compare this sum with $d(a_j)$ and $D(a_j)$. If $d(a_j) \leq Ex(a_j) + Acc(a_j) \leq D(a_j)$, the execution of a_j does not affect the consistency of the temporal constraints. So, we need not conduct any verification further and based on the verification computation conducted before and at the build-time and instantiation stages, the temporal constraints are still consistent for the time being.

If $d(a_j) > Ex(a_j) + Acc(a_j)$, then, if there are no any lower bound constraints which include a_j , we need not conduct any temporal verification. If there are, we take a_j as a checkpoint and re-compute their consistency. Suppose one of these lower bound constraints is between a_i and a_l ($l \geq j, j \geq i$), based on definition 5, we compute $Rcd(a_i, a_j) = E(a_j) - S(a_i)$ and $d(a_{j+1}, a_l)$, and then add them together to derive a sum. If the sum is bigger than or equal to $lob(a_i, a_l)$, this lower bound constraint is still consistent. But if the sum is less than $lob(a_i, a_l)$, this lower bound constraint is violated. And then, some time adjustment measures may be taken or some other special actions (called escalation in [13]) will be triggered to handle the violation such as replacing the triggering activity with a new activity, or aborting the workflow process and compensating all finished activities, and so on.

If $Ex(a_j) + Acc(a_j) > D(a_j)$, then, if there are no any upper bound constraints and deadline constraints which include a_j , we do not have to conduct any temporal verification. If there are upper bound constraints or deadline constraints including a_j , we take a_j as a checkpoint. For the upper bound constraint verification, it is similar to the lower bound constraint verification described above, so we simply omit its description. Now, we consider the deadline constraints.

Theorem 2. When the execution of a workflow instance reaches a checkpoint at a_j , if a deadline constraint

at an activity after a_j is consistent, any deadline constraint after this consistent deadline constraint must be consistent.

Proof: Suppose a_n has a deadline constraint and a_s also has one ($j \leq n < s$) and the deadline constraint at a_n is consistent. Now, we prove the deadline constraint at a_s is also consistent. Based on definition 7, we have:

$$Rcd(a_1, a_j) + D(a_{j+1}, a_n) \leq \text{deadline}(a_n) - S(a_1) \quad (8)$$

To prove the consistency of the deadline constraint at a_s , based on definition 7, we only need to prove the following inequation holds.

$$Rcd(a_1, a_j) + D(a_{j+1}, a_s) \leq \text{deadline}(a_s) - S(a_1) \quad (9)$$

Based on the verification computation conducted at the instantiation stage, we have:

$$D(a_{n+1}, a_s) \leq \text{deadline}(a_s) - \text{deadline}(a_n) \quad (10)$$

If we add (8) and (10) together, we will derive (9). So, theorem 2 holds.

According to theorem 2, at a checkpoint, from the first deadline constraint after it, we conduct temporal verification one after another until we meet one consistent deadline constraint or finish all deadline constraint verification. Suppose we now reach the deadline constraint at a_n , we compute $Rcd(a_1, a_j) = E(a_j) - S(a_1)$ and $D(a_{j+1}, a_n)$, then add them together to derive a sum. If the sum is less than or equal to the difference between $\text{deadline}(a_n)$ and $S(a_1)$, the deadline constraint at a_n is consistent and according to theorem 2, we need not conduct further verification computation for any deadline constraint after a_n . This will save some verification computations and improve the efficiency of the temporal verification. If not, the deadline constraint at a_n is violated and we still need to conduct verification computation for the next deadline constraint.

5. Comparison and discussion

On the fly with the workflow system environments, we integrate the verification computation of different stages. At the instantiation stage, we utilise the build-time temporal verification results of the upper bound and lower bound constraints. At the execution stage, based on the build-time and instantiation stage temporal verification and the dependency between deadline constraints conducted at the instantiation stage, we save the following computations:

- Any temporal verification if $d(a_j) \leq Ex(a_j) + Acc(a_j) \leq D(a_j)$;
- The upper bound and deadline constraint verification if $d(a_j) > Ex(a_j) + Acc(a_j)$;
- The lower bound constraint verification if $Ex(a_j) + Acc(a_j) > D(a_j)$;
- The deadline verification after a consistent deadline constraint.

All these integration measures make the temporal verification in WfMSs more efficient and compact, which

further contributes to the incorporation of the temporal verification into workflow systems.

At the instantiation and execution stages, we address the dependency between the deadline constraints and the accumulative extra time added by the possible temporal adjustments due to the violation of temporal constraints. According to the discussion in section 4.1, we can see that the dependency between the deadline constraints does really possibly affect the consistency of the deadline constraints. The previous verification work does not pay much attention to these factors. So, our work reinforces the temporal verification in WfMSs and also contributes to the consistency between the temporal verification and the workflow system environments.

As described in section 1, current popular checkpoint selection strategies mainly include: (a) checkpoints at every activity; (b) checkpoints at the start time and the end time of each activity and each flow; (c) checkpoints after each decision node is executed to decide which path to go; (d) user-defined checkpoints. To compare our checkpoint selection strategy with them, we use a practical example which can clearly reflect the difference between them. Figure 1 is the graphic representation of a workflow specification originated from [12, 6] with minor modifications. It depicts the processing of the payment requests in an Australian organisation.

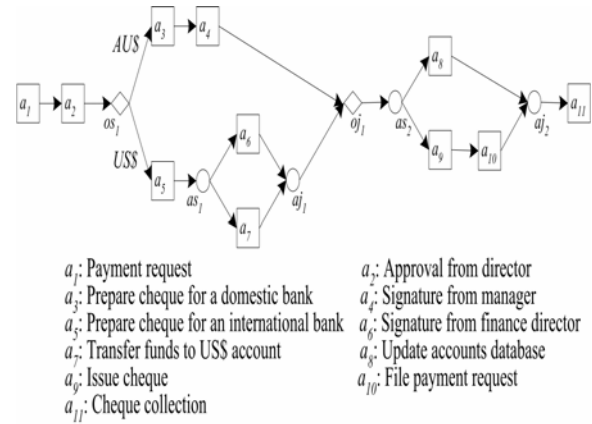


Figure 1. Payment request workflow process

Now, suppose a specific workflow instance taking the US\$ path with activity completion durations exceeding the corresponding activity maximum durations at activities a_2 , a_6 , a_{10} , but being consistent at all other activities and flows. According to the discussion in section 4.2.1, except for a_2 , a_6 , a_{10} , we need not conduct the temporal verification at any other activities or flows. Based on our checkpoint selection strategy, we can see that the selected checkpoints are just a_2 , a_6 , a_{10} . Therefore, we will not omit some temporal verifications or conduct some unnecessary temporal verifications. For (a), except the temporal verification at a_2 , a_6 , a_{10} , all other temporal verifications at

$a_1, a_5, a_7, a_8, a_9, a_{11}$ respectively are not necessary. For (b), except the temporal verification at the end times of activities a_2, a_6, a_{10} , all other temporal verifications at the end times of other activities and flows and the start times of all activities and flows are not necessary. For (c), there are three checkpoints respectively at os_1, as_1, as_2 . Because the execution of the workflow instance is consistent at all flows, we can use the checkpoint at os_1, as_2 to conduct the temporal verification for a_2, a_6 respectively although there is a delay. So, the temporal verification at as_1 is not necessary and the temporal verification at a_{10} is omitted. For (d), the checkpoints are selected in advance of the execution of the workflow instance. So, this strategy is static and may not be accurate. If the checkpoints are just at activities a_2, a_6, a_{10} and there are no other checkpoints, no temporal verification is omitted or conducted unnecessarily at other activities or flows. Otherwise, there may have some unnecessary and/or omitted temporal verifications. Moreover, even if the user defined checkpoints are just at a_2, a_6, a_{10} , for the current workflow instance, there are no omitted or unnecessary temporal verifications. However, at run-time, normally, there are lots of workflow instances. For those workflow instances whose inconsistent activities are not only among a_2, a_6, a_{10} , there still have some omitted and/or unnecessary temporal verifications.

In summary, based on our checkpoint selection strategy, for any workflow instances, there is no omitted temporal verification, and there is no temporal verification unnecessarily conducted at consistent activities or flows. So, finally, we can conclude that our checkpoint selection strategy for the run-time temporal verification is more effective than other four existing popular checkpoint selection strategies.

It may be argued that our checkpoint selection strategy needs to conduct judgment computations at each activity or flow to decide whether we should take the activity or flow as a checkpoint and the current four popular checkpoint selection strategies need not conduct this kind of computation. It is true, however, the judgment computations use the audit log data collected and maintained by WfMSs and hence is very simple. So, the judgment computations will not affect the overall effectiveness of our checkpoint selection strategy.

6. Conclusions and future work

In this paper, with the focus on the on-the-fly temporal verification for the workflow systems, the temporal verification at different stages is integrated and the dependency between deadline constraints is investigated at run-time instantiation and execution stages. In addition, a new effective run-time checkpoint selection strategy is presented based on the activity completion duration which includes the accumulative extra time added by the possible

temporal adjustments due to the violation of temporal constraints. This strategy dynamically selects checkpoints on the fly with the execution of workflow instances. In fact, different workflow instances may have different checkpoints, which is reasonable to the run-time evolving workflow system environments. At the same time, based on these analyses, some new methods are developed, which make the temporal verification more efficient because we can fully use previous verification results and avoid unnecessary temporal verification as well. These analyses and new methods can help us to eliminate the gap between the temporal verification and the workflow system environments and eventually strengthen the time management of WfMSs.

With these contributions, we can further consider some problems such as predictive schemas of the temporal constraint violation, and temporal adjustment methods when a temporal constraint is violated.

Acknowledgements

The work reported in this paper is partly supported by Swinburne Vice Chancellor's Strategic Research Initiative Grant 2002-2004 for project "Internet-based e-business ventures" and by ARC Australian Research Council Discovery Project Scheme under Grant No. DP0345147.

References

- [1] N. Adam, V. Atluri, and W. Huang, Modeling and analysis of workflows using petri nets, *Journal of Intelligent Information Systems, Special Issue on Workflow and Process Management*, 1998, 10(2), pp.131-158.
- [2] C. Bussler, Workflow instance scheduling with project management tools, In *Proc. of the 9th Workshop on Database and Expert Systems Applications DEXA'98*, IEEE Computer Society Press, Vienna, Austria, 26-28 Aug. 1998, pp.753-758.
- [3] S. Chinn and G. Madey, Temporal representation and reasoning for workflow in engineering design change review, *IEEE Transactions on Engineering Management*, 2000, 47(4), pp. 485-492.
- [4] J. Eder, E. Panagos, and M. Rabinovich, Time constraints in workflow systems, In *Proc. of the 11th International Conference on Advanced Information Systems Engineering(CAiSE'99)*, Springer Verlag, LNCS 1626, Germany, June 1999, pp.286-300.
- [5] J. Eder, H. Pichler, W. Gruber, and M. Ninaus, Personal schedules for workflow systems, In *Proc. of the Conference on Business Process Management BPM'03*, Eindhoven, The Netherlands, June 26-27, 2003, pp.216-231.

- [6] H. Li, Y. Yang, and T.Y. Chen, Resource constraints analysis of workflow specifications, *Journal of Systems and Software*, 2004, 73(2), pp. 271-285.
- [7] O. Marjanovic, Dynamic verification of temporal constraints in production workflows, In *Proc. of the Australian Database Conference*, IEEE Press, Canberra, Australia, Feb. 2000, pp. 74-81.
- [8] E. Panagos, M. Rabinovich, Escalations in workflow management systems, In *Proc. of the Workshop on Databases: Active and Real time*, ACM Press, Rockville, Maryland, United States, Nov. 1996, pp. 25-28.
- [9] E. Panagos, M. Rabinovich, Predictive workflow management, In *Proc. of the 3rd Workshop on the Next Generation Information Technology and Systems (NGITS-97)*, Neve Ilan, ISRAEL, June 1997, pp. 193-197.
- [10] E. Panagos, M. Rabinovich, 1997b. Reducing escalation-related costs in WFMSs, In A., Dogac, et al., editor, *NATO Advanced Study Institute on Workflow Management Systems and Interoperability*, Istanbul, Turkey, Aug. 1997.
- [11] H. Pozewaunig, J. Eder, and W. Liebhart, ePERT: Extending PERT for workflow management systems, In *Proc. of the 1st EastEuropean Symposium on Advances in Database and Information Systems ADBIS 97*, St. Petersburg, Russia, Sept. 1997, pp.217-224.
- [12] W. Sadiq, M.E. Orłowska, Analysing process models using graph reduction techniques, *Information Systems*, 2000, 25(2), pp. 117-134.
- [13] WfMC, The Workflow Reference Model, Document Number TC00-1003, 1995, <http://www.wfmc.org/>.
- [14] WfMC, Terminology & Glossary, Document Number WfMC-TC-1011, 1999, <http://www.wfmc.org/>.
- [15] H. Zhuge, T.Y. Cheung, and H.K. Pung, A timed workflow process model, *Journal of Systems and Software*, 2001, 55(3), pp. 231-243.