Methods of Computing Change Times of Dynamic Workflow Changes

Shingo Yamaguchi†, Qi-Wei Ge‡, and Minoru Tanaka†

†Faculty of Engineering, Yamaguchi University
2-16-1 Tokiwadai, Ube, 755-8611 Japan
Tel: +81-836-85-9511, Fax: +81-836-85-9501
E-mail: {shingo, tanaka}@csse.yamaguchi-u.ac.jp

‡Faculty of Education, Yamaguchi University
1677-1 Yoshida, Yamaguchi, 753-8513 Japan
Tel: +81-83-933-5401, Fax: +81-83-933-5304
E-mail: gqw@inf.edu.yamaguchi-u.ac.jp

Abstract: A workflow is a flow of work carried out by multiple people. In order to increase efficiency, it is required to change the current workflow dynamically. Till now, three types of dynamic changes: flush, abort, and synthetic cut-over (SCO), have been proposed. However, the performance evaluations for dynamic workflow changes have not been undertaken. To do so, measuring the amount of time cost making a single change (called *change time*) and comparing the methods for obtaining such times become ever important. In this paper, we first define change time and then propose a computation method individually for each change type. Finally, we evaluate the performance of an example net change by computing the change times.

1. Introduction

A workflow is a flow of work that is carried out, in a businesses, by several people in parallel and in series. Today's information systems are required to support not only the execution of individual work but also that of workflows. A workflow management system is such an information system that defines, creates, and manages the execution of workflows [1]. Since it provides procedural automation of a workflow, it can get rid of work stagnations or trivial errors in passing through the workflow.

A workflow needs to be constantly refined in order to effectively meet the constraints and opportunities posed by new technology, new market requirements, and new laws [2]. Using a workflow management system, it is relatively easy to change the structure of a workflow, because one need only to change the definitions stored in the workflow management system. On the other hand, with widespread use of workflow management systems, the coverage of workflow has become wide up. In changing part of a large workflow, it is difficult to suspend the workflow management system, and thus it is necessary to change the workflow while the system is running. Such a workflow change is called *dynamic workflow change* [3]. Dynamic workflow changes would probably make the workflow inconsistent and inefficient, and therefore it is important to analyze the mechanism and further to evaluate the performance of dynamic workflow changes.

Concerning the structural change of workflows, various studies have been conducted. Herrmann et al. have argued about negotiations with workflow changes [4]. Jager et al. have attempted automatic improvement of workflows [5]. These studies are not concerned with dynamic workflow changes. Ellis et al. have been concerned with dynamic workflow changes and proposed three types of such changes—flush, abort, and synthetic cut-over— that keep consistent for the workflows [3]. However, the quantitative evaluation of these three dynamic changes has not been done.

In this paper, we aim to quantitatively evaluate the performance of Ellis's three dynamic changes. To do the performance evaluation, it is important to compare the time (called *change time*) cost in an individual dynamic change. Firstly we define change time. Then we propose methods for computing change times for the three dynamic changes. Finally, we evaluate the performance of an example net change by computing the change times.

2. Preliminary

A workflow is a flow of work carried out by multiple people (called *resources* hereafter). An individual work is called an *activity*, and an individual enactment of a workflow definition is called a *case*. In general, many cases are handled according to the same workflow definition, and they are handled in the order of "First-In First-Out" (FIFO).

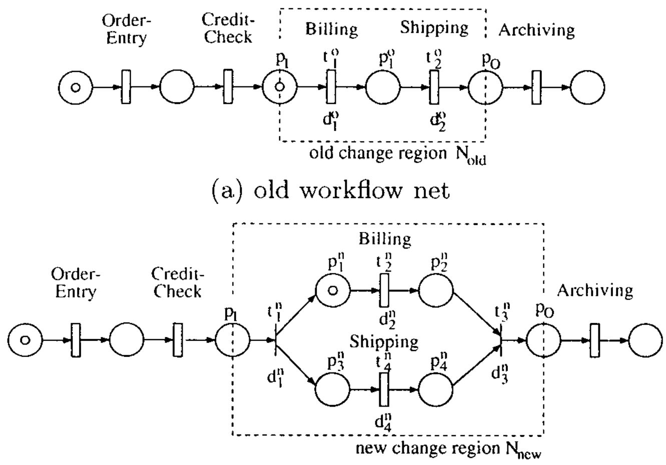
2.1 Workflow nets

A Petri net modeled workflow, called *workflow net* and denoted as *WF-net*, has been proposed by W.M.P van der Aalst [6]. Since in Ref. of [6] the delay time of transition has not been considered, here we extend WF-net as follows in order to do the quantitative evaluation.

Definition 1: A workflow net WF-net is a timed Petri net $N=(P,T,A,D)$ modeling workflow, which satisfies the followings.

(i) P is a set of places representing queue of activities, T is a set of transitions representing activities, A is a set of arcs representing flow relations and D is an n-vector, $D=(d_1, d_2, \ldots, d_{|D|})$ and d_i is delay time of transition t_i, which represents processing time of the corresponding activity.

(ii) A WF-net has one input (source) place p_I and one output (sink) place p_O. Every place and transition is located on a path from p_I to p_O.



(a) old workflow net

(b) new workflow net

Figure 1: An example of dynamic workflow change

(iii) The firing rules for WF-nets are defined as follows: If the firing of a transition t_i is decided, tokens required for the firing are reserved. We call these tokens as *reserved tokens* and denote them by a circle ($\circ$). When the delay time d_i of t_i passed, t_i fires to remove the reserved tokens from the input places of t_i and put *non-reserved tokens*, denoted by a filled circle ($\bullet$), into the output places of t_i.

(iv) The number of reserved and non-reserved tokens at place p are denoted as $M_r(p)$ and $M_{nr}(p)$ respectively, and the total token number is given by $M(p){=}M_r(p) + M_{nr}(p)$. □

In this paper, WF-nets are assumed to be marked graphs. Further, the arrival interval of the cases is supposed to be constant and not less than $\max\{d_i\}$. Figure 1(a) shows an example of WF-net for order processing. Note that transitions expressed by "|" are supposed to have 0 delay time in this example and hereafter.

2.2 Dynamic workflow changes

In terms of WF-net, a dynamic change is to replace a subnet N_{old} by a new one N_{new} in the original net $\mathcal{N}_{old}$, which results in a new net $\mathcal{N}_{new}$. Here, N_{old} and N_{new} are the old change region and the new change region, respectively, as shown in Fig. 1. In N_{new} and N_{old}, places, transitions, arcs, and delay times are generally different, but the input place p_I and the output place p_O are common. Note that N_{old} and N_{new} are WF-nets.

Dynamic change may encounter "dynamic bugs." For example, in the case of the change from N_{old} to N_{new} as shown in Fig. 1, if the tokens are simply fixed to the input place of the transition "Billing," then execution of the workflow will be obviously confused. Therefore, consistent dynamic changes are necessary. The three dynamic changes proposed by Ellis et al. can guarantee consistency and are as follows:

Flush change N_{old} is replaced by N_{new} later, after all tokens in N_{old} are transferred to p_O by firing. Tokens newly arriving at p_I are kept waiting until the replacement of N_{old} by N_{new} finishes.

Abort change N_{old} is immediately replaced by N_{new}. Meanwhile, all the tokens of N_{old} are put back to p_I in order to rehandle these tokens. After that all the tokens, including old and newly arriving ones, are handled in N_{new}. Note that the number of old tokens put back at p_I is equal to the number existing in a longest path of N_{old}.

SCO change When a dynamic change starts, N_{new} is immediately added to $\mathcal{N}_{old}$ by commonly sharing p_I and p_O without removing N_{old}. The tokens newly arriving at p_I can only go through N_{new}. N_{old} is removed after the tokens existing in N_{old} are all transferred to p_O. Further, at the output place p_O the tokens coming out from N_{new} are queued after those from N_{old}.

We suppose that resource numbers, before a dynamic change starts and after it finishes, are equal to that of transitions in N_{old} and N_{new}, respectively.

3. Computation and Evaluation

When a dynamic change is required to start from $\mathcal{N}_{old}$ to $\mathcal{N}_{new}$, there most probably exist tokens in the change region N_{old}. Of course, these tokens cannot be ignored and must be correctly handled according to either the N_{old} or N_{new}. In order to handle these tokens and further to keep the queuing order, the newly arriving tokens at p_I have to wait for some time before their arrival at p_O.

To evaluate whether a dynamic change is good or not, the waiting time of the newly arriving tokens is surely an important factor. We take this waiting time as a standard, called *change time*, to evaluate the performance of the three dynamic changes. Note that, since the times cost in removing N_{old} and adding N_{new} are the same in any one dynamic change, we assume these times to be zero to simplify our evaluation problem. Intuitively, change time is defined as the minimum total waiting time during the period when the first token, newly arriving at place p_I after a dynamic change starts, passes through N_{new}. Concretely, it is defined as follows:

Definition 2: Let τ_j^I be the time when the j-th token arrives at place p_I after the workflow initially started and τ_j^O be the minimum time when the token is transferred to p_O through N_{new}. Let $\Delta\tau_{N_{new}}$ be the minimum period for a token to pass through N_{new} from p_I to p_O without waiting time. Change time γ is given by

$$\gamma{=}(\tau_k^O - \tau_k^I) - \Delta\tau_{N_{new}},$$

where, the k-th token is supposed as the first token arriving at place p_I after dynamic change started. □

Hereafter, we are to show the methods how to compute change time for each dynamic change under the conditions: $\tau_{(j+1)}^I - \tau_j^I = d^*$ (as described in the last section) and $\tau_1^I = d^*$.

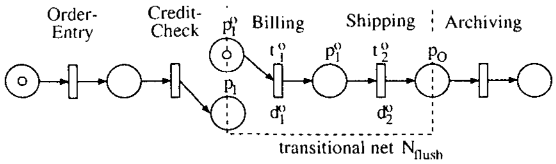


Figure 2: A transitional net N_{flush} for Fig. 1

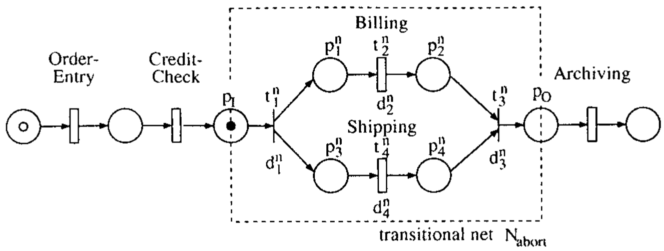


Figure 3: A transitional net N_{abort} for Fig. 1

Let T^o and T^n be the sets of transitions in N_{old} and N_{new}, and $\{d_i^o\}$ and $\{d_i^n\}$ be the firing delays of T^o and T^n respectively. The following properties are obvious.

Property 1: Let ρ^n be a longest path of N_{new} from p_I to p_O, with delay time d_i^n as its weight. Then

$$\Delta\tau_{N_{new}} = \sum_{t_i^n \in \rho^n} d_i^n. \qquad \square$$

Property 2: Change time γ can be obtained by the following equation:

$$\gamma = (\tau_k^O - kd^*) - \sum_{t_i^n \in \rho^n} d_i^n. \qquad \square$$

Therefore, to obtain change time of a dynamic change we need only to compute τ_k^O.

3.1 Change time γ_{flush} of flush change

In a flush change, the new arrival tokens have to wait at the input place p_I until all the tokens in N_{old} are transferred to the output place p_O. Figure 2 shows the transitional net N_{flush} of the case of Fig. 1.

According to Property 2, we need to compute τ_k^O. Obviously, $\tau_k^O = \tau_{(k-1)}^O + \Delta\tau_{N_{new}}$, where $\tau_{(k-1)}^O = (k-1)d^* + \sum_{t_i^o \in \rho^o} d_i^o$, that is the time when the last token in N_{old} is transferred to p_O. Note ρ^o is a longest path of N_{old}. Combining the result and Properties 1 and 2, we have following results.

Theorem 1: The change time γ_{flush} of flush change is

$$\gamma_{flush} = \sum_{t_i^o \in \rho^o} d_i^o - d^*. \qquad \square$$

3.2 Change time γ_{abort} of abort change

In an abort change, N_{old} is immediately replaced by N_{new} and, meanwhile, all the tokens in a longest path ρ^o of N_{old} are put back to p_I. Figure 3 shows the transitional net N_{abort} of the case of Fig. 1.

To compute the change time γ_{abort}, we need to compute the period of time for the $(\mu+1)$ tokens (including ones existing in N_{old} and the first token arriving

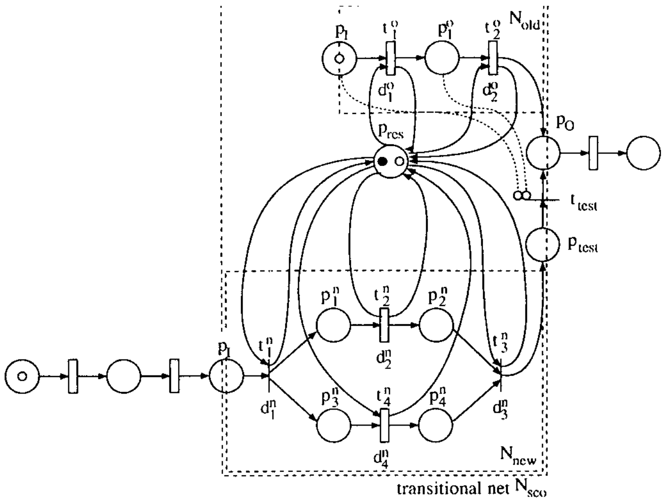


Figure 4: An transitional net N_{sco} for Fig. 1

at p_I after τ_{abort}) to pass through N_{new} from p_I to p_O. Firstly we give the following lemmas.

Lemma 1: Let N be a WF-net consisting of only a single path $\rho = p_I t_1 p_1 \cdots p_O$, in which there are no tokens at any other places except K tokens at p_I. The period of time $\Delta\tau_\rho^{(K)}$ for the K tokens to pass through ρ is

$$\Delta\tau_\rho^{(K)} = \sum_{t_j \in \rho} d_j + (K-1)\max_{t_j \in \rho}\{d_j\}. \qquad \square$$

Lemma 2: Let $\mathcal{P}^n = \{\rho_1^n, \rho_2^n, \ldots\}$ be the set of all the paths from p_I to p_O in N_{new} and K be the number of tokens existing inside of p_I. The period of time $\Delta\tau_{N_{new}}^{(K)}$ for the K tokens to pass through N_{new} from p_I to p_O is given by the following equation:

$$\Delta\tau_{N_{new}}^{(K)} = \max(\Delta\tau_{\rho_1^n}^{(K)}, \Delta\tau_{\rho_2^n}^{(K)}, \ldots, \Delta\tau_{\rho_{|\mathcal{P}^n|}^n}^{(K)}),$$

where $\Delta\tau_{\rho_i^n}^{(K)}$ $(i=1,2,\ldots)$ is computed by Lemma 1. $\square$

Now we are to give the change time τ_{abort}. The number of tokens put back to p_I can be calculated by $\mu = \sum_{t_i^o \in \rho^o} d_i^o / d^*$, where ρ^o is a longest path. If abort change is require to start at time $\tau_{abort} = kd^* - \delta$ $(0 \le \delta < d^*)$, then $\tau_k^O = kd^* - \delta + \Delta\tau_{N_{new}}^{(\mu+1)}$. Therefore from Property 2 and Lemma 2, we can get τ_{abort} by the following theorem.

Theorem 2: The change time γ_{abort} is given by

$$\gamma_{abort} = \Delta\tau_{N_{new}}^{(\mu+1)} - \sum_{t_i^n \in \rho^n} d_i^n - \delta,$$

where, $\mu = \sum_{t_i^o \in \rho^o} d_i^o / d^*$, $\Delta\tau_{N_{new}}^{(\mu+1)}$ is computed by Lemma 2 and δ is such that abort change is required to start at time $kd^* - \delta$ $(0 \le \delta < d^*)$. $\square$

3.3 Change time γ_{sco} of SCO change

In an SCO change, tokens of N_{old} and N_{new} are handled concurrently. However, the tokens of N_{new} must be transfered to p_O after all the tokens of N_{old} have arrived at p_O. This is because all tokens must be handled in the order of FIFO. To judge whether there exists no token in N_{old}, we need a transition t_{test} and inhibit arcs.

Besides being different from the last two dynamic changes, SCO change requies that the number of resources be determined, because during the transitional period, N_{old} and N_{new} exist simultaneously. In this paper, we assume that the number of resources is the larger of the number of resources of either old workflow or new workflow, simply because during the change the resources must take charge of both old and new workflows. Obviously in this case, resources are insufficient and thus resource conflict will occur. To solve this resource conflict problem, we adopt the following firing policies:

(i) The transitions of N_{old} have higher priority than those of N_{new} in firing;

(ii) For the transitions of N_{old} or N_{new}, ones with longer delay time have higher priority, which has been considered as effective scheduling policy [7].

Figure 4 shows the transitional net N_{sco} of the case of Fig. 1.

To compute γ_{SCO}, we need to obtain the minimum τ_k^O. However, such minimum τ_k^O is difficult to be obtained due to insufficient resources. Therefore in the following, we are to propose an algorithm to give its approximate value in order to give its upper bound. The overview of our algorithm is as follows:

1° Use L as a list of transitions that are sorted in descending order of delay time and T_F as a list of transitions that have be decided to fire.

2° For each time epoch τ, check if there are tokens at p_{res}. If p_{res} has tokens, select firable transitions from L to add into T_F.

3° For the transition $\forall t \in T_F$, fire t if its delay time has passed.

4° If there are no tokens in N_{old} and p_{test} has tokens, stop. Otherwise, τ is incremented and goto 2°.

The time τ obtained by the above algorithm is the approximate value of τ_k^O, and thus its upper bound of the change time is

$$\gamma = (\tau - kd^*) - \sum_{t_i^n \in \rho^n} d_i^n.$$

To make it convinient for our later discussions, we simply denote γ as γ_{sco}.

3.4 Evaluation

We applied the above proposed methods to compute the three change times for the example shown in Fig. 1. We assumed d_1^o, d_2^o, d_2^n, and d_4^n be 10, 8, 8, and 4, respectively. These change times, varying with the arrival interval of input cases, are shown in Fig. 5. From Fig. 5, we find that an SCO change is the best and costs the shortest change time for any arrival interval, and that the change times of flush and abort are dependent on the arrival intervals.

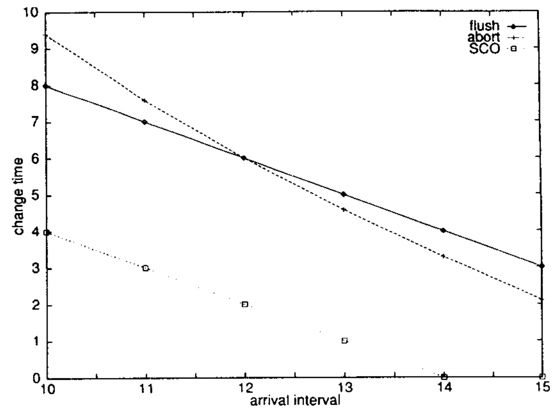


Figure 5: Results of computing change times for Fig. 1

4. Concluding Remarks

In this paper, based on workflow net, we have first introduced the definition of change time and then proposed the computation methods individually for each change type. Finally, we have done the performance evaluation of an example net change by computing the change times. Our experimental results show that:

(i) SCO change is the best dynamic change for any arrival interval of input cases;

(ii) Abort change is next to SCO change except for the cases of short arrival interval of input cases.

Since immediately after workflow change finished, the workflow has not reached a steady state, i.e., arrival intervals of the tokens at p_O are not the same as at p_I, in future work related to the performance evaluation of the three dynamic changes, we need to further investigate which dynamic change can reach a steady state earliest.

References

[1] Workflow Management Coalition, "Terminology & Glossary," Document No. WFMC-TC-1011, 1999.

[2] P. Koksal, I. Cingil, and A. Dogac, "A Component-Based Workflow System with Dynamic Modifications, " Lecture Notes in Computer Science, Issue 1649, pp. 238–255, 1999.

[3] C. Ellis, K. Keddara, and G. Rozenberg, "Dynamic change within workflow systems," Proc. ACM Conference on Organizational Computing Systems '95, pp.10–21, 1995.

[4] T. Herrmann, "Workflow management systems: ensuring organizational flexibility by possibilities of adaption and negotiation," Proc. ACM Conference on Organizational Computing Systems '95, pp.83–94, 1995.5.

[5] T. Jaeger, and A. Prakash, "Management and utilization of knowledge for the automatic improvement of workflow performance," Proc. ACM Conference on Organizational Computing Systems '95, pp.32–43, 1995.

[6] W.M.P. van der Aalst, "The application of petri nets to workflow management," J. Circuits, Systems, & Computers, vol.8, no.1, pp.21–65, 1998.

[7] Q. W. Ge, "PARAdeg-processor scheduling for acyclic switch-less program nets," J. Franklin Institute, vol.336 no.7, pp.1135–1153, 1999.