Computation of Change Time for Migrate Dynamic Workflow Changes

Shingo Yamaguchi[1], Akira Mishima[1], Qi-Wei Ge[2] and Minoru Tanaka[1]

[1]Faculty of Engineering, Yamaguchi University
2-16-1 Tokiwadai, Ube, 755-8611, Japan
Tel: +81-836-85-9511, Fax: +81-836-85-9501
[2]Faculty of Education, Yamaguchi University
1677-1 Yoshida, Yamaguchi, 753-8513, Japan
Tel: +81-83-933-5401, Fax: +81-83-933-5304
E-mail: {shingo, mishima, tanaka}@cs.csse.yamaguchi-u.ac.jp, gqw@inf.edu.yamaguchi-u.ac.jp

Abstract: A workflow is a flow of work supported by computers. An instance of a workflow is called *case*. Companies need to constantly refine their current workflows in order to meet various requirements. The change of current workflows is called *dynamic change* of the workflow. Since Ellis et al. proposed three change types, *Flush*, *Abort*, and *Synthetic Cut-Over* in 1995, various change types have been proposed. A promising change type is *Migrate* proposed by Sadiq et al., because Migrate changes workflow definitions immediately and makes the redo of cases minimum. However, the formal modeling and time-dependent analysis of Migrate has not been done. This paper proposes a method of computing *change time* of Migrate dynamic changes for time-dependent analysis. Change time is a measure for evaluating dynamic changes. We first show a Petri-nets-based model of Migrate dynamic changes. Then we present a method of computing change time based on the net model. Finally, we apply the method to 270 examples, and show experimental results, and comparison with Ellis et al.'s three change types.

1. Introduction

As core technology of next generation information systems, workflow management has been attracting a great deal of attention. A workflow expresses a flow of work, and workflow management is to automate the flow of work. An instance of a workflow is called *case*.

Companies need to constantly refine their workflows in order to adapt to new technology, market's demands, and new laws [1]. The change of current workflows is called *dynamic change* of the workflow. When a dynamic change of a workflow is required, there may exist cases within the region to be changed. If those cases are ignored or become deadlock, the changed workflow will become inconsistent. Since Ellis et al. [2] proposed three change types, *Flush*, *Abort*, and *Synthetic Cut-Over* (SCO) that keep consistency of workflows in 1995, various change types have been proposed. A promising change type is *Migrate* proposed by Sadiq [3]. Migrate can change workflow definitions immediately and make the redo of cases minimum.

We have proposed a measure, called *change time* [4], for evaluating dynamic change of workflows. Change time is the total waiting time resulting while the first newly-arrived case is processed through the changed region, after dynamic change is required. In [4], we have carried out the performance evaluation on change time of Ellis et al.'s three change types. The results show that (i) SCO is the best; (ii) Abort is better than Flush unless the arrival interval of input cases is short. However, the performance evaluation of Migrate has not been done; moreover even in [3] the formal modeling of Migrate has not been done.

In this paper, we propose a method of computing change time of Migrate dynamic changes. We first show a Petri-nets-based model of Migrate dynamic changes. Then we present a method of computing change time based on the net model. Finally, we apply the method to 270 examples, and show experimental results, and comparison with Ellis et al.'s three change types.

2. Preliminary

2.1 Workflows and WF-nets

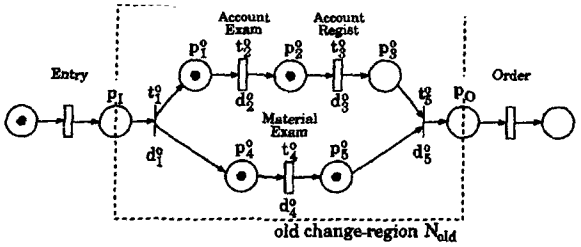
A workflow is composed of many *activities*. Each activity forms one logical step within the workflow. One workflow has many cases. In general, cases are processed in order of *First-In First-Out* (FIFO).

A workflow can be modeled by a Petri net, called *workflow net* (WF-net) [5]. We have extended WF-net by introducing time in order to do time-dependent analysis. The extended WF-net is defined as follows:

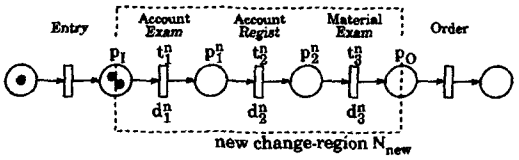
Definition 1. A timed Petri net $TPN=(P,T,A,D)$[§] is a WF-net iff (i) TPN has one input place p_I ($^\bullet p_I=\phi$) and one output place p_O ($p_O{}^\bullet=\phi$); (ii) every place and transition is located on a path from p_I to p_O. □

In WF-nets, activities, cases, and the processing time of activities are represented as transitions, tokens, and the delay-time of transitions, respectively. This paper assumes that WF-nets are acyclic marked graphs. Further, the arrival interval d^* of input cases is supposed to be constant and not less than $\max_{t_i \in T}\{d_i\}$. Figure 1(a) shows an example of WF-nets. Transitions with no label, denoted by bars (|), are for splitting or converging of flows and have no delay-time.

[§] P and T are disjoint sets of *places* and *transitions*, $A \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs*, and D is a set of the *delay-time* of transitons.



(a) old WF-net



(b) new WF-net

Figure 1. An example of dynamic changes (by Abort).

2.2 Dynamic Change of Workflows

In terms of WF-nets, a dynamic change is to replace a subnet $N_{old}=(P^o, T^o, A^o, D^o)$ by a new subnet $N_{new}=(P^n, T^n, A^n, D^n)$ in the original net $\mathcal{N}_{old}$, which results in a new net $\mathcal{N}_{new}$, as shown in Fig. 1. N_{old} is called *old change-region*, N_{new} is called *new change-region*. In N_{new} and N_{old}, places, transitions, arcs, and delay-times are generally different, but p_I and p_O are common. The change types treated in this paper are as follows:

- Flush: N_{old} is later replaced by N_{new}, after all tokens on N_{old} are outputted to p_O.
- Abort: N_{old} is immediately replaced by N_{new}. All tokens on N_{old} are put back to p_I in order to redo the tokens.
- SCO: N_{new} is immediately added to $\mathcal{N}_{old}$ without removing N_{old}. Tokens newly arriving at p_I can only go through N_{new}. N_{old} is removed after all tokens on N_{old} are outputted to p_O.
- Migrate: N_{old} is immediately replaced by N_{new}. Every token on N_{old} is moved to suitable places of N_{new} so that the redo of the token is made minimum.

2.3 Change Time γ

Intuitively, change time is the total waiting time resulting while the token newly arriving at p_I is outputted to p_O after a dynamic change is required. In order to simplify our performance evaluation, we assume that time needed for deletion of N_{old} and addition of N_{new} is 0, because the time is the same to any dynamic changes. The definition of change time is given as follows:

Definition 2. Let τ_{p_I}, τ_{p_O}, and $\Delta\tau_{N_{new}}$ be time when the first token is inputted to p_I after a dynamic change is required, time when the token is outputted to p_O, and the minimum period for a token to move from p_I to p_O on N_{new} with no waiting-time, respectively. Change time γ is given by $\gamma=(\tau_{p_O}-\tau_{p_I})-\Delta\tau_{N_{new}}$. □

3. Modeling of Migrate Dynamic Changes

One problem about Migrate is to decide where each token on N_{old} is migrated to so that redo is made minimum. To solve this problem, we propose the formal definition of Migrate with the procedure for mapping marking M^o of N_{old} to marking M^n of N_{new}.

Definition 3. A dynamic change is called *Migrate* iff N_{old} is immediately replaced by N_{new} and every token on N_{old} is migrated according to the following procedure.

(1) Division of marking M^o by case:
Let m be the number of cases on N_{old}, we divide M^o into m markings $M_1^o, M_2^o, \cdots M_m^o$ by case.

(2) Migration of every divided marking M_i^o to M_i^n:
We map every divided marking M_i^o to M_i^n according to a rule, called *migration rule*. Details of migration rule are given later.

(3) Combination of all migrated markings M_i^n:
Marking M^n is obtained by combining all migrated marking M_i^n, i.e. $M^n(p)\leftarrow\sum_{i=1}^{m} M_i^n(p)$. □

In step (1) of the procedure, we divide M^o into m markings $M_1^o, M_2^o, \cdots M_m^o$ by case. Let ρ^o be a path from p_I to p_O, m can be calculated as $m=\sum_{p\in\rho^o} M^o(p)$. The following is the outline of an algorithm for dividing M^o into $M_1^o, M_2^o, \cdots M_m^o$.

1° $i\leftarrow 1$, $m\leftarrow\sum_{p\in\rho^o} M^o(p)$, $P_T^o\leftarrow\{p_I\}$.

2° Repeat the followings until $i>m$.
If $\forall p\in P_T^o, M^o(p)>0$ then

$$M_i^o(p)=\begin{cases}1 & \text{if } p\in P_T^o\\ 0 & \text{otherwise}\end{cases},\quad i\leftarrow i+1.$$

else
$Q\leftarrow\{p\in P_T^o \mid M^o(p)=0\}$, $P_T^o\leftarrow P_T^o-Q+\{(p^\bullet)^\bullet \mid p\in Q\}$.

3° Stop, output $M_1^o, M_2^o, \cdots M_m^o$.

In step (2), we use the following migration rule.

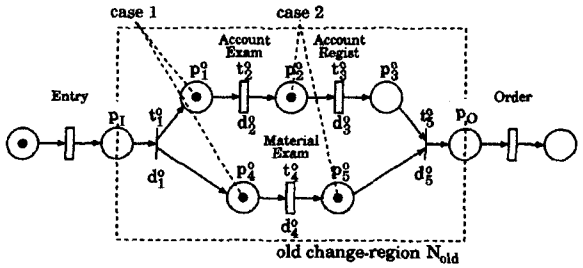
Definition 4. Let M_0^o and M_0^n be markings such that (i) if $p=p_I$, $M_0^o(p)=M_0^n(p)=1$; (ii) otherwise, $M_0^o(p)=M_0^n(p)=0$. A mapping $R:M_i^o\mapsto M_i^n$ is called *migration rule* iff
(i) $\forall t\in\{t_j^n \mid M_0^n[t_j^n\rangle\cdots M_i^n\}$, $L^n(t)\in\{L^o(t_k^o) \mid M_0^o[t_k^o\rangle\cdots M_i^o\}$,
(ii) $\forall t\in\{t_j^n \mid M_i^n[t_j^n\rangle\}$, $L^n(t)\notin\{L^o(t_k^o) \mid M_0^o[t_k^o\rangle\cdots M_i^o\}$,
where $L^o:T^o\rightarrow\Sigma$ and $L^n:T^n\rightarrow\Sigma$ are given labeling functions of transitions, $M[t\rangle$ means that t is firable in M, $M[t\rangle M'$ means that M changes to M' by firing t. □

The following is the outline of an algorithm for obtaining M_i^n from M_i^o according to the migration rule.

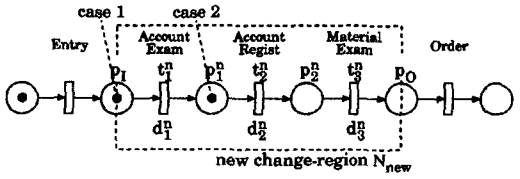
1° $M_i^n\leftarrow M_0^n$, $T_\mathcal{F}^o\leftarrow\{t_k^o \mid M_0^o[t_k^o\rangle\cdots M_i^o\}$.

2° Repeat the followings until $T_\mathcal{F}^o=\phi$.
Let T_F^n be the set of firable transitions in M_i^n. If $\exists t\in T_F^n, L^n(t)\in\{L^o(t_k^o) \mid t_k^o\in T_\mathcal{F}^o\}$, update M_i^n by firing t and $T_\mathcal{F}^o\leftarrow T_\mathcal{F}^o-\{t\}$. Otherwise, goto 3°.

3° Stop, output M_i^n.



(a) old WF-net



(b) new WF-net

Figure 2. An example of dynamic changes by Migrate.

Figure 2 shows an example of dynamic workflow changes by Migrate. In step (1), M^o=(0, 1, 1, 0, 1, 1, 0) is divided into two markings M_1^o=(0, 1, 0, 0, 1, 0, 0) and M_2^o=(0, 0, 1, 0, 0, 1, 0). In step (2), M_1^o=(0, 1, 0, 0, 1, 0, 0) is migrated to M_1^n=(1, 0, 0, 0) (=M_0^n), because $\{L^o(t_k^o)|t_k^o \in T_{\mathcal{F}}^o(=\{t_1^o\})\}$ =ϕ. On the other hand, M_2^o=(0, 0, 1, 0, 0, 1, 0) is migrated to M_2^n=(0, 1, 0, 0), because $L^n(t_1^n)$="*AccountExam* "$\in\{L^o(t_k^o)|t_k^o \in T_{\mathcal{F}}^o$ (=$\{t_1^o, t_2^o, t_4^o\}$) }={"*AccountExam*", "*MaterialExam*"} and thus the case doesn't redo activity *AccountExam*. In step (3), the migrated markings M_1^n and M_2^n are combined into M^n, i.e. M^n=$\sum_{i=1}^{2} M_i^n$=(1, 1, 0, 0). Thus M^o=(0, 1, 1, 0, 1, 1, 0) is mapped into M^n=(1, 1, 0, 0).

4. Computation of Change Time $\gamma_{migrate}$

To compute $\gamma_{migrate}$, we need to obtain τ_{p_O}. However it is difficult to obtain τ_{p_O} analytically, because τ_{p_O} fluctuates according to migrated markings. Therefore we propose a method for computing $\gamma_{migrate}$ based on an algorithm for obtaining τ_{p_O}. The following is the outline of the algorithm.

1° Use τ as a variable representing a current time.

2° $\tau \leftarrow \tau_{cs}$ (start time of dynamic change).

3° Repeat the followings until the first token newly arriving at p_I after τ_{cs} is outputted to p_O.

- If the first token arrives at p_I, $\tau_{p_I} \leftarrow \tau$.
- Update M^n by firing transitions firable at τ.
- $\tau \leftarrow \tau+1$.

4° Stop, output τ as τ_{p_O}.

$\gamma_{migrate}$ is obtained by $(\tau_{p_O}-\tau_{p_I})-\Delta\tau_{N_{new}}$, where $\Delta\tau_{N_{new}}$ =$\sum_{t_i^n \in \rho^n} d_i^n$ [4], ρ^n is the longest path from p_I to p_O with delay-time as its transitions' weight.

Table 1. Computation results of $\gamma_{migrate}$ when applying a change-operation once and $|T^o|$=50.

Op.	10	11	12	13	14	15	16	17	18	19	20	Ave.
1	79.5	63.8	50.8	42.7	36.7	31.4	26.8	22.7	19.0	15.7	12.7	36.5
2	95.2	79.5	66.3	55.0	45.4	38.5	33.8	29.7	26.0	22.7	20.2	46.6
3	11.6	3.6	1.6	0.8	0.4	0.2	0.1	0.1	0.0	0.0	0.0	1.7
4	48.4	33.3	25.2	20.9	17.5	14.6	12.0	9.8	7.8	5.9	4.3	18.2
5	84.2	70.7	59.4	49.7	41.4	34.1	27.7	21.9	16.8	12.1	8.4	38.8
6	8.1	3.4	1.6	0.8	0.4	0.2	0.1	0.1	0.0	0.0	0.0	1.3
7	17.6	9.0	4.4	2.6	1.3	0.5	0.3	0.1	0.0	0.0	0.0	3.3
8	7.7	3.4	1.6	0.8	0.4	0.2	0.1	0.1	0.0	0.0	0.0	1.3
9	7.9	3.4	1.6	0.8	0.4	0.2	0.1	0.1	0.0	0.0	0.0	1.3
10	9.9	5.2	3.0	1.5	0.8	0.5	0.2	0.1	0.0	0.0	0.0	1.9
Ave	37.0	27.5	21.6	17.6	14.5	12.0	10.1	8.5	7.0	5.6	4.6	15.1

5. Evaluation

We applied our method to the performance evaluation for Migrate and Ellis et al.'s three change types.

5.1 Experiments

In our experiment, we used 270 examples of dynamic changes that the following *change-operations* were applied to.

(1) Sequential insertion of activities.

(2) Parallel insertion of activities.

(3) Deletion of activities.

(4) Replacement of activities.

(5) Sequentiall exchange of activities.

(6) Parallel exchange of activities.

(7) Parallelization of activities.

(8) Serialization of activities.

(9) Addition of order-relation between activities.

(10) Deletion of order-relation between activities.

In WF-nets, *order-relation* between activities is represented as a path between transitions.

Concretely, old WF-nets and new WF-nets were generated as follows. We first generated 9 WF-nets $N_{old_1}, \cdots, N_{old_9}$ as old WF-nets. $N_{old_1}, N_{old_2}, N_{old_3}$ have 10 transitions ($|T^o|$=10), $N_{old_4}, N_{old_5}, N_{old_6}$ have 30 transitions ($|T^o|$=30), $N_{old_7}, N_{old_8}, N_{old_9}$ have 50 transitions ($|T^o|$=50).

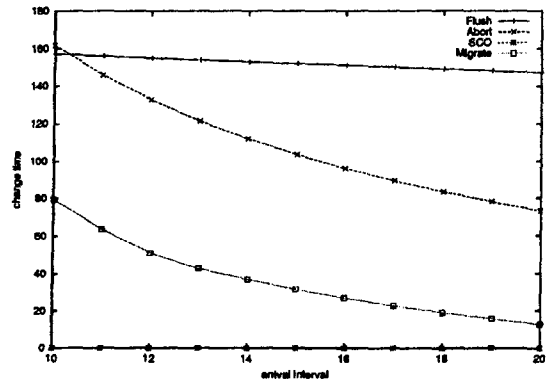
Then we generated 270 (=10×9×3) WF-nets $N_{new_1}, \cdots, N_{new_{270}}$ as new WF-nets by doing the followings: (i) first to generate 10 WF-nets for each of the 9 old WF-nets individually; (ii) then for each net generated in (i), to further apply the following 3 conditions to generate 3 nets.

(i) Applying a change-operation once;

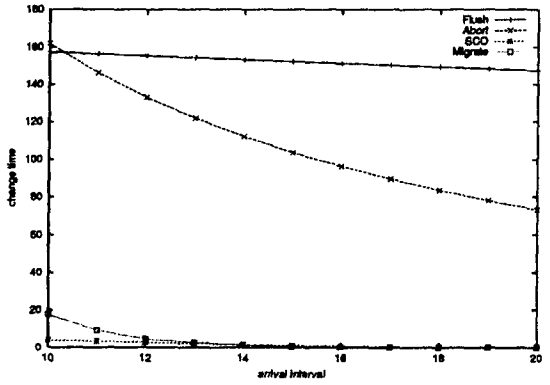
(ii) Applying a random change-operation twice;

(iii) Applying a random change-operation three to five times.

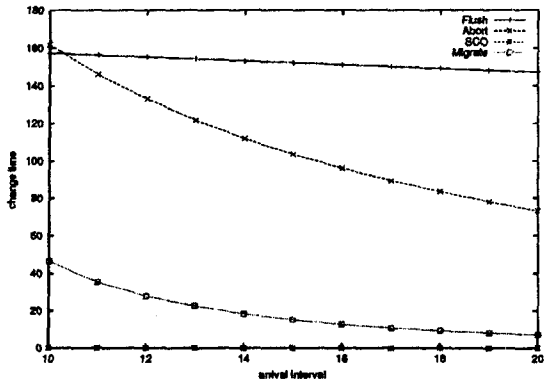
We increased arrival interval d^* of input tokens from 10 to 20, and computed $\gamma_{migrate}$ for each d^*.



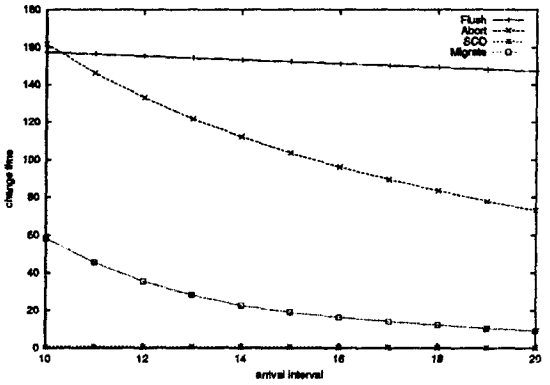
(a) applying *Sequential insertion of activities* once.



(b) applying *Parallerization of activities* once.



(c) applying a random change-operation twice.



(d) applying a random change-operation three to five times.

Figure 3. Comparison on change time of Migrate and Ellis's three change types ($|T^o|$=50).

5.2 Experimental Results and Discussion

Table 1 shows computation results of $\gamma_{migrate}$ ($|T^o|$=50) when applying a change-operation once. We see from Table 1 that $\gamma_{migrate}$ decreases with the increase of d^* nonlinearly. This may be because $\gamma_{migrate}$ depends on the number m of cases on N_{old} and furthermore m depends on $1/d^*$.

Further, we compared Migrate with Ellis et al's three change types. Figure 3 shows the comparison results on change time of the four change types. Figure 3(a) and (b) are the results when applying change-operation *Sequential insertion of activities* and *Parallerization of activities* once, respectively. We see from Fig. 3(a) and (b) that Migrate depends on type of change-operations more strongly than other change types. In fact, Migrate is better than SCO if a change-operation is *Parallelization of activities* or *Deletion of order-relation between activities.* Besides, Fig. 3(c) and (d) are the results when applying a random change operation twice or more. We see from Fig. 3(c) and (d) that (i) Migrate is better than Flush and Abort; (ii) SCO is better than Migrate.

6. Concluding Remarks

In this paper, we first showed a Petri-nets-based model of Migrate dynamic changes. Then we showed a method of computing change time based on the net model. Finally, we applied our method to the performance evaluation for Migrate and Ellis et al.'s three change types. Our experimental results show that

(i) Migrate is better than Flush and Abort;

(ii) SCO is better than Migrate unless change-operations include *Parallelization of activities* or *Deletion of order-relation between activities.*

As the future work, we need to further investigate comparison of Migrate and SCO analytically.

References

[1] P. Koksal, "A component-based workflow system with dynamic modification," Proc. Next Generation Information, pp.238–255, 1999.

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

[3] S.W. Sadiq, "Managing change and time in dynamic workflow processes," International journal of cooperative, vol.9, no.1&2, pp.93–116, 2000.

[4] S. Yamaguchi, Q.W. Ge, M. Tanaka, "Performance evaluation on change time of dynamic workflow changes," IEICE Trans., vol.E83-A, no.11, pp.2177–2187, 2000.

[5] W.M.P. van der Aalst: "The application of Petri nets to workflow management," Journal of Circuit, Systems, and Computers, vol.8, no.1, pp.21–65, 1998.