Jan. 2006, Vol.21, No.1, pp.66–71 J. Comput. Sci. & Technol.

A Workflow Process Mining Algorithm Based on Synchro-Net

Xing-Qi Huang (黄星琪), Li-Fu Wang (王立福), Wen Zhao (赵 文), Shi-Kun Zhang (张世琨), and Chong-Yi Yuan (袁崇义)

School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, P.R. China

E-mail: {huangxq, wlf, owen, zsk}@cs.pku.edu.cn

Received April 25, 2005; revised October 24, 2005.

Abstract Sometimes historic information about workflow execution is needed to analyze business processes. Process mining aims at extracting information from event logs for capturing a business process in execution. In this paper a process mining algorithm is proposed based on Synchro-Net which is a synchronization-based model of workflow logic and workflow semantics. With this mining algorithm based on the model, problems such as invisible tasks and short-loops can be dealt with at ease. A process mining example is presented to illustrate the algorithm, and the evaluation is also given.

Keywords workflow, process mining, workflow logic, workflow semantics, Petri net

1 Introduction

A workflow is a partial or total automation of a business process in which activities are executed by human or machines according to certain predefined rules[1].

Modern enterprises increasingly use workflow technology to design business processes, by means of management systems that provide mechanisms for formally specifying the schema of execution, for simulating its evolution under different conditions, for validating and testing whether it behaves as expected, and for evaluating the ability of a service to meet requirements with respect to throughput times, service levels, and resource utilization.

In an ideal situation, a well-defined business process should be designed before its enactment is possible and redesigned whenever changes take place. Sometimes, we may need process mining to get the information about workflow execution and find out how people and/or a procedure really work. The process mining can also be applied to delta analysis, which means comparing actual processes with predefined ones. Therefore, process mining is of great importance.

In general, a workflow process model is the static design of a business process, while the workflow execution is dynamic and decides which activities take place at runtime. The definition of workflow process mining is given in [1]: Given a workflow process log that records the orders in which activities take place, we try to find a workflow process model with some constrains, so that all traces correspond to the instance of the business process model. In fact, engine log is stored in database or a plat file and what we need to do is to use data mining technique to rediscover the process model from the log.

Considerable work has been done in this field. Aalst uses a special kind of Petri net named WF-net to model the control flow of a process[2]. Based on WF-net, Aalst applies the α-algorithm to mining the process model[3,4]. And later, some extended algorithms are proposed[5−8]. Some other people also do some researches in this field, Herbst and Karagiannis, as well as Greco, Guzzo, Manco and Pontieri[1,9,10], to name but a few. However, there still remain some problems to be adequately resolved, such as invisible tasks, none free choices, one-length loops, two-length loops and so on[3].

In this paper, we propose our process mining method which is based on a synchronization-based model of workflow logic and workflow semantics, named Synchro-Net[11], and present a mining algorithm to rediscover the process model from workflow engine's log. With the workflow model and our process mining method, problems such as invisible tasks and one-length loops can be dealt with at ease.

The rest of this paper is organized as follows. Section 2 introduces a synchronization-based model of workflow logic and workflow semantics. In Section 3, a process mining method based on the model is given and a concrete algorithm for constructing process model is proposed. To illustrate the mining algorithm, an example is given in Section 4, and the evaluation of the mining method is also outlined. Finally, the conclusion and future work are drawn in Section 5.

2 A Synchronization-Based Model of Workflow Logic and Workflow Semantics

Firstly, let us take a general look at the Synchro-Net, a synchronization-based model.

As mentioned in [11], a workflow is the formal description of a business process, including workflow logic which describes dependences (causal dependences and/or organizational regulations) between tasks and workflow semantics added on the logic to describe obvious contents. To check finished tasks according to the workflow logic and to start the next task or select and start the next task according to the workflow semantics involve workflow management. Both workflow logic

Short paper

Supported by the National Natural Science Foundation of China under Grant No. 60473058.

and semantics make up a process model, represented by Synchro-Net.

In [11], the details of workflow logic and semantics are explained rationally, and a formal model has been presented. Next, let us explain some most important points of the model and the formal definitions, so as to make readers understand the workflow mining algorithm based on the model described later.

2.1 Workflow Logic and Workflow Semantics

Workflow logic plays a decisive role in workflow. And in fact, synchronization between tasks is the central concern of workflow logic.

In general, workflow logic specifies how tasks of a business process are ordered and disordered, i.e., how they are synchronized. The order is derived from the casual dependences among tasks and from organizational regulations. Besides, it covers all possible routes for all possible cases allowed by the business in question, i.e., it is case irrelevant. And workflow logic is not concerned with the actual contents of a task execution. Furthermore, a task can be executed at most once for each run of the logic. So, iterative routing should not appear to be a logic feature. Thus, it is not concerned with case attributes needed to make decisions on selective routings. Such attributes will be introduced into the logic to form workflow semantics. The passing of control from task to task is to be done automatically by workflow engine since control-passing involves resource assignment to tasks. As such, control-passing is a matter of implementation which a task should not be involved in. The duty of a task is confined to the business itself, not including business management.

With the understanding of workflow logic, workflow semantics may be described as follows.

Workflow semantics is case-relevant, and it defines a unique route for each case based on case attributes. It involves only those attributes that are needed to make decisions on selective routing, while not concerned with actual contents of tasks beyond decision-making attributes. Workflow semantics does not deal with management affairs like resource assignments, time constraints and safety considerations. Performance analysis is also beyond its scope. There is no need for returns (redo) and skips to be introduced into a workflow semantics model, for the duty of workflow semantics is only to solve the conflict in the workflow logic and not concerned with the quality of the task that has been done. Whether to redo or not is the concern of workflow management, and the actual return to redoing some tasks is judged by the workflow engine according to the decisions by certain participants or management rules. Such postponed ship and returns are called implicit jumps: jump forward and jump backward, and are dealt with by workflow engine at runtime.

2.2 Formal Descriptions

Now, we introduce the formal descriptions of the synchronization-base model of workflow logic and semantics.

Order Relations Among Tasks. All tasks in $TASK$ are ordered by the nature of the related business process. So we have a relation $<$, $< \subseteq TASK \times TASK$.

And we have a sub-relation $\lessdot$ of $<$: $\lessdot \subseteq <$ and $(t,t') \in \lessdot \Rightarrow \forall t'' \in TASK$: $\neg(t < t'' \wedge t'' < t')$.

$\lessdot$ is the next relation among tasks. For $(T_1,T_2) \in \lessdot$ we say that T_1 is immediately before T_2, or T_2 is immediately after T_1. We will define workflow logic by specifying how T_1, T_2, and $T_1 \lessdot T_2$, are synchronized.

Synchronizer. The synchronizer is the central concern of workflow logic. Place p with a local structure as shown in Fig.1 is called a synchronizer of pattern (a_1,a_2) between T_1 and T_2, or simply a synchronizer. It is called a selective synchronizer if $a_1 < |T_1| \vee a_2 < |T_2|$, otherwise it is a decisive synchronizer. We write $p = (T_1,T_2,(a_1,a_2))$.

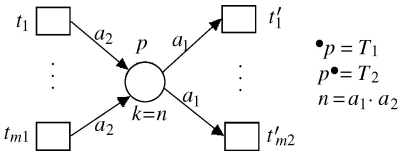

Fig.1. Synchronizer.

In workflow logic, each transition can occur at most once, and a synchronizer $p = (T_1,T_2,(a_1,a_2))$ can authorize the post transitions to happen only if $M(p) = a_1 \times a_2$.

Workflow Logic. $\Sigma = (P,T;F,K,W,M_0)$ is called the workflow logic of $(TASK,<)$, or WF-logic for short, if $\lessdot = \{(t,t') | t,\ t' \in T \wedge \exists p \in P:\ t \in {}^\bullet p \wedge t' \in p^\bullet\}$, $\forall p \in P$: $({}^\bullet p = \emptyset \Rightarrow M_0(p) = 1) \wedge ({}^\bullet p \neq \emptyset \Rightarrow M_0(p) = 0)$, and $(T,\lessdot) = (TASK,\lessdot')$.

Workflow Semantics. A C_net[11] system $\Sigma = (P, V, T; F, K, W, R, W_r, W_T, M_0)$ is called a workflow semantics frame, ws_frame for short, if $(P,T;F,K,W,M_{op})$ is a WF-logic, where $M_{op} = M_0|p$, and $\forall v \in V$: $|w(v)| \leqslant 1 \wedge |r(v)| > 1 \wedge M_0(v) = 0$, $\forall t \in T$: $M_T = guard(t) + body(t)$.

Fig.2 is an example of the workflow logic and semantics in applying for leave process.

In workflow logic and semantics model: P is the set of places; V is the obvious variables of the *B_form*[11], and variables in V are used in guard and body of transitions; T is the set of transitions and each transition in T holds for a task of the business process; F is the flow relation between places and transitions, while in the net it is the arc; K is the set of capacities of places in P; W is the set of weight on each arc of F; R is the reading relation that denotes which transition in T reads which variables in V; W_r is the writing relation that denotes which transition in T writes which variables in V; M_T is the set of guard and body of each transition in T; and M_0 is the initial marking of each place in P.

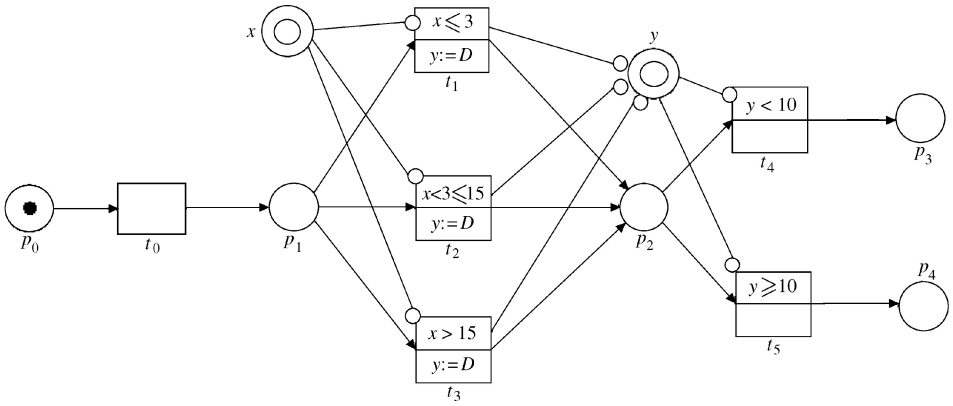

Fig.2. Example of workflow logic and semantics.

It should be noticed that both of the workflow logic and workflow semantics together make up the workflow process model. In this sight, we do mining and try to rediscover a workflow logic and semantics model like above.

Readers can refer to [11] for more details about the synchronization-based model of workflow logic and workflow semantics.

3 Process Mining

The log database stores the execution traces of workflow processes, such as when a transition starts and ends, as well as each transition's status, for example, the variable's value in guard and body of a transition. What we need to do is to mine this information so as to rediscover the workflow process model.

3.1 Some Assumptions Before Mining the Model

Before presenting how to do mining, we need to put forward some assumptions.

Firstly, in a more theoretical approach, we do not focus on issues such as noise. We assume that there is no noise and the log contains sufficient information.

Since our work is based on the synchronization-based model of workflow process, in which synchronizer is the central concern of the Petri net, our mining result is a Petri net strictly following the workflow logic and semantics model, in which loop or jump does not occur, for the latter two things are the concerns of the workflow management, and the actual return to redoing some tasks is judged by the workflow engine according to the decisions by certain participants or management rules.

There is another assumption that the original process model is well defined. Before the workflow process is actually executed by workflow engine, usually it has to be verified. That phase is called process model verification. In process model verification, the model is judged according to the rules in a rule base so as to make sure whether it is well-constructed. The synchronizers in our model make it clear that there neither exists such a structure that the choice and synchronization are mixed nor is synchronization without all its preceding transitions. This makes our net structure meet the definition of Aaslt's SWF-net[3]. In [11], the author presents some verification rules of the model. In a word, we assume the original process model is well-constructed before we try to rediscover the model.

3.2 Our Mining Method

For our process mining, we need to rebuild a Petri net $(P, V, T; F, K, W, R, W_r, M_T, M_0)$ from the log of workflow engine. T is recorded in the log database. Then, we can easily get M_T, which is corresponding to T. Obviously, we can come to the conclusion that the set of variables appear in guards and bodies of transitions in T, and that is V. Now we have T, M_T and V, and R and W_r can be concluded. M_0 can be easily built because only the start place contains a token. What is left for us is to mine P, F, K and W from the information above.

Our process mining algorithm receives an event log from the log database as input and returns to an extended Petri net named Synchro-Net[11] as output. It can be seen as an extension of α-algorithm.

Let T be a set of tasks. Let $\sigma = t_1 t_2 \dots t_n \in T^*$ a sequence over T of length n. $\in$, first, and last are defined as follows:

1) $t \in \sigma$ if and only if $t \in \{t_1, t_2, \dots t_n\}$;

2) if $n \geqslant 1$, then $\text{first}(\sigma) = t_1$ and $\text{last}(\sigma) = t_n$.

And next we define order relations between transitions in the log. There are four types of order relations: next, parallel, choice, and only possibility of next. Let L be a log over T, and $t_a, t_b \in T$:

1) $t_a < t_b$: if there is a trace $\sigma = t_1 t_2 t_3 \cdots t_n$, $\sigma \in L$, $t_a = t_i$ and $t_b = t_{i+1}$;

2) $t_a \| t_b$: if and only if $t_a < t_b$ and $t_a > t_b$;

3) $t_a \# t_b$: if neither $t_a < t_b$ nor $t_b t_a$;

4) $t_a < t_b$: if $t_a < t_b$, and not $t_b < t_a$.

The main idea of our algorithm is as follows.

1) Firstly, we define the first and the last transitions.

2) Secondly, we construct the set of pairs of transitions which have the $\lessdot$ relation. Given a set X_L, consider sets of transitions A and B, if for all t_1 included in A, t_2 included in B, the relation between t_1 and t_2 is $t_1 \lessdot t_2$, then put (A, B) into X_L.

3) And then, we refine X_L by taking only the largest elements with respect to set inclusion. It assures that the transitions that have choice relation share the common place. Y_L is the result set after refinement.

4) Between every pair of transitions, we build a place to connect them, as well as the arc to connect the transitions and places. But we should notice that the set of places built here is only a temporal one, not the final one. Then, the temporal arc set between transition and place is also constructed. For each A, B in Y_L, each t_a in A and each t_b in B, makes t_{ai} and t_{bi} connected by a place p_i. Then make an arc from t_{ai} to p_i, and an arc from p_i to t_{bi}. The temporal sets of places and arcs are built for reduction of the Petri net and computation of the weight of arc and of the capacity of places after net reduction.

5) Add a start place before all the first transitions in the set, and add end places behind every last transition. In our net, there is only one start place while there are multiple end places.

6) So far, a net has been constructed. However, there is still something to do. One is that the net needs to be reduced, for the places in our net need to be synchronizers and the net must follow constrains of the logic and semantics model. Here we refer to the reduction rules in [11]. It assures that the mined net has exact amount of places. Each place has a set of pre transitions and a set of post transitions. To do the reduction, we do a forward scan and then a reverse scan on the Petri net iteratively, until no more places can be reduced.

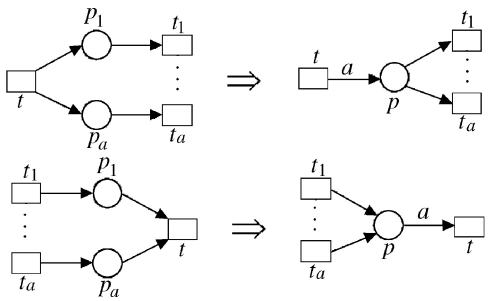

Fig.3. Reduction rules.

7) Later, the weight on the arc can be computed with the information recorded in the fourth step and reduction rule; meanwhile, we can get capacity of each synchronizer in the same way.

8) Only with the logic, the process model is not completed. We need to add workflow semantics on the logic layer. This means we should construct V, W, W_r, R, M_T and M_0.

9) Till now, the mining is over and the process model is rediscovered.

3.3 Mining Algorithm

Let L be a workflow log, and σ be a sequence of transitions in L. The algorithm is as follows:

Mining_Process (L):

1. $T_L = \{t \in T | \exists_{\sigma \in L}\, t \in \sigma\}$,
2. $T_{first} = \{t \in T | \exists_{\sigma \in L}\, t = first(\sigma)\}$,
3. $T_{last} = \{t \in T | \exists_{\sigma \in L}\, t = last(\sigma)\}$,
4. $X_L = \{(A, B) | A \subseteq T_L \wedge B \subseteq T_L \wedge \forall t_a \in A\ \forall t_b \in B,\ t_a \lessdot t_b \wedge\ \forall t_{a1} t_{a2} \in A,\ t_{a1} \# t_{a2} \wedge \forall t_{b1}, t_{b2} \in B,\ t_{b1} \# t_{b2}\}$,
5. $Y_L = \{(A, B) \in X_L\ | \forall (A', B') \in X_L,\ A \subseteq A' \wedge B \subseteq B' \Rightarrow (A, B) = (A', B')\}$,
6. $P'_L = \{p(t_a, t_b) | t_a \in A,\ t_b \in B \wedge (A, B) \subseteq Y_L\} \cup \{p(null, t_i) |\ t_i \in T_{first}\} \cup \{p(t_o, null) | t_o \in T_{last}\}$
7. $F' = \{(t, p) | t \in T_L,\ p \in P'_L \wedge t \in p.pre\} \cup \{((p, t) | t \in T_L,\ p \in P'_L \wedge t \in p.post\}$ $W' = \{(f, 1) | f \in F'\}$ $K' = \{(p, 1) | p \in P'_L\}$
8. $(P_L, W, K) = Reduce\_net(P'_L, W', K')$;
   $F = \{(t, p) | t \in T_L,\ p \in P_L \wedge t \in p.pre\}$
   $\cup \{((p, t) | t \in T_L,\ p \in P_L \wedge t \in p.post\}$
9. $M_T = \{(t, guard, body) | t \in T_L\}$
   $V = \{v | \exists_{(t, guard, body) \in M_T} v \in guard \vee v \in body\}$
   $W_r = \{(t, v) | t \in T_L,\ v \in V \wedge \exists_{m_t \in M_T}\ v \in m_t.body.l\}$
   $R = \{(t, v) | t \in T_L,\ v \in V \wedge \exists_{m_t \in M_T}\ v \in m_t.body.r$
   $\wedge \exists_{m_t \in M_T}\ v \in m_t.guard\}$
   $M_0 = \{(p, 1) | p \in \{p(null, t_i) | t_i \in T_{first}\} \cup$
   $\{(p, 0) | p \notin \{p(null, t_i) | t_i \in T_{first}\}$
10. Synchro-Net $(L) = (P_L, V, T_L, F, K, W, R, W_r, M_T, M_0)$.

Reduce_net (P'_L, W', K'):

$Flag = 1$

While $Flag! = 0$

For each $t_i \in T,\ i = 1, 2, \ldots, n$

$P_{temp_i} = \{p | t_i \{p | t_i \in p.pre \wedge p \in P'_L\}$

$P_i^* = \{\, p | p.pre = \cup p'.pre \wedge p.post = \cup p'.post \wedge p' \in P_{temp_i}\}$

$P'_{L_1} = P'_L;\ P'_{L_i} = (P'_{L_{i-1}} - P_{temp_i}) \cup P_i^*$

$W_{temp_i} = \{\, (f, w) | f = (t, p') \wedge t \in \cup p'.pre \wedge p' \in P_{temp_i}\} \cup \{(f, w) | f = (p', t) \wedge t \in \cup p'.post \wedge p' \in P_{temp_i}\}$

$W_i^* = \{\, (f, w) | f = (t, p) \wedge p \in P_i^* \wedge w = \Sigma w' \wedge (f'', w') \in W_{temp_i} \wedge f'' \in (t, p') \wedge t \in \cup p'.pre \wedge p' \in P_{temp_i}\}$

$W'_1 = W';\ W'_i = (W'_{i-1} - W_{temp_i}) \cup W_i^*$

$K'_1 = K';\ K'_i = (K'_{i-1} - \{(p, k) | p \in P_{temp_i}\}) \cup \{(p, k) | p \in P * i' \wedge k = \Sigma k, (p, k) \in K'_{i-1}, p \in P_{temp_i}\}$

$P''_L = P'_{L_n};\ W'' = W'_n;\ .K'' = K'_n$

For each $t_i \in T,\ i = n, n-1, \ldots, 1$

$P'_{temp_i} = \{p | t_i \in p.post \wedge p \in P''_L\}$

$P_i^{*\prime} = \{\, p | p.pre = \cup p'.pre \wedge p.post = \cup p'.post \wedge p' \in P'_{temp_i}\}$

$P''_{L_n} = P''_L;\ P''_{L_{i-1}} = (P''_{L_i} - P'_{temp_i - 1}) \cup P_{i-1}^{*\prime}$

$W'_{temp_i} = \{\, (f, w) | f = (t, p') \wedge t \in \cup p'.pre \wedge p' \in P'_{temp_i}\} \cup \{(f, w) | f = (p', t) \wedge t \in \cup p'.post \wedge p' \in P'_{temp_i}\}$

$W_i^{*\prime} = \{\, (f, w) | f = (t, p^* i) \wedge w = \Sigma w' \wedge (f'', w') \in W'_{temp_i} \wedge f'' \in (t, p') \wedge t \in \cup p'.pre \wedge p' \in P'_{temp_i}\}$

$W''_n = W'';\ W''_{i-1} = (W'_i - W'_{temp_i - 1}) \cup W*'_{i-1}$

$K''_n = K'';\ K''_{i-1} = K''_i - \{(p, k) | p \in P'_{temp_i - 1}\} \cup \{(p, k) | p \in P_{i-1}^{*\prime\prime} \wedge k = \Sigma k, (p, k) \in K''_i, p \in P'_{temp_i - 1}\}$

$P_L = P''_{L_1}$; $P'_L = P_L$; $W = W''_1$; $W' = W$; $K = K''_1$; $K' = K$

If $P_{temp_i} = \emptyset \wedge P'_{temp_i} = \emptyset$ **then** $Flag$ =0;

Return (P_L, W, K);

4 Example and Evaluation

To illustrate the process mining algorithm above, we consider the engine log shown in Table 1. The log contains information about four process instances (1001∼1004) and nine tasks ($t_1 \sim t_9$). It should be noticed that the information of guard and body in transitions and variables' values is omitted in the table for clarity. In fact, it is also kept in the log database.

Table 1. Event Log

PIProcessID	TaskID	PIProcessID	TaskID
1001	T1	1004	T3
1002	T1	1002	T8
1003	T1	1003	T4
1001	T2	1003	T5
1002	T3	1004	T4
1002	T2	1001	T6
1001	T3	1002	T9
1003	T3	1004	T5
1001	T4	1003	T7
1004	T1	1003	T9
1003	T2	1001	T9
1002	T4	1004	T7
1001	T5	1004	T9
1004	T2		

Based on the information and making some assumptions about the completeness of the log, we can deduce a process model. In this example, following the mining algorithm, we can deduce a process model presented by Synchro-net $(P, V, T, F, K, W, R, W_r, M_T, M_0)$, which actually describes a business process of Land and Resource Bureau.

1. $T_L = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9\}$
2. $T_{first} = \{t_1\}$
3. $T_{last} = \{t_9\}$
4. $X_L = \{\langle\{t_1\},\{t_2\}\rangle, \langle\{t_1\},\{t_3\}\rangle, \langle\{t_2\},\{t_4\}\rangle, \langle\{t_3\},\{t_4\}\rangle, \langle\{t_4\},\{t_5\}\rangle, \langle\{t_4\},\{t_8\}\rangle \langle\{t_5\},\{t_6\}\rangle, \langle\{t_5\},\{t_7\}\rangle, \langle\{t_6\},\{t_9\}\rangle, \langle\{t_7\},\{t_9\}\rangle, \langle\{t_8\},\{t_9\}\rangle\}$
5. $Y_L = \{\langle\{t_1\},\{t_2\}\rangle, \langle\{t_1\},\{t_3\}\rangle, \langle\{t_2\},\{t_4\}\rangle, \langle\{t_3\},\{t_4\}\rangle, \langle\{t_4\},\{t_5,t_8\}\rangle, \langle\{t_5\},\{t_6,t_7\}\rangle, \langle\{t_6,t_7,t_8\},\{t_9\}\rangle\}$
6. $P'_L = \{\langle\text{null}, \{t_1\}\rangle, \langle\{t_1\},\{t_2\}\rangle, \langle\{t_1\}, \{t_3\}\rangle, \langle\{t_2\}, \{t_4\}\rangle, \langle\{t_3\},\{t_4\}\rangle, \langle\{t_4\},\{t_5,t_8\}\rangle, \langle\{t_5\},\{t_6,t_7\}\rangle, \langle\{t_6,t_7,t_8\},\{t_9\}\rangle, \langle\{t_9\},\text{null}\rangle\}$ named $p'_0 \sim p'_8$
7. $F' = \{\langle p'_0, t_1\rangle, \langle t_1, p'_1\rangle, \langle t_1, p'_2\rangle, \langle p'_1, t_2\rangle, \langle p'_2, t_3\rangle, \langle t_2, p'_3\rangle, \ldots, \langle p'_7, t_9\rangle, \langle t_9, \text{null}\rangle\}$
8. $P_L = \{\langle\text{null}, \{t_1\}\rangle, \langle\{t_1\},\{t_2,t_3\}\rangle, \langle\{t_2,t_3\},\{t_4\}\rangle, \langle\{t_4\},\{t_5,t_8\}\rangle, \langle\{t_5\},\{t_6,t_7\}\rangle, \langle\{t_6,t_7,t_8\},\{t_9\}\rangle, \langle\{t_9\},\text{null}\rangle\}$ named $p_0 \sim p_6$

   $F = \{\langle p_0, t_1\rangle, \langle t_1, p_1\rangle, \langle p_1, t_2\rangle, \langle p_1, t_3\rangle, \langle t_2, p_2\rangle, \langle t_3, p_2\rangle, \langle p_2, t_4\rangle, \langle t_4, p_3\rangle, \ldots, \langle p_5, t_9\rangle, \langle t_9, p_6\rangle\}$ named $f_0 \sim f_{17}$

   $W = \{\langle f_0, 1\rangle, \langle f_1, 2\rangle, \langle f_3, 1\rangle, \langle f_4, 1\rangle, \langle f_5, 1\rangle, \langle f_6, 2\rangle, \langle f_7, 1\rangle, \langle f_8, 1\rangle, \langle f_9, 1\rangle, \langle f_{10}, 1\rangle, \langle f_{11}, 1\rangle, \langle f_{12}, 1\rangle, \langle f_{13}, 1\rangle, \langle f_{14}, 1\rangle, \langle f_{15}, 1\rangle, \langle f_{16}, 1\rangle, \langle f_{17}, 1\rangle\}$

   $K = \{\langle p_0, 1\rangle, \langle p_1, 2\rangle, \langle p_2, 2\rangle, \langle p_3, 1\rangle, \langle p_4, 1\rangle, \langle p_5, 1\rangle, \langle p_6, 1\rangle\}$
9. $M_T = \{\langle t_1, \text{null}, \text{null}\rangle, \langle t_2, \text{null}, \text{null}\rangle, \langle t_3, \text{null}, \text{null}\rangle, \langle t_4, \text{true}, x := D\rangle, \langle t_5, x, \text{null}\rangle, \langle t_6, y > 35, \text{null}\rangle, \langle t_7, y \leqslant 35, \text{null}\rangle, \langle t_8, \neg x, \text{null}\rangle, \langle t_9, \text{null}, \text{null}\rangle\}$

   $V = \{x, y\} W_r = \{\langle t_4, x\rangle\}$

   $R = \{\langle t_5, x\rangle, \langle t_6, x\rangle, \langle t_6, y\rangle, \langle t_7, y\rangle\}$

   $M_0 = \{\langle p_0, 1\rangle, \langle p_1, 0\rangle \langle p_2, 0\rangle \cdots \langle p_6, 0\rangle\}$
10. Synchro-net $= \{P_L, V, T_L, F, K, W, R, W_r, M_T, M_0\}$.

With our process mining method, problems such as invisible tasks and one-length loops can be dealt with at ease.

First, because there is no transition special for a routing purpose as in WF-net, there will be no invisible task in our original process model either. Therefore, the problem with our process mining, that no invisible tasks can be rediscovered, may not stand.

Second, there is no loop or return in the process model as presented by Synchro-Net, for they are the duty of workflow management, not of the workflow logic and semantics. However, when the process is in execution, actual loops may occur, or a task in model may extend to multiple copies to be executed by different participants. Hence the log may contain a sequent of transitions like $\sigma = t_1 t_2 \cdots t_k t_k \cdots t_n$, which could be called one-length loop. Obviously, with our model, the one-length loop makes no impact on the ability of the rediscovering. Both the original process model and the mined one contain exactly one copy of t_k which is dou-

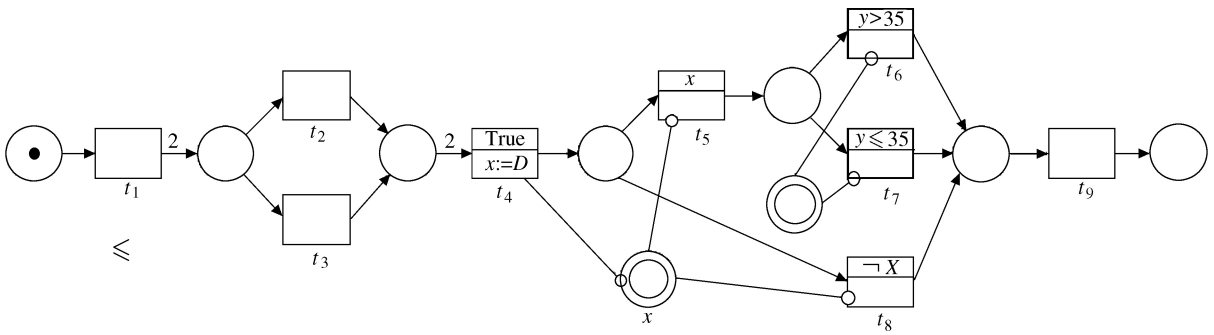

Fig.4. Rediscovered process model.

bled in execution. Consequently, the drawback of α-algorithm that cannot be dealt within one-length loop is solved in our work.

Our mining algorithm based on the model also has some drawbacks. Since return is either the duty of workflow logic nor that of workflow semantics, as mentioned above, the return of a task may occur in the execution time. The log may contain a sequent of transitions like $\sigma = t_1 t_2 t_3 t_2 t_3 \cdots t_n$, which could be called two-length loop or jump. In this case, our algorithm cannot rediscover the exactly same process model as the original one.

5 Conclusion and Future Work

In this paper, a new process mining algorithm is presented, which is based on a synchronization-based model of workflow logic and semantics. Since tasks take time, parallelism can be detected explicitly. According to the relations of tasks in a complete log, a Petri net can be constructed following the algorithm. Some of the known problems such as invisible tasks and short-loops are tackled by the algorithm together with the model.

As can be conceived, our future work will focus on the following aspects. First of all, we will try to find a way to solve the problems such as two-length loop or jump in process mining. And then, we may plan to apply the mining algorithm in practice and try to reduce the running time so as to improve the performance of the mining procedure.

References

[1] Gianluigi Greco, Antonella Guzzo, Giuseppe Manco, Domenico Saccà. Mining, reasoning on workflows. *IEEE Trans. Knowledge and Data Engineering,* April 2005, 17(4): 519–534.

[2] Wil van der Aalst, Kees Max van Hee. Workflow Management: Models, Methods and Systems. Cambridge, Massachusetts, London: The MIT Press, 2002, pp.22–73.

[3] A K A de Medeiros, W M P van der Aalst, A J M M Weijters. Workflow Mining: Current Status and Future Directions. Meersman R *et al.* (eds.), CoopIS/DOA/ODBASE 2003, LNCS 2888, Berlin, Heidelberg: Springer-Verlag, 2003, pp.389–406.

[4] W M P van der Aalst, A J M M Weijters, L Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowledge and Data Engineering,* September 2004, 16(9): 1128–1142.

[5] Laura Maruster, A J M M (Ton) Weijters, W M P van der Aalst, Antal van den Bosch. Process mining: Discovering direct successors in process logs. *Computers in Industry,* April 2004, 53(3): 231–244.

[6] A J M M Weijters, W M P van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering,* 2001, 10(2): 151–162.

[7] A K A de Medeiros, B F van Dongen, W M P van der Aalst, A J M M Weijters. Process mining: Extending the α-algorithm to mine short loops. BETA Working Paper Series, WP 113, Eindhoven University of Technology, Eindhoven, 2004.

[8] Lijie Wen, Jianmin Wang, Wil M P van der Aalst, Zhe Wang, Jiaguang Sun. A novel approach for process mining based on event types. Tsinghua University and Eindhoven University of Technology, ISBN 90-386-2057-8/ISSN 1386-9213, WP 118, May 2004.

[9] Gianluigi Greco, Antonella Guzzo, Giuseppe Manco, Domenico Saccà. Mining frequent instances on workflow. *Workshop on ID&CBM,* Hinterzarten, Germany, March 12, 2004, pp.209–221.

[10] Gianluigi Greco, Antonella Guzzo, Giuseppe Manco, Domenico Saccà. On the mining of complex workflow schemas. In *Proc. Italian Conference on Advanced Database Systems — SEBD04.* S. Margherita di Pula (CA), Italy, 2004, pp.118–129.

[11] Chongyi Yuan. Principals and Application of Petri Nets. Publishing House of Electronics Industry, 2005, pp.213–258.