Refactor Business Process Models with Maximized Parallelism

Tao Jin, Jianmin Wang, Yun Yang, *Senior Member, IEEE*, Lijie Wen, and Keqin Li, *Fellow, IEEE*

Abstract—With the broad use of business process management technology, there are more and more business process models. Since the ability of different modelers is different, the quality of these models varies. A question arises here is that, can we refactor these models to improve the quality as practised in software engineering? Business process modeling can be regarded as declarative programming, and business process models can be used to drive the process aware information systems, which are generally developed with model driven architecture, so business process models are crucial for the efficiency of process aware information systems. In this paper, we propose a novel approach on how to systematically refactor business process models with parallel structures for sequence structures for the first time. More specifically, we analyze the real causal relations between business tasks based on data operation dependency analysis, and refactor business process models with process mining technology. After comprehensive model refactoring, parallel execution of business tasks can be maximized, so the efficiency of business processing can be improved, that is, the quality of business process models can be improved. Analysis and experiments show that our approach is effective and efficient.

Index Terms—Business process, model, refactor, parallel

1 Introduction

Business process management (BPM) technology can be used to construct and update process aware information systems (PAISs) quickly [1]. The key idea of BPM technology is that business processes can be modeled in businessprocess models, which describe what tasks should be executed to complete some business objectives and what their execution orders are (from a control flow perspective), together with what data should be processed (from a data perspective) and who should be responsible for what task (from a resource perspective). With the help of BPM technology, process aware information systems can be developed with model driven architecture. Although the business processes in different enterprises are different, process aware information systems have many things in common except the business process models. That is, on one hand, different enterprises can use the same process aware information system platform, and configure their own business process models to drive their process aware information systems respectively. On the other hand, when there are some changes on the market, governmental policies and so on, enterprises can update their process aware information systems according to these changes quickly by only changing the business process models instead of building new process aware information systems. That is why BPM technology can be used to speed up the construction and updating of process aware information systems

With the broad use of BPM technology, there are more and more business process models. In some enterprises, there are thousands of models [2]. Since business process modeling is time-consuming and error-prone [3], and the ability of different modelers varies, the quality of business process models varies. In the area of software engineering, there are some works and tools on source code refactoring. A question which arises here is that, can we refactor business process models to improve the quality as well? In fact, business process modeling can be regarded as declarative programming. There are many similarities between business process modeling and software coding. For example, both activities must correspond to some language syntax, and both business process models and source codes can be executed by computer systems to complete some objectives. So some existing refactor technologies in software engineering can be adapted for workflow model refactoring. In this paper, we attempt to solve a new refactoring problem for business process models.

The problem to be solved in this paper is described informally as follows. Given a business process model, how to refactor it with parallel structures?

To solve this problem, we analyze the data operation dependency between tasks, and check whether there is a real causal relation between two investigated tasks. For two investigated tasks with a causal relation in the original model, if there is not any data operation dependency between them, we can refactor them into a parallel structure.

With parallelism refactoring, business tasks can be executed in parallel as much as possible, so that the efficiency of business processing can be improved, i.e., the quality of

- *T. Jin, J. Wang, and L. Wen are with the School of Software, Tsinghua University, Beijing 100084, China. E-mail: jintao05@gmail.com, wenlj00@mails.thu.edu.cn, jimwang@tsinghua.edu.cn.*
- *Y. Yang is with the Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne 3122, Australia. E-mail: yyang@swin.edu.au.*
- *K. Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561. E-mail: lik@newpaltz.edu.*

Manuscript received 26 Feb. 2014; revised 24 Aug. 2014; accepted 9 Dec. 2014. Date of publication 17 Dec. 2014; date of current version 15 June 2016. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TSC.2014.2383391

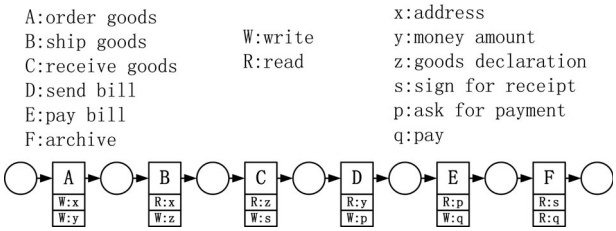

Fig. 1. A business process model example represented as a Petri net.

process models can be improved. Our contribution can be summarized as follows.

- *Problem*. We propose the problem of refactoring business process models with parallel structures for the first time in the literature.
- *Approach*. We propose an approach to refactor business process models with parallel structures. We show how process mining technology can be used for parallelism refactoring for the first time.
- *Tool*. We implement and evaluate our approach in BeehiveZ system, which is an open source system.

The rest of this paper is organized as follows. Section 2 introduces a motivating example and analyzes the problem to be solved in this paper. Section 3 introduces the definitions used throughout this paper. Section 4 describes how to refactor business process models with parallel structures in detail. Section 5 addresses the implementation and evaluation of our approach. Section 6 discusses the related work. Section 7 concludes this paper and points out future work.

2 Motivating Example and Problem Analysis

In this section, we first introduce a motivating example, and then analyze the problem to be solved in this paper.

2.1 Motivating Example

Example 1. Fig. 1 shows a business process model example represented as a Petri net. This example is adapted from the example in [4]. The rectangles denote tasks, and the circles denote states. Both control flow perspective and data perspective are considered in this example model. This model describes an online-shopping processing. First, the buyer orders goods (task A) through the Internet, and the address of the buyer (variable x) together with the money the buyer should pay (variable y) are written. Next, the seller ships goods (task B), the address of the buyer (variable x) is read, and the goods shipped (variable z) is written. When the buyer receives goods (task C), the goods shipped (variable z) is read, and the buyer signs for the goods (variable s is written). Then, the seller sends the bill to the buyer (task D), the money the buyer should pay (variable y) is read, and the requirement of payment (variable p) is written. After that the buyer pays the bill (task E), the requirement of payment (variable p) is read, and the completion of payment (variable q) is written. Finally, the seller archives this transaction (task F), the signature of the buyer (variable s) and the completion of payment (variable q) are read.

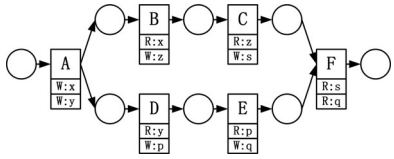

Fig. 2. A refactored model with a parallel structure for the model in Fig. 1.

All the tasks in Fig. 1 are arranged in a sequential structure, so the time for the whole business processing is the sum of the time for processing each task. Intuitively, to execute task D, it is unnecessary to wait the completion of task C, because there are no dependency relations between task D and task C. If there is a causal relation between two tasks, we can denote this causal relation through some data operation dependency. Based on the data operation dependency analysis, there are no causal relations between tasks D and C, so these two tasks are unnecessary to be executed sequentially as they can be executed in parallel.

Based on the idea above, the model in Fig. 1 can be refactored, and the refactored model is presented in Fig. 2. The time for completing the new process is the time of the critical path. Compared to the model in Fig. 1, the completion time of the refactored model should be less. So this refactoring technology can improve the efficiency of business processing, in other words, this refactoring technology can improve the quality of process models.

2.2 Problem Analysis

Let $T(A)$ denote the time of completing task A since it is enabled to be executed. For the model in Fig. 1, the time for completing a business case can be calculated as $T_{original} = T(A) + T(B) + T(C) + T(D) + T(E) + T(F)$. For the refactored model in Fig. 2, the time for completing a business case can be calculated as $T_{refactored} = T(A) + max(T(B) + T(C), T(D) + T(E)) + T(F)$. Obviously, $T_{original} > T_{refactored}$. The efficiency of business processing can be improved if the tasks involved can be executed in parallel. So, to improve the efficiency of business processing for better quality of process models, we need to refactor models with parallel structures as much as possible.

Since the execution of tasks would consume some information and produce some information, and information can be encoded as data, if there is a real causal relation between two tasks, there must be some data operation dependency. For example, in the model in Fig. 1, task A writes data x and task B reads data x, so task B can only be executed after the completion of task A, i.e. , there is a real causal relation between tasks A and B. Similarly, there is a real causal relation between tasks B and C, C and F, A and D, D and E, E and F.

If there is not any data operation dependency, the causal relation or transitive causal relation between two investigated tasks does not exist, and these tasks can be refactored into parallel structures. For example, in the model in Fig. 1, there is not any data operation dependency between tasks C and D, so they can be executed in parallel, similarly for tasks C and E, B and D, B and E. Here, the causal relation between C and D does not exist, and the transitive causal relations between B and D, B and E, C and E do not exist.

For two tasks with a transitive causal relation, there is some data operation dependency. After the tasks between them on the causal chain are refactored into parallel structures, these two tasks may be connected directly. That is, the transitive causal relation can be changed into a direct causal relation. For example, in the model in Fig. 1, there is a transitive causal relation between tasks C and F. Task C writes data s, and task F reads data s. In the refactored model in Fig. 2, there is a causal relation between C and F because tasks D and E are refactored into a parallel structure.

As a conclusion, to improve the efficiency of business processing for better quality of process models, we need to refactor the models with parallel structures as much as possible. To solve the refactoring problem proposed in this paper, we need to extract all the causal relations and transitive causal relations from the given models, analyze data operation dependency, change some false causal relations and false transitive causal relations to parallel relations, and change some false transitive causal relations to causal relations.

There may be constraints from the resource perspective that hinder the parallel execution, and this problem can be solved by adding resources, so we think that the resource perspective should not impact the models, especially our primary goal is maximized parallelism. So we ignore the resource perspective in this paper.

3 Preliminaries

Since Petri net has a sound formalization foundation and is easy to understand and use, it was introduced into business process management area for modeling, verification and analysis [5].

Definition 1 (Petri net). *A petri net is a triple $N = (P, T, F)$, where P and T are finite disjoint sets of places and transitions $(P \cap T = \emptyset)$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs (flow relation).*

We write $X = (P \cup T)$ for all nodes of a Petri net. For a node $x \in X, \bullet x = \{y \in X | (y, x) \in F\}, x\bullet = \{y \in X | (x, y) \in F\}$. A node $x \in X$ is an input (output) node of a node $y \in X$, iff $x \in \bullet y (x \in y\bullet)$.

Definition 2 (Petri net semantics). *Let $N = (P, T, F)$ be a Petri net.*

- *$M : P \to \mathbb{Z}$ is a marking of N, where $\mathbb{Z}$ is the set of nonnegative integer numbers. A marking indicates that in some state what places have how many tokens. $\mathbb{M}$ denotes all markings of N. $M(p)$ denotes the number of tokens in place p. $[p]$ denotes the marking when place p contains just one token and all the other places contain no tokens.*
- *For any transition $t \in T$ and any marking $M \in \mathbb{M}$, t is enabled in M, denoted by $(N, M)[t\rangle$, iff $\forall p \in \bullet t : M(p) \geq 1$.*
- *Marking M' is reached from M by firing of t, denoted by $(N, M)[t\rangle(N, M')$ or $M \xrightarrow{t} M'$ for simplicity, meaning that $M' = M - \bullet t + t\bullet$, i.e., one token is taken from each input place of t and one token is added to each output place of t.*
- *A firing sequence $\sigma = t_0 t_1 \ldots t_{n-1}$ leads from marking M_0 to marking M_n, i.e. , $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \cdots \xrightarrow{t_{n-1}} M_n$. It can be denoted as $M_0 \xrightarrow{\sigma} M_n$.*
- *For any two markings $M, M' \in \mathbb{M}$, M' is reachable from M in N, denoted by $M' \in [N, M\rangle$, iff there exists a firing sequence σ leading from M to M'.*
- *A net system is a pair $\Sigma = (N, M_0)$, where N is a net and M_0 is the initial marking of N.*

Workflow net (WF-net) is a subclass of Petri net designed to represent business process models. A Workflow net is a Petri net with two special places, source place $i : \bullet i = \emptyset$ and sink place $o : o\bullet = \emptyset$. All the nodes except the source place and sink place are on the path from the start to the end. A Workflow net system is a pair (N, M_i), where $M_i = [i]$.

Definition 3 (Sound WF-net). *A WF-net $N = (P, T, F)$ is sound iff:*

- *$\forall M([i] \xrightarrow{*} M \Rightarrow \xrightarrow{*} [o])$. That is, each task/condition is on the path from i to o.*
- *$\forall M([i] \xrightarrow{*} M \wedge M \geq [o] \Rightarrow M = [o])$. That is, for any case, the process will terminate eventually and the moment the process terminates there is only one token in place o and all the other places are empty.*
- *$\forall t \in T(\exists M, M'([i] \xrightarrow{*} M \xrightarrow{t} M'))$. That is, there should be no dead tasks.*

If a WF-net is not sound, there must be some errors in that model and these errors should be eliminated. There would be at most one token in any place of a sound workflow net during the execution. More details of Petri net can be found in [6], and more details of workflow net can be found in [5].

Definition 4 (Implicit place). *Let $N = (P, T, F)$ be a Petri net with initial marking s. A place $p \in P$ is called implicit in (N, s) iff, for all reachable markings $s' \in [N, s\rangle$ and transitions $t \in p\bullet$, $s' \geq \bullet t \setminus \{p\} \Rightarrow s' \geq \bullet t$.*

The addition of implicit places does not change the behavior of the net. Please see [7] for details.

Definition 5 (SWF-net). *A WF-net $N = (P, T, F)$ is a SWF-net (Structured workflow net) iff:*

- *For all $p \in P$ and $t \in T$ with $(p, t) \in F$, $|p \bullet| > 1$ implies $|\bullet t| = 1$.*
- *For all $p \in P$ and $t \in T$ with $(p, t) \in F$, $|\bullet t| > 1$ implies $|\bullet p| = 1$.*
- *There are no implicit places.*

SWF-net is a subclass of WF-net, and SWF-net is more readable and understandable. More information can be found in [7]. It is suggested that the business process model should be as structural as possible [9]. There are already some work on structuring business process models, for example, the work in [10] can structure acyclic business process models. So in this paper, we assume all the given business process models are sound SWF-nets.

Definition 6 (Log based task relations). *Let W be all the traces (firing sequences) of a workflow net system $N = (P, T, F)$ with the initial marking as M_i. For $a, b \in T$:*

- $a \triangle_W b : \exists \sigma = t_1 t_2 t_3 \cdots t_n \in W, x \in \{1, 2, \ldots, n-2\}, t_x = t_{x+2} = a, t_{x+1} = b$, *that is, there is a trace like* $\cdots aba \cdots$.
- $a \diamond_W b$: $a \triangle_W b \wedge b \triangle_W a$.
- $a >_W b$:$\exists \sigma = t_1 t_2 t_3 \cdots t_n \in W, x \in \{1, 2, \ldots, n-1\}, t_x = a, t_{x+1} = b$.
- $a \rightarrow_W b$: $a >_W b \wedge b \not>_W a \vee a \diamond_W b$.
- $a \leftarrow_W b$: $a \not>_W b \wedge b >_W a \vee a \diamond_W b$.
- $a \#_W b$: $a \not>_W b \wedge b \not>_W a$.
- $a ||_W b$: $a >_W b \wedge b >_W a \wedge a \not\diamond_W b$.

Definition 7 (α mining algorithm). *Let W be a workflow log over T. $\alpha(W)$ is defined as follows:*

1) $T_W = \{t \in T | \exists \sigma \in W (t \in \sigma)\}$
2) $T_I = \{t \in T | \exists \sigma \in W (t = first(\sigma))\}$, *i.e., the set of transitions executed first.*
3) $T_O = \{t \in T | \exists \sigma \in W (t = last(\sigma))\}$, *i.e., the set of transitions executed last.*
4) $X_W = \{(A, B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall a \in A \forall b \in B (a \rightarrow_W b) \wedge \forall a_1, a_2 \in A (a_1 \#_W a_2) \wedge \forall b_1, b_2 \in B (b_1 \#_W b_2)\}$
5) $Y_W = \{(A, B) \in X_W | \forall (A', B') \in X_W (A \subseteq A' \wedge B \subseteq B' \Rightarrow (A, B) = (A', B'))\}$
6) $P_W = \{p_{(A,B)} | (A, B) \in Y_W\} \cup \{i_W, o_W\}$
7) $F_W = \{(a, p_{(A,B)}) \mid (A, B) \in Y_W \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A, B) \in Y_W \wedge b \in B\} \cup \{(i_W, t) \mid t \in T_I\} \cup \{(t, o_W) | t \in T_O\}$
8) $\alpha(W) = (P_W, T_W, F_W)$

In brief, α mining algorithm tries to find all the places between tasks and connect these places and tasks. In [7], it is proved that, given a complete log based on the relations of $>_W$, α mining algorithm can discover a unique SWF-net.

The above definitions are reused from other papers, which are referenced accordingly. To explain our problem and approach, we need new definitions as follows.

Definition 8 (Transitive causal relation). *The relation of $\rightarrow_W$ in Definition 6 can be transitive. If $a, b, c \in T \wedge a \rightarrow_W b \wedge b \rightarrow_W c, a \twoheadrightarrow_W c$. The relation of $\twoheadrightarrow_W$ can also be transitive.*

Example 2. For the model in Fig. 1, since $A \rightarrow_W B \wedge B \rightarrow_W C$, $A \twoheadrightarrow_W C$. Similarly, $C \rightarrow_W D \wedge D \rightarrow_W E$, so $C \twoheadrightarrow_W E$. Since $A \twoheadrightarrow_W C \wedge C \twoheadrightarrow_W E$, $A \twoheadrightarrow_W E$.

Since we refactor business process models based on data operation dependency analysis in this paper, we introduce data into WF-net as follows.

Definition 9 (DWF-net). *$DN = (P, T, F, D, Wt, Rd)$ is a DWF-net (workflow net with data), where:*

- *$N = (P, T, F)$ is a WF-net.*
- *D is the set of data that are operated by WF-net $N = (P, T, F)$.*
- *$Wt : D \nrightarrow 2^T$ describes what data are written by which task.*
- *$Rd : D \nrightarrow 2^T$ describes what data are read by which task.*

Example 3. The model in Fig. 1 is a DWF-net, where all the circles are places, $T = \{A, B, C, D, E, F\}$, and all the arrows connecting the places and transitions constitute the set F, $D = \{x, y, z, s, p, q\}$, and the Wt and Rd are denoted on the transitions, for example, $Wt(x) = \{A\}$, $Rd(x) = \{B\}$.

Definition 10 (Causal relation based on data operation). *In a DWF-net $DN = (P, T, F, D, Wt, Rd)$, if $t_1, t_2 \in T$ satisfy:*

- $t_1 \rightarrow_W t_2 \vee t_1 \twoheadrightarrow_W t_2$
- $\exists x \in D (t_1 \in Wt(x) \wedge t_2 \in Rd(x) \vee t_1 \in Wt(x) \wedge t_2 \in Wt(x) \vee t_1 \in Rd(x) \wedge t_2 \in Wt(x))$

t_2 must be executed after the completion of t_1 based on read-write operations of x, denoted as $t_1 >_x t_2$.

Example 4. For the model in Fig. 1, since $A \rightarrow_W B \wedge A \in Wt(x) \wedge B \in Rd(x)$, $A >_x B$. That is, B must be executed after A based on read-write operation of x. Similarly, $B \rightarrow_W C \wedge B \in Wt(z) \wedge C \in Rd(z)$, $B >_z C$. That is, C must be executed after B based on read-write operation of z.

Definition 11 (Transitive causal relation based on data). *In a DWF-net $DN = (P, T, F, D, Wt, Rd), t_1, t_2, t_3 \in T \wedge x, y \in D \wedge t_1 >_x t_2 \wedge t_2 >_y t_3 \Rightarrow t_1 >>_* t_3$. $>>_*$ can also be transitive.*

Example 5. For the model in Fig. 1, since $A >_x B \wedge B >_z C$ (see Example 4), $A >>_* C$. That is, C must executed after A based on read-write operations of data.

Definition 12 (Core causal relation based on data). *In a DWF-net $DN = (P, T, F, D, Wt, Rd)$, if $t_1, t_2 \in T \wedge x \in D \wedge t_1 >_x t_2 \wedge \neg(t_1 >>_* t_2)$, there is a core causal relation based on data operations between t_1 and t_2, denoted as $t_1 \rightarrow_x t_2$.*

Example 6. For the model in Fig. 1, since $A >_x B \wedge \neg(A >>_* B)$, $A \rightarrow_x B$. That is, after completion of A, B can be executed immediately, because there are no transitive dependences between A and B based on read-write operations of data.

Based on the above definitions, the problem to be solved in this paper can be specified as follows. Given a sound structured workflow net with data operations, how to refactor the model with parallel structures so that the tasks in the model can be executed in parallel as much as possible? There are two assumptions here:

1) *The given model is a sound structured workflow net with data.* Besides the work on structuring process models aforementioned (e.g., the work in [10]), there are some works on how to check whether a given workflow net is sound. For example, in [11], the authors proposed a tool named woflan to check whether the control flow is sound, and in [12] the authors proposed an approach to check whether a WF-net with data is sound.
2) *If two tasks have causal or transitive causal relations, there must be some data operation dependency.* There is data processing during business processing, and different tasks have different data processing. The execution order of tasks having causal relation or transitive causal relation will affect the result of data processing. If there is no data operation dependency between two tasks, the execution order of these two tasks does not matter for the data processing result.

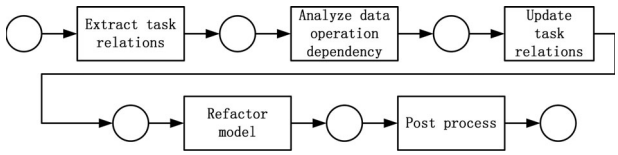

Fig. 3. Refactoring process.

4 Refactoring Process

The process of refactoring models with parallel structures is presented in Fig. 3. There are five steps.

- Step 1: extract task relations from the original model.
- Step 2: analyze data operation dependency.
- Step 3: update task relations.
- Step 4: refactor the model with process mining technology.
- Step 5: post process the mined model.

The details of how each step works can be found in the following sections.

4.1 Extract Task Relations

At this stage, we try to find all task relations. Since we assume that all the given models are sound structured workflow nets, we can extract all the task relations from the model directly. According to Definition 6, we can see that the relation between every two tasks is unique, that is, one relation in $\rightarrow_W \vee \leftarrow_W$, $\#_W$, $||_W$. Since the relation of $\leftarrow_W$ is the inverse version of $\rightarrow_W$; we only need to compute the relations of $\rightarrow_W$. We can traverse all the places in the original model to obtain the relations of $\rightarrow_W$ directly. The relations between every predecessor transition and every successor transition of the investigated place are $\rightarrow_W$. For two transitions, if their nearest common ancestor (NCA) is a third transition, the relation between these two transitions is $||_W$. If the relation between two transitions is neither $\rightarrow_W \vee \leftarrow_W$ nor $||_W$, the relation must be $\#_W$.

Theorem 1. *The relation between any two transitions in a sound SWF-net is unique, that is, one relation in $\rightarrow_W \vee \leftarrow_W$, $\#_W$, $||_W$.*

Proof. The relation of $\leftarrow_W$ is the inverse version of $\rightarrow_W$. The relation of $\#_W$ is commutative, that is, $a\#_W b \Rightarrow b\#_W a$, the same to the relation of $||_W$. According to Definition 6, the relations of $\rightarrow_W \vee \leftarrow_W$, $\#_W$ and $||_W$ are a partition among all the relations between any two transitions based on the relations of $>_W$ and $\diamond_W$. In Definition 6, if we ignore the relation of $\diamond_W$, we can see the relations of $\rightarrow_W \vee \leftarrow_W$, $\#_W$ and $||_W$ are a partition among all the relations between any two transitions based on the relations of $>_W$. then, some pairs with the relation of $\diamond_W$ are removed from $||_W$ and added to $\rightarrow_W \vee \leftarrow_W$, so the relations of $\rightarrow_W \vee \leftarrow_W$, $\#_W$ and $||_W$ are still a partition among all the relations between any two transitions. □

Example 7. The model in Fig. 1 is a sound SWF-net, the relation between any two transitions is unique, for example, $A \rightarrow_W B$, $A\#_W C$.

Theorem 2. *In a sound SWF-net $N = (P, T, F)$, $t_1, t_2 \in T$, $\exists p \in P(t_1 \in \bullet p \wedge t_2 \in p\bullet) \Leftrightarrow t_1 \rightarrow_W t_2$. That is, if there is a place p, t_1 is an input transition of p and t_2 is an output transition of p, there must be a causal relation between t_1 and t_2, and vice versa.*

Proof.

- $\Rightarrow$: After t_1 is executed, it produces a token to p that would be consumed by t_2, so $t_1 >_W t_2$. If $t_2 >_W t_1$, there must be another place connecting t_2 and t_1, that is, t_1 and t_2 are involved in a length-two loop ($t_1 \diamond_W t_2$). If there is not such a place, $\neg(t_2 >_W t_1)$. According to Definition 6, $t_1 >_W t_2 \wedge t_2 \not>_W t_1 \vee t_1 \diamond_W t_2 \Rightarrow t_1 \rightarrow_W t_2$. We can prove this theorem in another way, when α mining algorithm (Definition 7) works, it will connect t_1 and t_2 with $t_1 \rightarrow_W t_2$ through a place. So naturally, $t_1 \rightarrow_W t_2$ if $t_1 \in \bullet p \wedge t_2 \in p\bullet$.
- $\Leftarrow$: From α mining algorithm, we know that if $t_1 \rightarrow_W t_2$, they will be connected through a place, so there must be a place in the model that $t_1 \in \bullet p \wedge t_2 \in p\bullet$. □

Example 8. The model in Fig. 1 is a sound SWF-net, since there is a place between A and B, there is a causal relation between A and B.

Theorem 3. *In a sound SWF-net, if the nearest common ancestor of two transitions is a third transition, the relation between these two transitions must be $||_W$. $t_1, t_2 \in T$, $\exists c \in T(c = NCA(t_1, t_2) \wedge c \neq t_1 \wedge c \neq t_2) \Leftrightarrow t_1 ||_W t_2$.*

Proof.

- $\Rightarrow$: Let $t_1, t_2, c \in T$ and c be the nearest common ancestor of t_1 and t_2, after c is executed, the path from c to t_1 and the path from c to t_2 are enabled, and the execution of these two paths is not affected by each other, that is, t_1 can be executed after t_2 and t_1 can also be executed before t_2, so there must be $t_1 >_W t_2$ and $t_2 >_W t_1$ in the log. If $t_1 \diamond_W t_2$, t_1 and t_2 are involved in a length-two loop, the nearest common ancestor of t_1 and t_2 must be t_1 or t_2, which is not a third transition, so $t_1 \not\diamond_W t_2$. According to Definition 6, $t_1 >_W t_2 \wedge t_2 >_W t_1 \wedge t_1 \not\diamond_W t_2 \Rightarrow t_1 ||_W t_2$.
- $\Leftarrow$: If $t_1 ||_W t_2$, t_1 and t_2 can be executed in parallel, so the path to t_1 and the path to t_2 can be enabled at the same time. According to Petri net semantics (Definition 2), there must be a transition producing tokens to enable these two paths at the same time and this transition is the nearest common ancestor of t_1 and t_2. □

Example 9. The model in Fig. 2 is a sound SWF-net, the nearest common ancestor of C and E is a third transition A, so $C||_W E$.

To find the nearest common ancestor, we can backtrack from the given two transitions in the model and stop when their nearest common ancestor is found. To facilitate finding the nearest common ancestor, we first traverse the model by breadth-first algorithm and mark every node with level information. The algorithm of marking every node in a model with level information can be found in Algorithm 1, where *queue* is a queue, and *add* means adding an element

to the queue, *remove* means retrieving and removing an element from the queue.

Algorithm 1. Mark every node in a model with level information

Input: a sound SWF-net $N = (P, T, F)$
Output: a map *levels* with nodes as keys and level information as values

```
1  find the start place pSource with •pSource = ∅;
2  queue.add(pSource);
3  levels.put(pSource, 0);
4  while queue.size()> 0 do
5    node = queue.remove();
6    level = levels.get(node);
7    foreach n ∈ node• do
8      if levels.containsKey(n) == false then
9        queue.add(n);
10     levels.put(n, level + 1);
11 return levels;
```

The algorithm of finding the nearest common ancestor can be found in Algorithm 2. Algorithm 2 backtracks from two given nodes and then finds the common node between two predecessor sets. During the common node searching, the level information is used. First, only the nodes with the same level in two predecessor sets are compared. Second, the common node with the largest level is found first, which is the nearest common ancestor.

Algorithm 2. Find the nearest common ancestor

Input: two nodes in a model $n1$ and $n2$
Output: the nearest common ancestor node of $n1$ and $n2$

```
1  mark the model with level information and get levels;
2  queue1.add(n1);
3  queue2.add(n2);
4  while true do
5    backtrack from the first node in a queue with larger level
     until the levels of the first nodes of two queues are the
     same, during backtracking put the predecessors with
     smaller levels into queue and keep the queue in a
     descending order according to levels;
6    level1 = levels.get(queue1.get(0));
7    level2 = levels.get(queue2.get(0));
     // level1 == level2
8    for i = 0 : queue1.size() do
9      n1 = queue1.get(i);
10     if levels.get(n1) < level1 then
11       break;
12     for j = 0 : queue2.size() do
13       n2 = queue2.get(j);
14       if levels.get(n2) < level2 then
15         break;
16       if n1 == n2 then
17         return n1;
18   delete the nodes with level as level1 from two queues, add
     the predecessor nodes with smaller levels into queues and
     keep queues in a descending order according to levels;
```

The algorithm of computing the task relations can be found in Algorithm 3.

Algorithm 3. Compute task relations

Input: a sound SWF-net $N = (P, T, F)$
Output: the task relation matrix $CM[|T|][|T|]$

```
   // compute the relations of →W
1  foreach p ∈ P do
2    foreach a ∈ •p do
3      foreach b ∈ p• do
4        set (a →W b),(b ←W a) in CM;
   // compute the relation of ||W
5  foreach a ∈ T do
6    foreach b ∈ T do
7      if c ∈ T is the nearest common ancestor of a and b and
       a ≠ c ∧ b ≠ c then
8        set (a||W b),(b||W a) in CM;
   // compute the relations of #W
9  foreach a ∈ T do
10   foreach b ∈ T do
11     if the relation between a and b is not set before
       then
12       set (a#W b),(b#W a) in CM;
13 return CM;
```

Let n be the number of nodes in a model, and let e be the number of edges in a model. Since Algorithm 1 is a breadth-first search algorithm, the time complexity is $O(n + e)$, and $O(e)$ may vary between $O(n)$ and $O(n^2)$, the worst case time complexity of Algorithm 1 is $O(n^2)$. Algorithm 2 backtracks from two given nodes and then finds the common node between two predecessor sets. The worst case time complexity of backtracking is $O(n)$, and the worst case time complexity of comparing two predecessor sets is $O(n^2)$, so the worst case time complexity of Algorithm 2 is $O(n^3)$. In Algorithm 3, the worst case time complexity for computing the relations of $\rightarrow_W$ is $O(n^3)$, the worst case time complexity for computing the relations of $||_W$ is $O(n^5)$, and the worst case time complexity for computing the relations of $\#_W$ is $O(n^2)$, so the worst case time complexity of Algorithm 3 is $O(n^5)$.

Theorem 4. *Algorithm 3 can compute all the task relations correctly and completely.*

Proof. According to Theorem 2, the first stage of Algorithm 3 (Lines 1-4) can compute the relations of $\rightarrow_W$ correctly and completely. According to Theorem 3, the second stage of Algorithm 3 (Lines 5-8) can compute the relations of $||_W$ correctly and completely. According to Theorem 1, the third stage of Algorithm 3 (Lines 9-12) can compute the relations of $\#_W$ correctly and completely. □

After we obtain the relations of $\rightarrow_W$ using Algorithm 3, we can compute the relations of $\twoheadrightarrow_W$ by using transitive closure algorithm (Algorithm 4). The core idea is similar to the FloydWarshall algorithm [13]. Let n be the number of nodes in the model. The worst case time complexity is $O(n^3)$.

Example 10. We can extract task relations from the model in Fig. 1. There are causal relations between tasks A and B, B and C, C and D, D and E, E and F, which means that task B can be executed immediately after the completion of A, and so on. There are transitive causal relations

TABLE 1
Task Relations for the Model in Fig. 1

	A	B	C	D	E	F
A		$\rightarrow_W$	$\twoheadrightarrow_W$	$\twoheadrightarrow_W$	$\twoheadrightarrow_W$	$\twoheadrightarrow_W$
B			$\rightarrow_W$	$\twoheadrightarrow_W$	$\twoheadrightarrow_W$	$\twoheadrightarrow_W$
C				$\rightarrow_W$	$\twoheadrightarrow_W$	$\twoheadrightarrow_W$
D					$\rightarrow_W$	$\twoheadrightarrow_W$
E						$\rightarrow_W$
F						

between tasks A and C, A and D, A and E, A and F, B and D, B and E, B and F, C and E, C and F, D and F, which means that after task A is executed, task C can be executed later instead of immediately, and so on. All the task relations are shown in Table 1. Since the relations of $\#_W$ and $||_W$ are commutative, and the relation of $\leftarrow_W$ is the inverse version of $\rightarrow_W$, we omit the lower triangular matrix.

Algorithm 4. Compute transitive causal relations

Input: the task relation matrix $CM[|T|][|T|]$
Output: the task relation matrix $CM[|T|][|T|]$
1 **foreach** $c \in T$ **do**
2 **foreach** $a \in T$ **do**
3 **foreach** $b \in T \wedge (a \rightarrow_W c \vee a \twoheadrightarrow_W c) \wedge$
4 $(c \rightarrow_W b \vee c \twoheadrightarrow_W b) \wedge a \not\twoheadrightarrow_W b$ **do**
5 set $(a \twoheadrightarrow_W b)$;

4.2 Analyze Data Operation Dependency

To check whether there is a real causal relation or transitive causal relation between two tasks having a causal relation or transitive causal relation in the original model, we analyze the data operation dependency in this section. For the operation on the same data, the preceding task may write the data and then the succeeding task reads the data, denoted as W-R for simplicity. Besides, there are three other scenarios: W-W, R-W, R-R. The order of the execution of the investigated two tasks will impact the data processing result for W-R, W-W, R-W operations, so the causal relations based on data operation dependency cannot be changed.

First, we compute the causal relations based on data operation dependency according to Definition 10 (Algorithm 5). Let $|D|$ denote the number of data items operated by a model, and let n denote the number of transitions in a model. The worst case time complexity of Algorithm 5 is $O(|D| \times n^2)$.

Theorem 5. *Algorithm 5 can compute causal relations based on data operation dependency correctly and completely.*

Proof. This theorem follows Definition 10 directly. □

Example 11. For the model in Fig. 1, we obtain the causal relations based on data operation dependency, the result can be found in Table 2. For example, since there is an operation dependency based on data x between tasks B and A, B can only be executed after the completion of A.

Second, we compute the transitive causal relations based on data operation dependency according to Definition 11. The transitive closure algorithm is similar to Algorithm 4, so it is omitted here. Finally, we obtain the core causal relations based on data operation dependency according to Definition 12.

TABLE 2
Causal Relations Based on Data Operation Dependency for the Model in Fig. 1

	A	B	C	D	E	F
A		$>_x$		$>_y$		
B			$>_z$			
C						$>_s$
D					$>_p$	
E						$>_q$
F						

Algorithm 5. Compute causal relations based on data operation dependency

Input: a DWF-net $DN = (P, T, F, D, Wt, Rd)$ and its task relation matrix $CM[|T|][|T|]$
Output: the causal relations based on data operation dependency
1 **foreach** $d \in D$ **do**
2 **foreach** $t_2 \in Rd(d)$ **do**
3 **foreach** $t_1 \in Wt(d)$ **do**
4 **if** $t_1 \rightarrow_W t_2 \vee t_1 \twoheadrightarrow_W t_2$ **then**
5 set $(t_1 >_d t_2)$;
6 **foreach** $t_2 \in Wt(d)$ **do**
7 **foreach** $t_1 \in Wt(d) \cup Rd(d)$ **do**
8 **if** $t_1 \rightarrow_W t_2 \vee t_1 \twoheadrightarrow_W t_2$ **then**
9 set $(t_1 >_d t_2)$;

Example 12. For the model in Fig. 1, after we compute the causal relations based on data operation dependency as shown in Example 11, we compute the transitive causal relations and the core causal relations based on data operation dependency. The result can be found in Table 3. For example, after task A is completed, task C can be executed later instead of immediately since there is a data operation dependency chain between these two tasks.

4.3 Update Task Relations

The goal of this paper is to refactor sound structured workflow models with parallel structures for sequence structures. Based on data operation dependency analysis, we can find some tasks having causal relations or transitive causal relations in the original model without any data operation dependency. For these tasks, we can refactor them into

TABLE 3
Transitive and Core Causal Relations Based on Data Operation for the Model in Fig. 1

	A	B	C	D	E	F
A		$\rightarrow_x$	$>>_*$	$\rightarrow_y$	$>>_*$	$>>_*$
B			$\rightarrow_z$			$>>_*$
C						$\rightarrow_s$
D					$\rightarrow_p$	$>>_*$
E						$\rightarrow_q$
F						

parallel structures. Since some tasks on a sequence path can be refactored into parallel structures, the other tasks on the same path not adjacent before will be adjacent after refactoring. So there are three types of relation changes:

1) change false causal relations to parallel relations,
2) change false transitive causal relations to parallel relations,
3) change false transitive causal relations to causal relations.

Theorem 6. *The above three types of changes are complete.*

Proof. We can enumerate all the possibilities as follows. For causal relations, they can be changed into parallel relations or remain unchanged. For transitive causal relations, they can be changed into parallel relations or causal relations, or remain unchanged. So there are three types of changes in total. □

The details of each type of changes can be found in the following sections.

4.3.1 Change False Causal Relations to Parallel Relations

Theorem 7. *If* $a, b \in T \wedge a \rightarrow_W b \wedge \nexists x \in D(a >_x b)$, *we can refactor the relation between* a *and* b *to* $a||_W b$.

Proof. Since the execution order of a and b does not impact any data processing result, it is unnecessary to execute b only after the completion of a. □

Example 13. In Table 1, $C \rightarrow_W D$, but in Table 2, $\nexists x$ $(C >_x D)$, so we can refactor the relation between C and D to $C||_W D$.

4.3.2 Change False Transitive Causal Relations to Parallel Relations

Theorem 8. *If* $a, b \in T \wedge a \twoheadrightarrow_W b \wedge a \not\rightarrow_W b \wedge \nexists x \in D(a >_x b) \wedge \neg(a >>_* b)$, *we can refactor the relation between* a *and* b *to* $a||_W b$.

Proof. It is similar to Proof of Theorem 7. □

Example 14. In Table 1, $B \twoheadrightarrow_W D$, but in Tables 2 and 3, $\nexists x(B >_x D) \wedge \neg(B >>_* D)$, so we can refactor the relation between B and D to $B||_W D$. Similarly, in Table 1, $C \twoheadrightarrow_W E$ can be refactored to $C||_W E$, and $B \twoheadrightarrow_W E$ can be refactored to $B||_W E$.

4.3.3 Change False Transitive Causal Relations to Causal Relations

Theorem 9. *If* $a, b \in T \wedge a \twoheadrightarrow_W b \wedge \exists x \in D(a \rightarrow_x b)$, *we can refactor the relation between* a *and* b *to* $a \rightarrow_W b$.

Proof. Between a and b, there is no transitive data operation dependency. After a is finished, b can be executed immediately and will not impact any data processing result. □

Example 15. In Table 1, $A \twoheadrightarrow_W D$, and in Table 3, $A \rightarrow_y D$, so we can refactor the relation between A and D to $A \rightarrow_W D$. Similarly, in Table 1, $C \twoheadrightarrow_W F$ can be refactored to $C \rightarrow_W F$.

TABLE 4
Updated Task Relations for the Model in Fig. 1

	A	B	C	D	E	F
A		$\rightarrow_W$	$\twoheadrightarrow_W$	$\rightarrow_W$	$\twoheadrightarrow_W$	$\twoheadrightarrow_W$
B			$\rightarrow_W$	$\|\|_W$	$\|\|_W$	$\twoheadrightarrow_W$
C				$\|\|_W$	$\|\|_W$	$\rightarrow_W$
D					$\rightarrow_W$	$\twoheadrightarrow_W$
E						$\rightarrow_W$
F						

Example 16. After updating task relations with the above three types of changes, the relations in Table 1 is updated, as shown in Table 4.

For the above three types of changes, we only need to scan two tables such as the running examples show. Let n denote the number of transitions in a model. The time complexity for updating task relations is $O(n^2)$.

4.4 Refactor Model with Process Mining Technology

α mining algorithm first obtains the relations of $\rightarrow_W$, $\#_W$, $||_W$, and then finds the places and connects them with the transitions to construct a Petri net. So we need to change some relations of $\twoheadrightarrow_W$ to $\#_W$ first, and then use α mining algorithm to build a new SWF-net.

Theorem 10. *If* $a, b \in T \wedge a \twoheadrightarrow_W b \wedge a \not\rightarrow_W b$, $a \#_W b$.

Proof. Since $a \twoheadrightarrow_W b$, there must be a path between a and b on which there are at least two places, b cannot be executed immediately after a, that is, $a \not\succ_W b$. Since $a \not\rightarrow_W b$, a cannot be executed after b, that is, $b \not\succ_W a$. According to Definition 6, $a \not\succ_W b \wedge b \not\succ_W a \Rightarrow a \#_W b$. □

Example 17. Based on Table 4, after we change some relations of $\twoheadrightarrow_W$ to $\#_W$, we obtain Table 5. Based on the relations in this table, α mining algorithm can build a new SWF-net as shown in Fig. 2.

The time complexity of this stage mainly depends on α mining algorithm. According to [7], the time complexity is exponential in the number of tasks.

4.5 Post Process

After α mining algorithm is applied, there would be some transitions without any input place or without any output place. For the transitions without any input place, it means that those transitions can be executed in parallel from the start. For the transitions without any output place, it means that those transitions can be executed in parallel to the end. To keep the result models as workflow nets, we add a new source place and an invisible transition to fire the parallel executions, and add a new sink place and an invisible transition to synchronize the parallel executions. Since we only need to scan every transition at this stage to check its in-degree and out-degree, the time complexity of this stage is $O(n)$, where n denotes the number of transitions in a model.

TABLE 5
Final Task Relations for the Model in Fig. 1

	A	B	C	D	E	F
A		$\rightarrow_W$	$\#_W$	$\rightarrow_W$	$\#_W$	$\#_W$
B			$\rightarrow_W$	$\|\|_W$	$\|\|_W$	$\#_W$
C				$\|\|_W$	$\|\|_W$	$\rightarrow_W$
D					$\rightarrow_W$	$\#_W$
E						$\rightarrow_W$
F						

4.6 Discussion on Effectiveness

We have explained five steps of our approach in preceding sections, and we have proved some theorems related to the effectiveness of the corresponding steps. In this section, we answer three questions.

1) Can our approach refactor process models with parallel structures at the utmost?
2) Are all data operation dependencies preserved in refactored models?
3) Is there any new data operation dependency introduced in refactored models?

Theorem 11. *Our approach can refactor process models with parallel structures at the utmost.*

Proof. At step 2 (analyze data operation dependency, see Section 4.2), our approach can analyze all causal relations based on data operation dependency correctly and completely. At step 3 (update task relations, see Section 4.3), our approach can find all the causal relations and transitive causal relations that do not exist, and then change them into parallel relations. So based on data operation dependency, all the parallel relations are discovered. In other words, our approach can refactor process models with parallel structures at the utmost. □

Theorem 12. *All data operation dependencies are preserved in refactored models.*

Proof. At step 2 (analyze data operation dependency, see Section 4.2), our approach can analyze all causal relations based on data operation dependency correctly and completely. At step 3 (update task relations, see Section 4.3), our approach preserves all causal relations based on data operation dependency. At step 4 (refactor the model with process mining technology, see Section 4.4), our approach constructs the new model with α mining algorithm. According to α mining algorithm, all the relations of $>_W$ will be presented in the model through places connecting the corresponding two transitions. So all data operation dependencies are preserved in refactored models. □

Theorem 13. *No new data operation dependency will be introduced in refactored models.*

Proof. At step 4 (refactor the model with process mining technology, see Section 4.4), our approach constructs the new model with α mining algorithm. According to α mining algorithm, all the relations of $>_W$ will be presented in the new model through places connecting the corresponding two transitions. On the other hand, all the places in the new model denote relations of $>_W$ between the corresponding two transitions connected by corresponding places. No new relations of $>_W$ will be introduced. So no new data operation dependency will be introduced. □

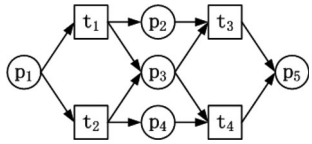

Fig. 4. A model which is not a structured workflow net.

4.7 More Discussion

In this section, we answer two questions: (1) why our approach limits the models to be sound structured workflow nets? (2) can our approach guarantee the refactored models to be sound structured workflow nets?

4.7.1 Why Our Approach Limits the Models to be Sound Structured Workflow Nets?

As we mentioned before, if a WF-net is not sound, there must be some errors in that model and these errors should be eliminated, and SWF-net is more readable and understandable [7]. It is suggested that the business process model should be as structural as possible [9].

On the other hand, our approach depends on α mining algorithm [7], which can only handle sound structured workflow nets well, so our approach limits the models to be sound structured workflow nets.

When we consider unstructured workflow net, for example, the model in Fig. 4. According to Definition 5, since $|p_3 \bullet| > 1$ (p_3 has two outputs) while $|\bullet\, t_3| > 1$ (t_3 has two inputs), this model is an unstructured workflow net. Some theorems cannot hold any more, for example, according to Theorem 2, $t_2 \in \bullet p_3 \wedge t_3 \in p_3 \bullet \Rightarrow t_2 \rightarrow_W t_3$, which means that t_3 can be executed after t_2, however, it is impossible to execute t_3 after t_2, because t_2 and t_3 cannot be executed in the same case. This model can only be executed as (t_1, t_3) and (t_2, t_4).

4.7.2 Can Our Approach Guarantee the Refactored Models to be Sound Structured Workflow Nets?

Since our approach changes some relations between transitions, and based on the new set of relations, α algorithm can construct an unstructured workflow net, our approach cannot guarantee the refactored models to be sound structured workflow nets. For example, given a model as Fig. 5a, where t_2 and t_4 can be executed in parallel, after refactored with our approach, we can get a model as Fig. 5b. In Fig. 5b, since $|p_4 \bullet| > 1$ (p_4 has two outputs), and $|\bullet\, t_5| \neq 1$ (t_5 has two inputs), according to Definition 5, this model is not a SWF-net.

When the refactored models are not sound SWF-nets, we need to let modelers intervene to make those models to be sound SWF-nets and then the models can be used further.

In what scenarios cannot our approach refactor models to be sound SWF-nets? And in these scenarios how can we

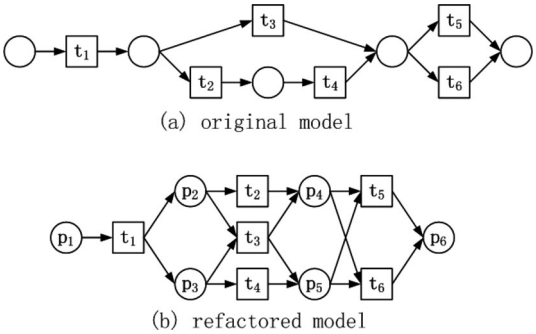

Fig. 5. A refactored model which is not a structured workflow net.

revise the task relations more to get sound SWF-nets? Those problems are under investigation.

Since our primary goal is to refactor models with maximized parallelism, after refactoring the process models can be very different from the original ones. The biggest advantage of this research is that more tasks can be executed in parallel so that the efficiency of business processing can be improved. Our approach points out what task relations have been changed in the step 3, which shows why the refactored models are different with the original ones. Moreover, there are already some works on visualizing the difference between two models such as the work presented in [8].

5 Implementation and Evaluation

In Section 5.1, we describe the implementation of our approach. In Section 5.2, we present the evaluation results to show that our approach can maximize parallel execution of business tasks at a negligible extra effort.

5.1 Implementation

To evaluate our approach, we implement it in BeehiveZ system, which can be accessed at http://code.google.com/p/beehivez/. The screenshot of implementation can be found in Fig. 6, in which the running example in this paper is evaluated.

To store DWF-nets in files, we extend PNML schema[1] with data operations. The part of data operations for the model in Fig. 1 can be found in Fig. 7.

Similar to the refactoring work in the area of software engineering, our approach proposes parallelism refactoring suggestions, and it is up to the user to decide whether to accept or not.

5.2 Evaluation

In this section, we evaluate the effectiveness and efficiency of our approach on randomly generated models. During our experiments, we used a computer with Intel(R) Pentium(R) 4 CPU 3.00 GHz and 2 GB memory. This computer ran Microsoft Windows XP Professional Service Pack 3 and JDK6. The heap memory for JVM was configured as 1 GB.

5.2.1 Settings of Models

We generated 9,800 different DWF-nets randomly. In every model, all the tasks are on a sequential path from the start to the end. According to 7PMG proposed in [9], models should be decomposed if they have more than 50 elements. Hence, the maximum number of transitions per model is configured as 50. Since we want to refactor process models with parallelism, the minimum number of transitions per model is configured as 2. The numbers of models with the number of transitions from 2 to 50 are all configured as 200. We use n to denote the real number of transitions in a model, the maximum number of data items per model is configured as n, and the maximum numbers of tasks for writing and reading one data item are both configured as n.

1. see http://www.pnml.org/index.php for details

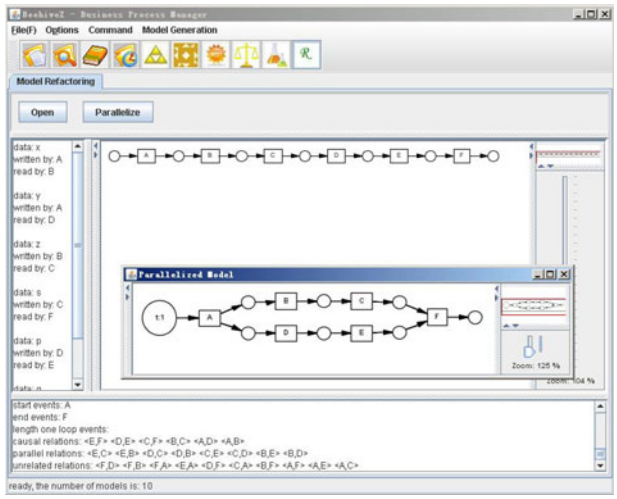

Fig. 6. The screenshot of our implementation.

5.2.2 Effectiveness

Since the effectiveness of our approach has been proved in Section 4, in this section, we show the effectiveness of our approach through experiments. When business tasks can be executed in parallel as much as possible, the efficiency of the corresponding business processing can be improved accordingly. Hence, in this section, we show how our approach can maximize the parallel execution of business tasks instead of how the efficiency of business processing can be improved. According to the work in [14], we measure the degree of parallelism as Equation (1), where $d_{out}(t)$ means the out degree of transition t

$$TS = \sum_{d_{out}(t)>1} d_{out}(t) - 1. \quad (1)$$

If $TS = -1$, it means that no tasks can be executed in parallel. For those models with $TS = -1$, we set $TS = 0$ for future computation. For every generated model, before it is refactored, $TS = 0$, because $\forall t \in T(d_{out}(t) = 1)$. After all the generated models are refactored, $Min(TS) = 0$, $Max(TS) = 49$, $Avg(TS) = 3$, $Stdev(TS) = 6.1$, which means that some models are refactored with parallel structures. We can see that our approach really works. In other words, after parallelism refactoring, business tasks can be executed in parallel as much as possible, so that the efficiency of business processing can be improved.

```
<transition id="trans_0">
    <name>
        <text>F</text>
    </name>
</transition>
<transition id="trans_1">
    <name>
        <text>E</text>
    </name>
</transition>
<transition id="trans_2">
    <name>
        <text>D</text>
    </name>
</transition>

<data id="data_0">
    <name>x</name>
    <readby>trans_4</readby>
    <writtenby>trans_5</writtenby>
</data>
<data id="data_1">
    <name>y</name>
    <readby>trans_2</readby>
    <writtenby>trans_5</writtenby>
</data>
<data id="data_2">
    <name>z</name>
    <readby>trans_3</readby>
    <writtenby>trans_4</writtenby>
</data>
<data id="data_3">
    <name>s</name>
    <readby>trans_0</readby>
    <writtenby>trans_3</writtenby>
</data>
<data id="data_4">
    <name>p</name>
    <readby>trans_1</readby>
    <writtenby>trans_2</writtenby>
</data>
<data id="data_5">
    <name>q</name>
    <readby>trans_0</readby>
    <writtenby>trans_1</writtenby>
</data>
</net>
</pnml>
```

Fig. 7. The file snippet with data operations for the model in Fig. 1.

5.2.3 Efficiency

The time complexity has been analyzed in Section 4. In this section, we evaluate the efficiency of our approach through experiments. We record the time for refactoring every generated model. The minimum time is 0.43 ms (we record the time in nanoseconds), the maximum time is 926.26 ms, the average time is 218.72 ms, and the standard deviation is 240.92 ms, which are negligible in general. The average time for models with the same number of transitions can be found in Fig. 8, in which, the different times are calculated as follows. Let m denote a model, and $|m|$ denote the number of transitions in the model m. Let $TC(m)$ denote the time for refactoring the model m with parallel structures. We have the following results:

$$real\_time(n) = \frac{\sum_{|m_i|=n} TC(m_i)}{\sum_{|m_i|=n} 1}. \quad (2)$$

$$O(n^2)\_time(n) = \frac{n^2}{2^2} \times real\_time(2). \quad (3)$$

$$O(n^3)\_time(n) = \frac{n^3}{2^3} \times real\_time(2). \quad (4)$$

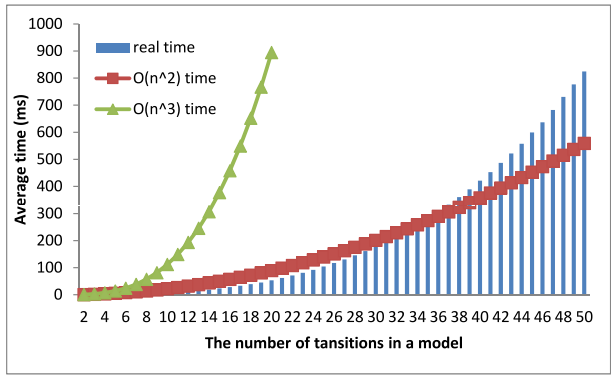

Fig. 8. The average time for refactoring generated models with the same number of transitions.

$real\_time(n)$ calculates the average time for refactoring all the models with the number of transitions as n. $O(n^2)\_time(n)$ predicts the time cost if our refactoring algorithm has a square time to the number of transitions, and $O(n^3)\_time(n)$ predicts the time cost if our refactoring algorithm has a cubic time to the number of transitions. Since the minimum number of transitions in a model is configured as 2, $O(n^2)\_time(n)$ and $O(n^3)\_time(n)$ start with the n as 2. Since the curve of $O(n^3)\_time$ grows fast, we only show part of it. We can see that our approach can refactor randomly generated models quickly, with the time complexity between $O(n^2)$ and $O(n^3)$, where n denotes the number of transitions in a model.

6 Related Work

We review the related work in both areas of software engineering and business process management as follows.

In the area of software engineering, there is some work on parallelism refactoring. The goal is to improve the efficiency of code execution on multi-core processors. For example, in [15], the authors presented a tool that can refactor an array to a ParallelArray. In [16], the author presented the state-of-the-art tools and technologies for parallelism refactoring. In [17], the author presented a toolset supporting parallelism refactoring. In [18], the authors presented a refactoring support for X10 language to make user-selected code in the loop body to run in parallel with other iterations of the loop. In [19], the authors presented a tool that can refactor sequential code into parallel code by using three java.util.concurrent concurrent utilities.

However, on one hand, the technologies existing in the area of software engineering cannot be used to solve the problem proposed in this paper. On the other hand, the idea used in this paper can be used to refactor the existing source codes so that they can be executed in parallel on multi-core CPU or multi machines such as cloud computing platforms. Because source codes can be divided into single-entry-single-exit blocks, and every block can be regarded as a transition in our approach, we can analyze data dependency between different blocks the same way and finally refactor source codes with process mining technology to enable parallel execution.

In the area of business process management, there is also some work on refactoring process models. In [20], the

authors presented a RPST-based refactoring technology to make the model more structural. In [21], the authors summarized the work on refactoring process models. In [22], the authors proposed a technique that can automatically detect four kinds of refactoring opportunities in process models. In [23], the authors proposed an approach to refactor activity labels so that the label quality can be improved. In [10], the authors proposed an approach that can structure acyclic process models. All the work preserves the behavior of process models, and the goal is to make process models more understandable and maintainable. However, our work focuses on improving the efficiency of process model execution, and the behavior of the refactored model is different with the original one, but the data operation dependency is preserved so that the data processing result of business processing is preserved (see Section 4.6).

There have been some works on Project Evaluation and Review Technique (PERT) [24], which is used in project management to analyze and represent the tasks involved in completing a given project, and it is commonly used in conjunction with the critical path method (CPM). For example, PERT can be used to analyze the earliest completion time of some task or the whole project, and also can be used to tell the critical task or critical path so that more attention can be paid to in order to ensure that the project will not be postponed. However, the order between tasks cannot be decided by PERT, and it is the topic of business process model designing, which exists as an independent research domain now. So our work can be used for PERT to decide the order between tasks. In particular, we decide the order between tasks based on data flow analysis. And after the order between tasks is decided, PERT can be used to analyze and manage the project.

There are some work on workflow performance analysis, for example, in [25], the authors proposed a method for computing the lower bound of average turnaround time of transaction instances. In [26], the authors present an analytical method to evaluate the performance of workflow stochastic Petri nets based on block reduction. All these methods can be used to evaluate the performance of refactored models with our method. Our work is different with these works in that our work focuses on how to refactor models to improve the performance.

To the best of our knowledge, we are the first to propose and solve the problem of parallelism refactoring for business process models.

7 Conclusion and Future Work

In this paper, we investigate the problem of parallelism refactoring for business process models for the first time. To solve this problem, we analyze data operation dependency between tasks first, and then refactor some sequence structures to parallel structures. The causal relations and transitive causal relations between tasks without any data operation dependency are changed to parallel relations. Process mining technology is used to construct new models. In a refactored business process model, tasks can be executed in parallel as much as possible, so the efficiency of business processing can be improved. In other words, the quality of process models can be improved. The effectiveness of our approach is proved, and the approach is implemented in an open-source tool. Experiments show that our approach has a quadratic to cubic time complexity in terms of the number of transitions in a model. In a conclusion, our approach can maximize parallel execution of business tasks at a negligible extra effort. Besides the application in the area of business process management, our approach can also be potentially used for software refactoring so that software systems can exploit the power of multi-core processors or computer cluster platforms such as cloud computing platforms.

Given a business process model, based on data operation dependency analysis, some false causal relations and false transitive causal relations are changed into parallel relations, and some false transitive causal relations are changed into causal relations, then a new model is constructed by α mining algorithm. The refactored model may not be a sound SWF-net. In this case, the modelers may need to intervene, which will be further investigated.

Acknowledgments

The work was supported by the HGJ project of China (No. 2010ZX01042-002-002-01), NSF Projects of China (No. 61325008 and No. 61472207), and an Australian Research Council Linkage Project (LP0990393). Tao Jin is the corresponding author.

References

[1] M. Dumas, W. M. P. van der Aalst, and A. H. M. ter Hofstede, *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. New York, NY, USA: Wiley, 2005.
[2] M. Rosemann, "Potential pitfalls of process modeling: Part B," *Bus. Process. Manag. J.*, vol. 12, no. 3, pp. 377–384, 2006.
[3] J. Herbst and D. Karagiannis, "Workflow mining with InWoLvE," *Comput. Ind.*, vol. 53, no. 3, pp. 245–264, 2004.
[4] W. M. P. van der Aalst, "Generic workflow models: How to handle dynamic change and capture management information?" in *Proc. IFCIS Int. Conf. Cooperative Inf. Syst.*, 1999, pp. 115–126.
[5] W. M. P. van der Aalst, "The application of petri nets to workflow management," *J. Circuits, Syst., Comput.*, vol. 8, no. 1, pp. 21–66, 1998.
[6] T. Murata, "Petri Nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[7] W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004.
[8] S. Kriglstein, G. Wallner, and S. Rinderle-Ma, "A visualization approach for difference analysis of process models and instance traffic," in *Proc. 11th Int. Conf. Bus. Process. Manage.*, 2013, pp. 219–226.
[9] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst, "Seven process modeling guidelines (7PMG)," *Inf. Softw. Technol.*, vol. 52, no. 2, pp. 127–136, 2010.
[10] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas, "Structuring acyclic process models," *Inf. Syst.*, vol. 37, no. 6, pp. 518–538, 2012.
[11] H. M. W. E. Verbeek, T. Basten, and W. M. P. van der Aalst, "Diagnosing workflow processes using woflan," *Comput. J.*, vol. 44, no. 4, pp. 246–279, 2001.
[12] N. Sidorova, C. Stahl, and N. Trcka, "Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible," *Inf. Syst.*, vol. 36, no. 7, pp. 1026–1043, 2011.
[13] S. Warshall, "A theorem on Boolean matrices," *J. ACM*, vol. 9, no. 1, pp. 11–12, 1962.
[14] J. Mendling, *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, vol. 6. New York, NY, USA: Springer, 2008.
[15] D. Dig, M. Tarce, C. Radoi, M. Minea, and R. E. Johnson, "Relooper: Refactoring for loop parallelism in Java," in *Proc. 24th ACM SIGPLAN Conf. Companion Object Oriented Programm. Syst. Lang. Appl. Companion*, 2009, pp. 793–794.

[16] D. Dig, "A practical tutorial on refactoring for parallelism," in *Proc. Int. Conf. Softw. Maintenance*, 2010, pp. 1–2.
[17] D. Dig, "A refactoring approach to parallelism," *IEEE Softw.*, vol. 28, no. 1, pp. 17–22, Jan–Feb. 2011.
[18] S. Markstrum, R. M. Fuhrer, and T. D. Millstein, "Towards concurrency refactoring for X10," in *Proc. 14th ACM SIGPLAN Symp. Principles Practice Parallel Programm.*, 2009, pp. 303–304.
[19] D. Dig, J. Marrero, and M. D. Ernst, "Refactoring sequential java code for concurrency via concurrent libraries," in *Proc. 31st Int. Conf. Softw. Eng.*, 2009, pp. 397–407.
[20] J. Vanhatalo, H. Völzer, F. Leymann, and S. Moser, "Automatic workflow graph refactoring and completion," in *Proc. 6th Int. Conf. Service-Oriented Comput.*, 2008, vol. 5364, pp. 100–115.
[21] B. Weber, M. Reichert, J. Mendling, and H. A. Reijers, "Refactoring large process model repositories," *Comput. Ind.*, vol. 62, no. 5, pp. 467–486, 2011.
[22] R. M. Dijkman, B. Gfeller, J. M. Küster, and H. Völzer, "Identifying refactoring opportunities in process model repositories," *Inf. Softw. Technol.*, vol. 53, no. 9, pp. 937–948, 2011.
[23] H. Leopold, S. Smirnov, and J. Mendling, "On the refactoring of activity labels in business process models," *Inf. Syst.*, vol. 37, no. 5, pp. 443–459, 2012.
[24] W. Fazar, "Program evaluation and review technique," *Amer. Statist.*, vol. 13, no. 2, p. 10, Apr. 1959.
[25] J. Li, Y. Fan and M. Zhou, "Performance modeling and analysis of workflow," *IEEE Trans. Syst., Man, Cybern., Part A*, vol. 34, no. 2, pp. 229–242, Mar. 2004.
[26] L. C. Tsironis, D. S. Sfiris, and B. K. Papadopoulos, "Fuzzy performance evaluation of workflow stochastic petri nets by means of block reduction," *IEEE Trans. Syst., Man, Cybern., Part A*, vol. 40, no. 2, pp. 352–362, Mar. 2010.

Tao Jin received the bachelor's degree from the School of Information, Inner Mongolia University of Science and Technology, China, in 2002, the master's degree from the School of Software, Tsinghua University, China, in 2008, and the PhD degree from the Department of Computer Science and Technology, Tsinghua University, China, in 2013, and currently is a postdoctor in the School of Software, Tsinghua University, China. His research focuses on business process model management, including process model retrieval, process model refactoring, process model difference, behavior computing and so on. He leads the development of BeehiveZ system, whose details can be found at http://code.google.com/p/beehivez/.

Jianmin Wang received the graduation degree from Peking University, China, in 1990, and the ME and PhD degrees in computer software from Tsinghua University, China, in 1992 and 1995, respectively. He is currently a professor at the School of Software, Tsinghua University. His research interests include unstructured data management, workflow and BPM technology, benchmark for database system, software watermarking, and mobile digital right management. He has published more than 100 DBLP indexed papers in Journals, such as *TKDE, TSC, DMKD, CII, DKE, FGCS, and IJIIS*, and in conferences, such as SIGMOD, WWW, ICDE, AAAI, IJCAI, ICWS, and SAC. He has led to develop a product data/lifecycle management system, which has been implemented in hundreds enterprises in china. Nowadays, he leads to develop an unstructured data management system, LaUDMS.

Yun Yang received the BSci degree from Anhui University, Hefei, China, in 1984, the MEng degree from the University of Science and Technology of China, Hefei, China, in 1987, and the PhD degree from the University of Queensland, Brisbane, Australia, in 1992, all in computer science. He is currently a full professor in the School of Software and Electrical Engineering at Swinburne University of Technology, Melbourne, Australia. Prior to joining Swinburne as an associate professor, he was a lecturer and a senior lecturer at Deakin University during 1996-1999. Before that, he was a senior research scientist at DSTC Cooperative Research Centre for Distributed Systems Technology during 1993-1996. He was also at Beihang University during 1987-1988. He has coauthored four monographs and published more than 200 papers on journals and refereed conferences. His current research interests include software engineering, cloud computing, workflow systems, big data, and service-oriented computing. He is a senior member of the IEEE.

Lijie Wen received the BS degree in 2000 and the PhD degree in the Department of Computer Science and Technology, Tsinghua University in 2007. He has been an associate professor in the School of Software, Tsinghua University since January 2013. His research interests are business process management technologies, especially on process data management (such as mining, integration, indexing, similarity, retrieval and refinement). He was a postdoc in the Department of Automation, Tsinghua University from 2007 to 2009. He was the organization chair of BPM 2013 as well as its colocated events, i.e., APBPM 2013 and CBPM 2103. By now, he has published more than 70 academic papers on conferences and journals. All his papers have been cited more than 1,000 times by Google Scholar.

Keqin Li is a SUNY Distinguished professor of computer science. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU-GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of things and cyber-physical systems. He has published more than 320 journal articles, book chapters, and refereed conference papers, and has received several best paper awards. He is currently or has served on the editorial boards of *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Computers*, *IEEE Transactions on Cloud Computing*, *Journal of Parallel and Distributed Computing*. He is a fellow of the IEEE.