PRV: An Approach to Process Model Refactoring in Evolving Process-Aware Information Systems

Jin Zeng, Hailong Sun, Xudong Liu, Ting Deng and Jianing Zou
School of Computer Science and Engineering
Beihang University
Beijing, China
zengjin@act.buaa.edu.cn, sunhl@act.buaa.edu.cn, liuxd@buaa.edu.cn, dengting@act.buaa.edu.cn, zoujn@act.buaa.edu.cn

***Abstract*—Process model improvement is one of the most important challenges to deal with in the field of evolving process-aware information systems. In this paper, we present, PRV, an approach to process model refactoring supporting automatic process model improvement, which enables PAIS to adapt to the varying business requirements. First, a set of change operations preserving soundness is defined to avoid complex verification. Second, an algorithm for process model refactoring based on variants is designed to restructure optimal models automatically. Finally, an extensive set of simulations are performed to show the feasibility and effectiveness of the proposed PRV algorithm.**

Keywords-Process-Aware Information Systems, Process Improvenment, Change Operation, Process Model Refactoring

I. INTRODUCTION

In the last decade, PAIS (Process-Aware Information Systems) have become an important type of software systems, which manage and execute operational processes involving people, applications and/or information sources on the basis of process models[1]. As Aalst pointed out in [2], the last trend of information systems is the shift from carefully planned designs to redesign and organic growth. At the same time, PAIS are facing the challenges of constantly varying user requirements and complicated Internet environments. Therefore, PAIS should have the capability of self-adaptation so as to meet those challenges. In response to these needs, evolving PAIS have emerged to achieve the process flexibility[3], which is usually obtained through dynamic process evolution. Generally speaking, the dynamic process evolution should be a continuous process that is a systematic and comprehensive issue facing a number of identified and open technical challenges. According to Yang [4], the main exhibition of dynamic process evolution are changeability of component services, adjustability and configurability of structural relations. And the evolution due to the changing of the structural relation is especially complicated, and it is also the research topic of this work.

Fig. 1 depicts the lifecycle of a typical evolving PAIS on the basis of the PAIS lifecycle[5]. A *process evolution* phase is added between the *process diagnosis* phase and the *process design* phase. According to the feedback of the *process diagnosis* phase, the *process evolution* phase can evolve process models or instances and instruct further re-design in the next lifecycle. According to [6-9], the *process evolution* phase encompasses a set of activities including *process model improvement*, *correctness assurance*, *instance migration*, *evolution validation*, *evolution propagation* and *evolution identification*. In this paper, we are mainly concerned with two of those issues: *process model improvement* and *correctness assurance*.

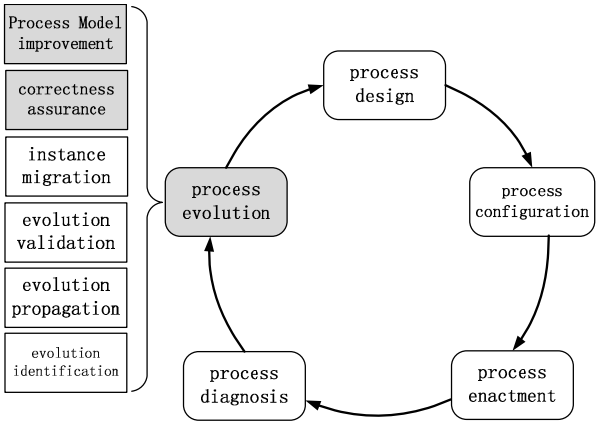

Figure 1. Lifecycle of evolving PAIS

Especially, process model improvement has been identified as one of the fundamental needs for dynamic process evolution in PAIS[6-8, 10, 11]. In traditional PAIS, process models are concrete and invariable to be hardly modified either manually or automatically at runtime. A process model (usually defined by workflow languages such as XPDL and BPEL) exists in the format of a persistent file in practical PAIS. According to user requirements, specific process instances are initiated before execution. However, according to the above analysis, a process instance can be evolved during execution for adapting to users' varying requirements and dynamic Internet environment. And these evolved process instances (usually called *variants*[6]), may not follow the definition of the original process model. On the one hand, this will result in the increase of maintenance costs and the decrease of stability of PAIS; on the other hand, large numbers of variants mean that the original process model cannot capture the evolved business process logic any more, thus it should be replaced with an improved

978-0-7695-4126-6/10 $26.00 © 2010 IEEE
DOI 10.1109/SCC.2010.19

one. However, how to restructure a new and optimal process model basing on the existing process model and related execution information is a critical challenge.

Aiming at process model refactoring, process mining[12] is a potential method. However, the method can only restructure new models from normal execution logs, but it does not deal with execution logs with variants. Some existing researches[10, 11] suggest restructuring process models based on variants. The effectiveness of a process model can be observed through its behaviors (*execution paths*) and frequencies of its variants. If a better process model can automatically be generated based on variants, PAIS can have self-optimized capability, which is the purpose of dynamic evolution. Currently, the works[10, 11] think variants should possess concrete process structures, and restructure new process models using structural similarity or order matrix clustering. Unfortunately, these methods can not precisely measure the difference between a process model and its variants due to high computational complexity.

Another important issue is the correctness guarantee of restructured process models. For process models, the most important correctness criterion is structural soundness (or *soundness* for short)[6]. As arbitrary process change operations[13, 14] without any constraint is dangerous, we need to consider the correctness of process model evolution. Especially, process model improvement is an instant and online process, for a complex business process model it may be intractable to verify the soundness. Especially under the scenarios of online process evolution, it is difficult to realize real-time verification.

This paper presents a complete and practical approach to automatic process model refactoring based on variants in evolving PAIS. First, a set of change operations preserving soundness is defined to avoid complex verification. And we also prove that the soundness of new process model can be satisfied after using the change operations. Second, we present an algorithm of process model refactoring based on variants (PRV for short). The algorithm can generate new cost-effective process models by analyzing *execution paths* and frequencies of variants, to enable PAIS to adapt to the varying business requirements better. Finally, an extensive set of simulations are performed to show the feasibility and effectiveness of the proposed PRV algorithm.

Major contributions of this paper are as follows:

- We define a set of change operations preserving soundness to guarantee the structural soundness of the evolved process models.
- We propose a novel measuring method based on edit distance to evaluate the difference between a process model and its variants.
- We present an algorithm of process model refactoring based on variants, and we design an extensive set of simulations to evaluate the proposed algorithm.

The rest of the paper is organized as follows: In Section II, we introduce the preliminaries of this paper. We present a motivating scenario to explain process model improvement problem in Section III. A set of change operations preserving structural soundness for process model evolution is given in Section IV. In Section V, an algorithm for process model refactoring based on variants are designed. Section VI describes a case study and experiments. The related work is discussed in Section VII. Finally, we conclude this paper by summarizing the main results and pointing out future work in Section VIII.

II. PRELIMINARIES

To exactly present our solution, PRV, we adopt WF-net[15] to formally describe all process models in this paper. WF-net is a special Petri net and possesses the advantages of formal semantics definition, graphical nature and analysis techniques.

Definition 1 ***(WF-net)***[15]***:*** *A Petri net WFN=(P,T,F) is a WF-net (Workflow net) if and only if:*

- *There is one source place $i \in P$ such that $\bullet i = \Phi$;*
- *There is one sink place $o \in P$ such that $o\bullet = \Phi$; and*
- *Every node $x \in P \bigcup T$ is on a path from i to o;*

It should be noted that a WF-net specifies the structure and behavior of a process model. In WF-net, tasks (or called activities) are represented by transitions, conditions are represented by places, and flow relationships are used to specify the partial ordering of tasks. In addition, the *state* of a WF-net is indicated by the distribution of tokens amongst its places. We use $|P|$ dimension vector M to present the state of a live procedure in WF-net, where every element means the number of tokens in a place. We denote the number of tokens in place p in state M by M_p. Specifically, we use M_0 (M_{end}) to denote initial state (final state) which has only token in source (sink) place. A transition $t \in T$ is enabled in state M iff $M_p>0$ for any place p *such that* $(p,t) \in F$. If t is enabled, t can fire leading to a new state M' such that $M'_p = M_p - 1$ and $M'_{p'} = M_{p'} + 1$ for each $(p,t) \in F$ and $(t,p') \in F$, which is denoted by $M\,[t>M'$.

Definition 2 ***(Sound)*** [15]***:*** *A procedure modeled by a WF-net WFN=(P, T, F, i, o, M_0) is sound if and only if:*

- *For every state M reachable from initial state M_0, there exists a firing sequence leading from state M to final state M_{end}. Formally:*
$$\forall M (M_0 \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} M_{end})$$
- *State M_{end} is the only state reachable from state M_0 with at least one token in place o. Formally:*
$$\forall M (M_0 \xrightarrow{*} M \wedge M \geq M_{end}) \Rightarrow (M = M_{end})$$
- *There are no dead transitions in WFN. Formally:*
$$\forall t \in T \; \exists M, M' \text{ s.t. } M_0 \xrightarrow{*} M \xrightarrow{t} M'$$

Soundness is an important correctness criterion that guarantees proper termination of process instances. The first requirement in Definition 2 means that starting from the

initial state M_0, it is always possible to reach the state with only one token in place *o*. The second requirement states that when a token is put in place *o*, all the other places should be empty. The last requirement states that there are no dead transitions in the initial state M_0. Generally speaking, for a complex WF-net it may be intractable to decide soundness[16]. In this paper we do not verify soundness of an evolved process model or instance, but preserve the one using a basic change operation set (section IV).

According to van der Aalst[6], process evolution includes two types: the evolution of process models is called *evolutionary change* and the evolution of process instances is called *momentary change*. *Momentary change* may introduce *variants* which may be created in instance initialization (called *entry time*) as well as running (called *on-the-fly*). Usually, a variant may be regarded as bias of an instance, that is the instances being executed do not conform to order relationship specified by the process model. In this paper, the emphasis is how to restructure optimal process models based on existing variant information, and we do not focus on the evolution of process instances (i.e. how to derive variants from instances). To explain our PRV farther, execution paths must be defined.

Definition 3 (*Execution Path*). *Let a sound WF-net WFN=(P, T, F, i, o, M_0) be the process structure of a process model, a transition sequence (firing sequence) from initial state M_0 to final state M_{end} is named execution path (denoted by ep), and ep $\in T^*$; in addition, a set containing all of the execution paths is called Log of the model (denoted by L).*

An instance is a normal *execution path* of the model, i.e. this *execution path* belongs to the *Log* of the model. In the same way, a variant can also be regarded as an exceptional *execution path*, instead of a normal *execution path* of the model, i.e. the exceptional *execution path* does not belong to the *Log* of the model.

In addition, all of the discussed process models are modeled by WF-net without any cyclical structures and a task (transition) appears only once in a process model (WFN). We do not consider data flow and data constraints issues in this work.

III. MOTIVATING SCENARIO

In this section, we give a motivating scenario for process model improvement in PAIS.

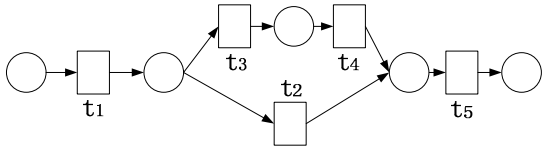

Figure 2. Original process model

Figure 2 illustrates an existing business process model that has two normal execution paths ($L=\{ep_1=t_1t_3t_4t_5, ep_2=t_1t_2t_5\}$). We suppose that the model is an ill process model, which can not satisfy users' varying requirements at runtime, so four different types of variants are produced (see Figure 3).

The execution path of variant 1 is $varep_1=t_1t_2t_6t_5$, which is gotten by adding transition t_6 behind transition t_2. The execution path of variant 2 is $varep_2=t_1t_2t_6t_7$, which is derived by replacing transition t_5 by t_7, besides adding transition t_6 behind transition t_2. Then, the execution path of variant 3 is $varep_3= t_1t_4t_3t_5$, which is introduced by exchanging transition t_3 and t_4. The execution path of variant 4 is $varep4= t_1t_2t_3t_4t_7$, which is gotten by adjusting order relationship between t_2 and t_3t_4. The execution frequency of four types is 20%, 35%, 15% and 30%, respectively. A necessary explanation is that every variant is only an exceptional *execution path*, which has not a persistent process structure. The process structure in Figure 3 is only for the convenience of understanding.

Obviously, the ill process model is not our expectation. Each actual execution on this process model has not followed expectative normal paths, i.e. ep_1 or ep_2. Large numbers of variants result in expansive maintenance costs. This means that the original process model should retire and be replaced by an improved model. Therefore, we need to evolve (restructure) the original process model so as to produce a new optimal process model. The new process model should be more suitable for users' varying requirements, which may produce less variants or more easily derive variants. As mentioned above, the basic of process model evolution is change operations. In next section, we will define a set of change operations preserving soundness.

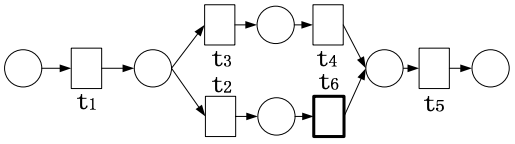

Figure 3a. Variant 1

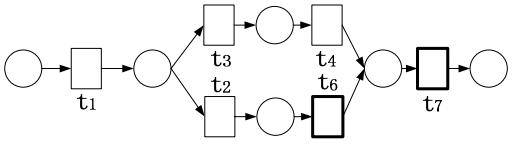

Figure 3b. Variant 2

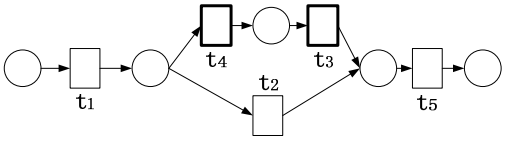

Figure 3c. Variant 3

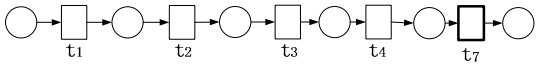

Figure 3d. Variant 4

IV. CHANGE OPERATIONS PRESERVING SOUNDNESS

In the section a set of change operations preserving soundness is defined including replacement, addition, deletion and process structural adjustments to support process model evolution. *Replacement* operation means replacing a sound sub WF-net in a sound WF-net by an other

sound WF-net. (Each dashed box denotes original WF-net and a "cloud" presents a complicated internal structure in the WF-net in Figure 4.)

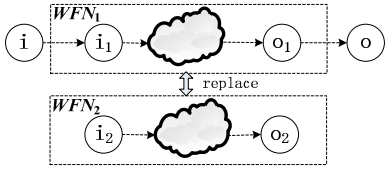

Figure 4. Replacement

Definition 4 ***(Replacement Operation).*** *Let $WFN_1=(P_1,T_1,F_1, i_1,o_1,M_{01})$ be a sound sub net of a sound WF-net $WFN=(P,T,F,i,o,M_0)$ and $WFN_2=(P_2,T_2,F_2,i_2,o_2, M_{02})$ be a sound WF-net such that $P_1\cap P_2=\Phi$, $T_1\cap T_2=\Phi$, $F_1\cap F_2=\Phi$, then $WFN'=(P', T', F', i', o', M_0')$ is the WF-net obtained by replacing WFN_1 by WFN_2, such that $P'=(P\backslash P_1) \cup P_2$, $T'=(T\backslash T_1) \cup T_2$, $F'=(F\backslash F_1) \cup F_2 \cup F''$, where $F''=\{(x,i_2) \in P\times T_2 | (x, i_1) \in F_1 \} \cup \{(o_2, y) \in T_2\times P \mid (o_1, y) \in F_1 \}$, and initial state M_0' is a $|P'|$ dimension vector, where $S\backslash S'$ denotes the difference between set S and S', i.e. the set of elements which is in S and not in S'. (Figure 4)*

We know if a transition in a sound WF-net is replaced by an other sound WF-net, then the resulting WF-net is also sound (Theorem 3 in literature [17]). Our replacement operation is an extension to the result mentioned above, because a sound WF-net behaves like a transition. Obviously, we have the conclusion:

Proposition 1. *Replacement operation can preserve soundness of a sound WF-net.*

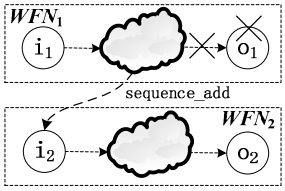

Figure 5a. Sequence_add

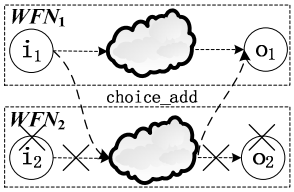

Figure 5c. Choice_add

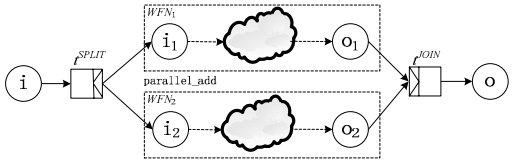

Figure 5b. Parallel_add

Definition 5 ***(Addition Operations).*** *Let $WFN_1=(P_1,T_1,F_1, i_1, o_1, M_{01})$ and $WFN_2=(P_2,T_2,F_2,i_2,o_2,M_{02})$ be two sound WF-nets, such that $P_1\cap P_2=\Phi$, $T_1\cap T_2=\Phi$, $F_1\cap F_2=\Phi$.*

- ***Sequence_add ($WFN_1 \rightarrow WFN_2$)****: $WFN=(P, T, F, i_1, o_2, M_0)$ is the WF-net obtained by sequence_adding WFN_1 with WFN_2, where $P=(P_1\backslash\{o_1\}) \cup P_2, T=T_1 \cup T_2$, $F=\{(x,y) \in F_1 | y\neq o_1\} \cup \{(x, i_2) \in T_2\times P_1 | (x, o_1) \in F_1\} \cup F_2$ and M_0 is a $|P|$ dimension vector (where first element is 1 and rest ones are 0). (Figure 5a)*
- ***Parallel_add ($WFN_1 || WFN_2$)****: $WFN=(P, T, F, i, o, M_0)$ is the WF-net obtained by parallel_adding WFN_1 with WFN_2, where $P= P_1 \cup P_2 \cup \{i, o\}$, $T= T_1 \cup T_2 \cup \{t^{SPLIT}, t^{JOIN}\}$, $F=F_1 \cup F_2 \cup \{< i , t^{SPLIT} >, < t^{SPLIT} , i_1>, < t^{SPLIT} , i_2>, < o_1, t^{JOIN} >, <o_2, t^{JOIN} >, < t^{JOIN} , o>\}$ and M_0 is a $|P|$ dimension vector (where first element is 1 and rest ones are 0). (Figure 5b)*
- ***Choice_add (WFN_1+WFN_2)****: $WFN=(P, T, F, i_1, o_1, M_0)$ is the WF-net obtained by choice_adding WFN_1 with WFN_2, where $P=P_1 \cup (P_2\backslash\{i_2, o_2\})$, $T=T_1 \cup T_2$, $F=F_1 \cup \{(x, y) \in F_2 | x\neq i_2 \wedge y\neq o_2 \} \cup \{(i_1, y) \in P_1\times T_2 | (i_2, y) \in F_2\} \cup \{(x, o_1) \in T_2\times P_1 | (x, o_2) \in F_2\}$ and M_0 is a $|P|$ dimension vector (where first element is 1 and rest ones are 0). (Figure 5c)*

Addition operations show the method of combining two sound WF-nets together. Proposed *Addition* operations are sequential, parallel and choice adding a sound WF-net to another sound WF-net, obviously so obtained new WF-net is sound. Hence we have the conclusion:

Proposition 2. *Addition operations can preserve soundness of a sound WF-net.*

Delete operations are the reverse of addition operations; we do not discuss a series of *delete* operations due to the limitation of paper space. To process evolution, besides above *replacement*, *addition* and *deletion* operations, sometimes we need deal with process structure adjustments. In this paper, five basic *structural adjustment* operations are presented to allow modifying some sound sub nets in a sound WF-net.

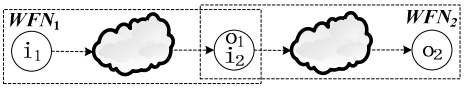

Figure 6a. Original sequence structure of two WF-net

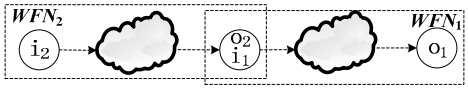

Figure 6b. Reversal sequence adjustment

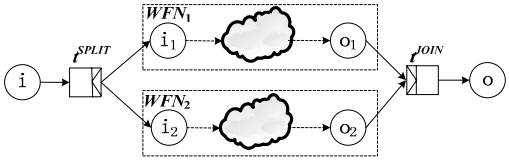

Figure 6c. Sequence_to_parallel adjustment

Figure 6d. Sequence_to_choice adjustment

Definition 6 ***(Sequence Structure Adjustments).*** *Let $WFN=(P, T, F, i, o, M_0)$ be a sound WF-net consisting of $WFN_1=(P_1,T_1,F_1,i_1,o_1, M_{01})$ and $WFN_2=(P_2,T_2,F_2,i_2,o_2,M_{02})$ in sequence, such that $P=P_1 \cup P_2$, $T=T_1 \cup T_2$, $F=F_1 \cup F_2$, $i=i_1$, $o_1=i_2$, $o=o_2$ (Figure 6a).*

- ***Reversal_sequence adjustment****: $WFN'=(P, T, F, i', o', M_0)$ is the WF-net obtained by reversal sequence adjustment between WFN_1 and WFN_2, where $i'=i_2$, $o'=o_1$, $o_2=i_1$; (Figure 6b)*
- ***Sequence-to-parallel adjustment****: $WFN_{||}=(P_{||}, T_{||}, F_{||}, i_{||},o_{||},M_{0||})$ is the WF-net obtained by sequence_to_parallel adjustment between WFN_1 and WFN_2, where $P_{||}= P \cup\{i_{||}, o_{||}\}$, $T_{||}=T \cup\{ t^{SPLIT}, t^{JOIN} \}$, $F_{||}=F \cup\{< i_{||}, t^{SPLIT} >, < t^{SPLIT} , i_1>, < t^{SPLIT} , i_2>, <o_1, t^{JOIN} >, <o_2, t^{JOIN} >, < t^{JOIN}, o_{||}>\}$ and $M_{0||}$ is a $|P_{||}|$ dimension vector (where its first element is 1 and rest ones are 0). (Figure 6c)*
- ***Sequence_to_choice adjustment****: $WFN_+=(P_+,T_+,F_+ i_+, o_+,M_{0+})$ is the WF-net obtained by sequence_to_choice adjustment between WFN_1 and WFN_2, where $P_+=(P\setminus\{i_x, i_y, o_x, o_y\}) \bigcup \{i_+, o_+\}$, $T_+=T$, $F_+=\{(p, q) \in F|p\neq i_1, i_2 \nwedge q\neq o_1, o_2\} \cup\{(i_+, q)|(i_1, q) \in F_1 \vee (i_2, q) \in F_1 \} \cup\{(p, o_+)|(p, o_1) \in F_1 \vee (p, o_2) \in F_2\}$ and M_{0+} is a $|P_+|$ dimension vector (where its first element is 1 and rest ones are 0). (Figure 6d)*

Structure Adjustment operations indicate the reassembly method of two sound sub WF-nets in a sound WF-net. Similarly the "*parallel_to_sequence adjustment*" and "*choice_to_sequence adjustment*" could be defined. We omit them due to the limitation of paper space. Because proposed *structural adjustments* act on two sound sequential sub WF-net, clearly obtained new WF-net is sound. Hence we have the conclusion:

Proposition 3. *Structural adjustment operations can preserve soundness of a sound WF-net.*

V. Process Model Refactoring based on Variants

We know, when user requirements are considered insufficiently in the design phase or the actual operational environment is distinct from developer's expectations, some ill process models emerged which may bring large numbers of variants. Unfortunately, it is expensive to maintain these variants at runtime, so we need a cost-effective approach. In this section we introduce a novel approach to automatically refactoring these ill process models deriving large numbers of variants, which is called process model refactoring based on variants (PRV).

As a basis of PRV algorithm, measuring similarity between a process model and its variants must be concentrated on. As mentioned in Section II, a variant is essentially an exceptional *execution path*. Since variants have no actual process structure, we do not suggest that structural similarity[10] and similarity based on high-level change operations[11] are adopted to measure difference between a variant and its model, which can exactly not compute actual difference. In practice, the structure of a process model is completely equivalent to its *Log*. Namely, the similarity between a variant and its model equals the similarity between the *execution path* of the variant and the *Log* of its model. For measuring difference between a variant and its model exactly, we introduce concept of edit distance[18, 19]. Edit distance is commonly used to solve string-to-string correction problem. Given strings A and B, a set of "edit operations" taking strings into strings, and a set of weights for those operations, find a sequence of edit operations S such that the sum of the weights of the edit operations is minimal and S converts A to B. The following four edit operations and their weights were considered: (1) insert a character (weight W_I); (2) delete a character (weight W_D); (3) change a character into any other character (weight W_C); and (4) interchange any two adjacent characters (weight W_S). For the sake of simplifying computation, weight for each edit operation is 1 in our work. Thus, we can define *bias distance* of variants and average bias distance of models based on edit distance .

Definition 7 ***(Bias Distance).*** *Given a variant and its model, bias distance of this variant is the minimum of all edit distances between the execution path of the variant and every execution path in Log of the model, i.e.*

$$bd = min\{EditDis(varep, ep_i) \mid ep_i \in L\}$$

where varep is the execution path of this variant, L is the Log of the model and EditDis is function computing edit distance.

Definition 8 ***(Average Bias Distance).*** *Given a model and its all variants, the average bias distances of this model is the sum of all products of bias distance and execution frequency of every variant, i.e.*

$$abd = \sum_{i=1}^{n} f_i \bullet min\{EditDis(varep_i, ep_i) \mid ep_i \in L\}$$

where f_i is the frequency of variant i, $varep_i$ is the execution path of variant i, L is the Log of the model and EditDis is function computing edit distance.

Average bias distance can reflect design quality of a process model, i.e. measurement of difference between design requirements and actual users' requirements at runtime. Obviously, the quality is better when the value is lower. Therefore, we give the problem statement of PRV: *given a process model and its all variants, restructure a new optimized process model based on the set of change operations mentioned above, and enable the average bias distance between the new process model to be as low as possible.*

As far as we know, no applicable approaches can solve the problem directly, so our PRV algorithm is a heuristic search algorithm. First, we find average bias distance of a existing ill process model, which is a source node of PRV algorithm. Second, a candidate set of change operations is used to try to evolve the structures the process model. All change operations in the candidate set are executed and one

PRV Algorithm [Process model refactoring based on variants]

input:

$WFN=(P, T, F, i, o, M_0)$,
$InsSet=\{<insep_i, f_i> \mid i=1\ldots m\}$,
$VarSet=\{<varep_i, f_i, ops_i> \mid i=1\ldots n\}$

output:

WFN'
abd'

1 **begin**
2 L:=getLog(WFN)
3 $abd:=\sum_{i=0}^{n} f_i \bullet min\{EditDis(varep_i, p_j) \mid p_j \in L\}$
4 $canOpsSet:=ops_1 \cup \ldots \cup ops_n + \Delta Ops$
5 $WFN':=WFN$
6 $abd':=abd$
7 $epSet:=\{ep \mid <ep, x> \in InsSet$ or $<ep, x, y> \in VarSet\}$
8 **do**
9 { **for each** $op \in canOpsSet$
10 { WFN^{temp}:=Evolve(WFN', op)
11 L^{temp}:=getLog(WFN^{temp})
12 $abd^{temp}:=\sum_{i=0}^{m+n} f_i \bullet min\{EditDis(ep_i, p_j) \mid p_j \in L^{temp}\}$
13 TempList.add(< abd^{temp}, WFN^{temp}>) }
14 abd^{min}:=min(TempList.getabd)
15 **if** $abd^{min} < abd'$
16 { abd':= abd^{min}
17 WFN':= TempList(abd^{min}).getWFN }
18 **else**
19 { **return** WFN'
20 **return** abd'
21 **end** }
22 } **while**

change operation is only executed each time. In this way, a new set of process models will be generated. Third, average bias distance of each new process model in the set is computed and a special process model is selected, average bias distance of which is minimum. Finally, the process model is regarded as the new source node. Then the search is iterated step by step, until a optimum process model is found out.

Input of PRV algorithm is the structure (sound WF-net) of a existing process model, its set of instances and set of variants. Each element in the set of instances is a pair, ins*ep* denotes *execution path* of instances, and *f* denotes frequency. Each element in the set of variants is also a triple, *varep* denotes *execution path* of variants, and *f* denotes frequency, and *ops* denotes change operation used when the variant is derived. Output of PRV algorithm is the structure and average bias distance of a new process model. First, a *Log* of the current model must be found out. It is difficult to compute a *Log* of a sound WF-net directly, so we convert it into a *reachability graph*[20]. Then the *Log* can be found out by means of Breadth-First-Search based on the *reachability graph* and the current average bias distance *abd* of the model is get (see lines 2 and 3). A important step is to generate a candidate set of change operations, which will affect execution efficiency of complete algorithm. If this set is too large, it will result in search space explosion. If too small, we will not get optimum result. We think that the candidate set of change operations should include some change operations(ΔOps) in common use, besides all change operations of variants. Obviously, operations ΔOps are depend on actual scenarios and practical experience (see line 4). The key of PRV algorithm is a heuristic search. First, a search source node is assigned (see lines 5-7). And then based on the candidate set of change operations, the search with step length for 1 is executed. One change operation is executed each time to get a list of new potential models (see lines 8-13). Especially when computing average bias distance of every new model, bias distance of instances should be considered, besides bias distance of variants, since the existing instances are equivalent to variants of every new evolved model. The new model with minimum average bias distance is selected in the list. If the value is less than average bias distance of the source node, the source node will be updated by the selected model and the search is iterated. Otherwise the current source node is the target node of this search and the algorithm ends (see lines 14-21). The size of *reachability graph* of WF-net is exponential on the size of WF-net in the worst-case time, so the worst-case complexity of PRV algorithm is also exponential.

VI. CASE STUDY AND EXPERIMENTS

A. Case Study

According to PRV algorithm, it is very easy to restructure the process model given in Section III. First of all, bias distances of four types of variants are found, which are 1, 2, 2 and 2 respectively. Since execution frequency of each type is known (20%, 35%, 15% and 30%), the average bias distance of the process model can be worked out, which is 1.8. A new process model (shown in Figure 7) is found by using proposed PRV algorithm. Obviously, new bias distances of four types of variants are 1, 0, 3 and 1 respectively, and average bias distance of the new model is 0.95. The reduction of average bias distance has proved that the new model is more accordant with actual users' requirements and that the system maintenance costs is lower. Therefore, we think the new model is cost-effective.

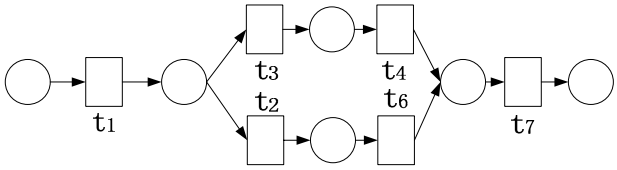

Figure 7. New restructured process model

B. Simulation Experiments

In order to evaluate further actual effectiveness of PRV, we have made simulation experiments. The simulation program is developed with JDK1.5 and runs in a PC (Intel Core 2 Duo CPU 2.99GHz CPU, 4G memory).

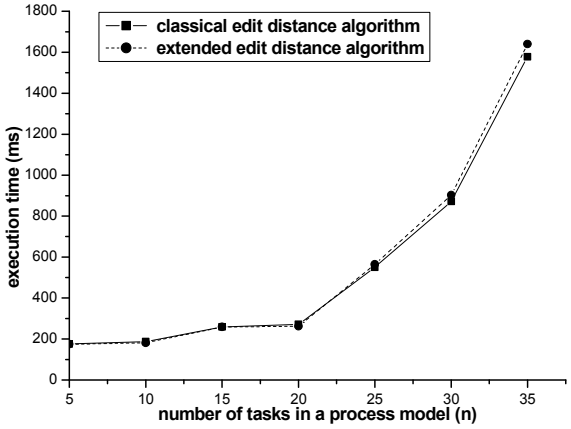

Figure 8. Execution time of PRV

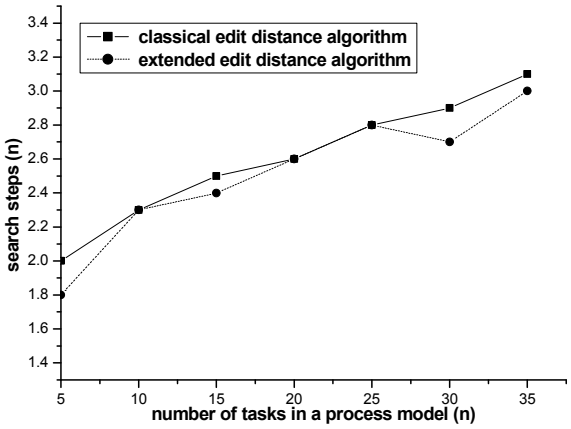

Figure 9. Search steps with PRV

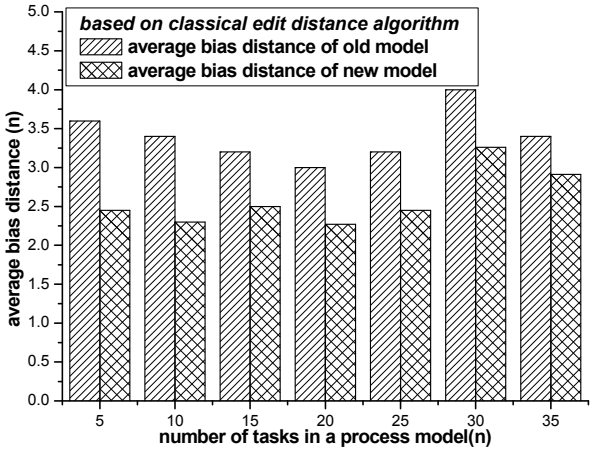

Figure 10. Comparison between average bias distances using classical edit distance algorithm

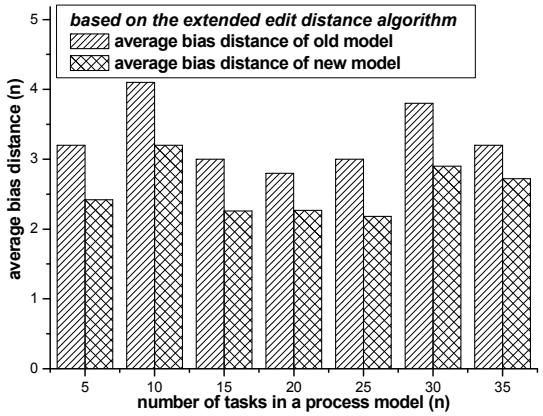

Figure 11. Comparison between average bias distances using extended edit distance algorithm

In these experiments, all the process models are sound and are created at random by using set of building blocks (Section 4.3.3 in the book[15]). Based on the proposed set of change operations, all of the variants are derived after implementing 3-5 changes at random and there are 5 variants each model. Two algorithms for edit distance are adopted to measure average bias distance of process model. The classical algorithm[18] only includes *insert*, *delete* and *change* operations, and yet *interchange* operation is added in extended algorithm[19] besides those operations. By these actual experiments, we know whichever algorithm for edit distance hardly affects PRV algorithm. In addition, we suppose that sum of the frequencies of all variants in each model is 100%. All the experiments are repeated 10 times, and we report their average results. Experiment 1 (figure 8) illuminates relationship between size (the number of tasks) of process model and execution time in proposed PRV algorithm, so the execution time of PRV algorithm is acceptable. Search step length is given in experiments condition above in experiment 2(figure 9), even if the size of a process model reaches 35, PRV algorithm can return a optimum solution through around three steps. The effect of PRV algorithm is shown in experiment 3 and 4 (figure 10 and 11), indicating the average bias distance of a process model can be lowered by PRV algorithm. It is demonstrated that our works can shorten average bias distance of process models and reduce maintenance costs of PAIS effectively.

VII. RELATED WORK

Process model improvement is fundamental issue in the field of dynamic process evolution. Papazoglou et al.[7, 8] thought it should be considered that identification of change operations, version control mechanisms (model improvement) and instance migration mechanisms and so on. Aalst et al.[6] gave a survey and analyzed the problems brought by evolution time, evolution impact, evolution operation categorizing, evolution management and dealing with existing instances. Especially, the difference between

the evolution of process models (evolutionary change) and the evolution of process instances (momentary change) is discussed. Rinderle et al.[21] discussed the correctness criterion and thought completeness, correctness and change realization are three fundamental issues about business process dynamic change. In [13, 14], the process model evolution methods are given respectively based on change operations and unfortunately correctness assurance of process evolution was not discussed. Lu et al.[10] proposed a process variant repository and its retrieval techniques, also discussed how to discover a preferred work practice (i.e. new process model) through business process variants based on structural similarity. Li et al.[11] provided a heuristic search algorithm supporting the discovery of new reference process models by learning from a collection of block-structured process variants in the proposed order matrix. They thought the distance between a process model and related variants is measured by the number of high-level change operations needed to transform the reference model into the variant. It is thought that a variant should possess a concrete process structure in these works[10, 11], which is different from the understanding of process models and variants in this paper. Unfortunately, these methods can not precisely measure the difference between a process model and its variants due to high computational complexity. We think instances and variants have actually not structural nature, and are only some normal and exceptional execution paths of models. The difference of understanding instances and variants between this paper and mentioned works[10, 11] above caused different approaches to process model refactoring.

VIII. CONCLUSION

In this paper, we introduce a novel approach to process model improvement, i.e. process model refactoring based on variants (PRV) in evolving PAIS. First, a set of change operations preserving soundness is introduced. And we also prove that the soundness of new process can be satisfied after using these change operations. With this, complex verification process is avoided. Second, we introduce PRV algorithm. The algorithm can restructure new cost-effective process models by analyzing execution paths and frequencies of variants, to enable PAIS to adapt to the varying business requirements better. Finally, an extensive set of simulations are performed to show the feasibility and effectiveness of the proposed PRV algorithm. For future work, on uncontrollable and dynamic Internet environments, the change of QoS will cause that process models have to be evolved, so we plan to perform research on QoS-aware dynamic process evolution and provide a comprehensive solution for process model improvement.

ACKNOWLEDGMENT

This work was partly supported by China 863 program under Grant No. 2007AA010301 and partly supported by China 973 program under Grant No. 2005CB321803.

REFERENCES

[1] Aalst, W.M.P.v.d., *Process-Aware Information Systems: Lessons to Be Learned from Process Mining.* LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC), 2009. **2**(Special Issue on Concurrency in Process-aware Information Systems): p. 1-26.

[2] Aalst, W.M.P.v.d., *Process-Aware Information Systems: Design, Enactment and Analysis*, in *Wiley Encyclopedia of Computer Science and Engineering*, B.W. Wah, Editor. 2009, Wiley & Sons. p. 2221-2233.

[3] Reichert, M., S. Rinderle-Ma, and a.P. Dadam, *Flexibility in Process-Aware Information Systems.* LNCS Transactions on Petri Nets and Other Models of Concurrency (ToPNoC), 2009. **2**(Special Issue on Concurrency in Process-aware Information Systems): p. 115-135.

[4] Fu-Qing, Y., *Thinking on the Development of Software Engineering Technology*. Journal of Software, 2005. **16**(1): p. 1-7.

[5] Aalst, W.M.P.v.d., A.H.M.t. Hofstede, and a.M. Weske. *Business Process Management: A Survey*. in *Business Process Management (BPM)*. 2003.

[6] Aalst, W.M.P.v.d. and S. Jablonski, *Dealing with workflow change: identification of issues and solutions.* International Journal of Computer Systems Science & Engineering, 2000. **15**(5): p. 267-276.

[7] Papazoglou, M.P. *The Challenges of Service Evolution*. in *The 20th International Conference on Advanced Informa-tion Systems Engineering (CAiSE)*. 2008.

[8] Andrikopoulos, V., S. Benbernou, and M.P. Papazoglou. *Managing the Evolution of Service Specifications*. in *The 20th International Conference on Advanced Information Systems Engineering (CAiSE)*. 2008.

[9] Weidlich, M., M. Weske, and J. Mendling. *Change Propagation in Process Models using Behavioural Profiles*. in *IEEE International Conference on Services Computing (SCC)*. 2009.

[10] Lu, R. and S. Sadiq. *On the Discovery of Preferred Work Practice Through Business Process Variants*. in *ER*. 2007.

[11] Li, C., M. Reichert, and A. Wombacher. *Discovering Reference Models by Mining Process Variants Using a Heuristic Approach*. in *BPM*. 2009.

[12] Aalst, W.M.P.v.d., A.J.M.M. Weijters, and L. Maruster, *Workflow Mining: Discovering Process Models from Event Logs*. 2003, Queensland University of Technology: Brisbane.

[13] Lee, N., et al. *A Version Management Method for Managing Business Process Changes Based on Version-stamped Business Process Change Patterns*. in *The 3rd Intetnational Conference on Innovative Computing Information*. 2008.

[14] Chaâbane, M.A., et al. *Dealing with Business Process Evolution using Versions*. in *Proceedings of the International Conference on e-Business*. 2008.

[15] Aalst, W.v.d. and K.v. Hee, *Workflow Management Models, Methods, and Systems*. 2002, Massachusetts London, England: The MIT Press Cambridge.

[16] Cheng, A., J. Esparza, and J. Palsberg, *Complexity Results for 1-safe Nets.* Foundations of Software Technology and Theoretical computer Science, 1993. **761**: p. 326-337.

[17] Aalst, W.M.P.v.d., *Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques*, in *Business Process Management: Models, Techniques, and Empirical Studies*. 2000, Berlin: Springer-Verlag. p. 161-183.

[18] Levenshtein, V.I., *Binary codes capable of correcting deletions, insertions and reversals.* Doklady Akademii Nauk SSSR, 1966. **163**(4): p. 707-710.

[19] LOWRANCE, R. and R.A. WAGNER, *An Extension of the String-to-String Correction Problem.* Journal of the Association for Computing Machinery, 1975. **22**(2): p. 177-183.

[20] Ye, X., J. Zhou, and X. Song, *On reachability graphs of Petri nets.* Computers & Electrical Engineering, 2003. **29**(2): p. 263-272.

[21] Rinderle, S., M. Reichert, and P. Dadam, *Correctness criteria for dynamic changes in workflow systems--a survey.* Data & Knowledge Engineering, 2004. **50**: p. 9-34.