The Journal of Systems and Software 124 (2017) 260–273

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Software cybernetics in BPM: Modeling software behavior as feedback for evolution by a novel discovery method based on augmented event logs

Chuanyi Li[a,b], Jidong Ge[a,b,*], Liguo Huang[c], Haiyang Hu[b,d], Budan Wu[b], Hao Hu[a], Bin Luo[a]

[a] State Key Laboratory for Novel Software Technology, Software Institute, Nanjing University, 210093 China
[b] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, 100876 China
[c] Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275-0122 USA
[d] School of Computer, Hangzhou Dianzi University, Hangzhou 310018 China

ARTICLE INFO

Article history:
Received 2 January 2015
Revised 20 January 2016
Accepted 4 March 2016
Available online 14 March 2016

Keywords:
Software cybernetics
Process discovery
Petri nets

ABSTRACT

Business Process Management (BPM) is a quickly developing management theory in recent years. The goal of BPM is to improve corporate performance by managing and optimizing the businesses process in and among enterprises. The goal is easier to achieve with the closed-loop feedback mechanism from business process execution to redesign in BPM life cycle, where the business process itself and the set of activities in BPM are viewed as a controlled object and a controller respectively. In this feedback control system, process mining plays an important role in generating feedback of process execution for redesign. However, the existing discovery methods cannot mine certain special structures from execution logs (e.g., implicit dependency, implicit place and short loops) correctly and their mining efficiencies cannot meet the requirements of online process mining. In this paper, we propose a novel discovery method to overcome these challenges based on a kind of augmented event log that will also bring new research directions for process discovery. A case study is presented for introducing how the mined model can be used in business process evolution. Results of experiments are described to show the improvements of the proposed algorithm compared with others.

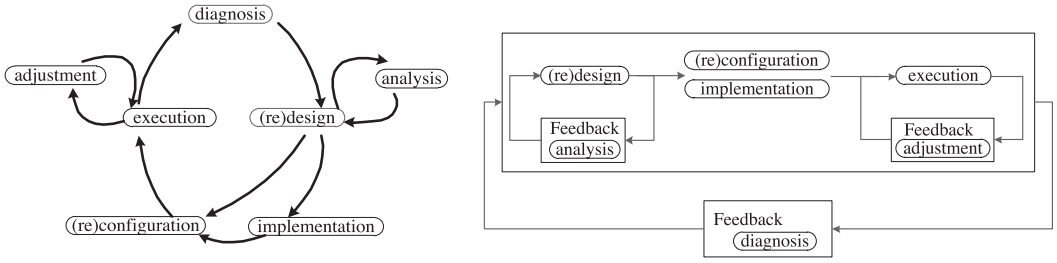
1. Introduction

Cybernetics, which is also interpreted as control theory (He et al., 2007; Peng et al., 2012), creates the general governing principle that controlling the controlled object with a controller and updating the controller according to the feedbacks (Fescioglu and Kokar, 2011) generated from the pre controlling activities. Although certain feedback mechanisms and control activities can always be found in both software system itself and its engineering process, the feedback information is not normally formal and therefore control cannot be achieved precisely (Yang and Zhang, 2014). So, software cybernetics was introduced to address this problem (Cai, 2002). Software cybernetics is motivated to control the software behavior that includes the behavior of software development processes, software maintenance processes, software evolution processes as well as that of software execution itself (Cai et al., 2004). The purpose of control is to make the related software processes more efficient and effective so as to improve the quality of software systems (Ravindran and Rabby, 2013; Ravindran, 2014).

The term 'software' in software cybernetics means software development, maintenance, evolution processes and all related artifacts delivered during such processes, including the delivered software system (Cai et al., 2004). For example, towards software testing (Yang et al., 2014) in software cybernetics, the software under test serves as controlled object, while the software testing strategy serves as the corresponding controller (Cai et al., 2002). The software under test and the corresponding testing strategy make up a closed-loop feedback control system. In this way, the feedback mechanism in software testing is formalized (Cai et al., 2002). Generally, it is the software process itself that controls the software products and improves their quality in software cybernetics. But it can also be said that software cybernetics controls the software processes, because the feedback loop control system supply information for upgrading the process itself. The feedback loops in software cybernetics will help to improve the efficiency of software process in producing high quality products. It provides

* Corresponding author at: Software Institute, Nanjing University, 210093 China. Tel.: +86 25 83621360; fax: +86 25 83621370.
E-mail addresses: gjd@software.nju.edu.cn, gjdnju@163.com, gjd@nju.edu.cn (J. Ge).
URL: http://www.nju.edu.cn (J. Ge)

http://dx.doi.org/10.1016/j.jss.2016.03.013
0164-1212/



(a). BPM life cycle (b). Feedback mechanisms in BPM life cycle

Fig. 1. Interpret *BPM* life cycle with *cybernetics* and connect it to the techniques of *process discovery.*

a clear framework for upgrading the controller's working strategy, such as changing testing strategy in software testing process for finding more potential bugs according to pre testing feedbacks. Feedbacks generated by the controller are keys to accelerate the processes and the more precise the feedbacks are, the more efficient the processes are.

Business process management (BPM) is commonly discussed in optimizing a company's business process. The key concepts in BPM are design, modeling, execution, monitoring, optimization and reengineering (Harmon, 2015; Rosemann and Brocke, 2015) and the process composed by these activities is similar to certain sub processes in software process, such as testing process. Fig. 1(a) shows the process of BPM (i.e., life cycle) (van der Aalst et al., 2003, 2012b). There are three intuitive feedback loops in the process of BPM shown in Fig. 1(b). The first one is getting feedback from design stage for redesign where analysis stage is treated as a method for generating feedback. The next is during the execution stage and there are two different kinds of understanding of feedback mechanisms here. The information derived from executing actions can be viewed as feedback for adjustment and the decision made by adjustment can also be viewed as feedback for making better executing rules. The last one is generating feedback for the evolution process of the controlled business process system through diagnosis activities. Regarding to these feedback mechanisms as well as the similarity between the process of BPM and software processes, software cybernetics theories can be applied to the BPM for improving the business process under management and the management activities themselves. In this paper, we mainly discuss the third feedback mechanism in BPM from the viewpoint of software cybernetics where the business process under management and the set of BPM activities are viewed as controlled object and controller respectively.

The outputs of the execution step usually are system logs and they cannot be used as feedback for the controller directly. In order to make precise and useful feedback for the control system (i.e., the process of BPM), the techniques of process discovery (van der Aalst et al., 2004) are needed for help. Process discovery is to discover software internal execution process and it serves as a modeling tool to derive an equal execution model of real software behavior from the system logs. Then, with the derived execution process model, the compliance between the designed software and the implemented one can be measured easily by comparing them. Process discovery can also benefit software engineering processes by mining software execution process because in the field of software engineering, software execution process modeling provides an intuitive structure of developed software system during software development, especially on verification and validation of requirement specifications in the early and late life cycle of software system development (e.g., software testing processes, requirements engineering processes). In this paper, the discussed business processes are modeled with Petri nets because Petri nets are particularly capable of modeling all structures (i.e., sequences, choices, parallelisms and loops) in software execution processes by providing both the graphical notation and well-established mathematical theory.

In this paper, we focus on discovering an execution process model from the event log representing the control flow of the business process. Till now, there are mainly three kinds of discovery algorithms and they are region-based ones (Carmona et al., 2010; Carmona, 2012), genetic ones (van der Aalst et al., 2005; Vázquez-Barreiros et al., 2014; van Eck et al., 2015) and traditional ones (i.e., α(van der Aalst et al., 2004), α^{+}(de Mederios et al, 2004), λ(Wang et al., 2012), α^{++}(Wen et al., 2007), $\alpha^{\#}$(Wen et al., 2010) and β(Wen et al., 2009). The region-based and genetic ones aim at breaking restrictions of traditional ones, such as that the event log should be complete and without noise. But the mining efficiency of these algorithms is evidently lower than the traditional ones. Our approach is inherited from traditional algorithms, so no comparison between our approach and the region-based or genetic ones will be made in this paper. Although there have been many traditional discovery algorithms, there are still many kinds of structures that cannot be discovered, such as *implicit places, implicit dependencies, non-SWF-nets* and short loops (i.e., *one-loop* and *two-loop*) defined in (Wen et al., 2007) and (van der Aalst et al., 2004). To tackle these problems, a novel discovery method whose core is a mining algorithm called λ^{+} based on a type of augmented event logs is proposed in this paper. The essential difference between λ^{+} and λ is that λ^{+} augments the event logs by inserting the *pre tasks* into events. We follow the conventional definitions of the terms widely accepted in the former algorithms including *event log, event* and *task*. The major contributions we made in this paper are:

(1) We propose to use the reasonable augmented event logs with both pre and post tasks in process discovery. The method for generating the new event logs is also presented.
(2) A new mining algorithm called λ^{+} is proposed which is proved to have improved both the efficiency and ability of traditional mining algorithms.
(3) Plug-ins for generating the augmented event logs and for implementing the mining algorithm are developed on top of the existing open source workflow engine and process mining tools.

The remainder of this paper is organized as follows. Section 2 introduces the modeling tools used in this paper, the Petri nets (Reisig, 1985) and Workflow nets (van der Aalst et al., 1998). Section 3 details the challenges in process discovery. In Section 4, we illustrate our new discovery method by defining the augmented event log, analyzing the strategies for making the augmented logs and detailing the λ^{+} mining algorithm, along with a case study. Section 5 evaluates the new discovery method by proving the algorithm's mining ability formally and comparing its mining efficiency with other algorithms. The limitation of the new method is also described in Section 5. Section 6 describes the related works and in Section 7 a further comparison between the Petri Nets and software cybernetics in what they can do and they

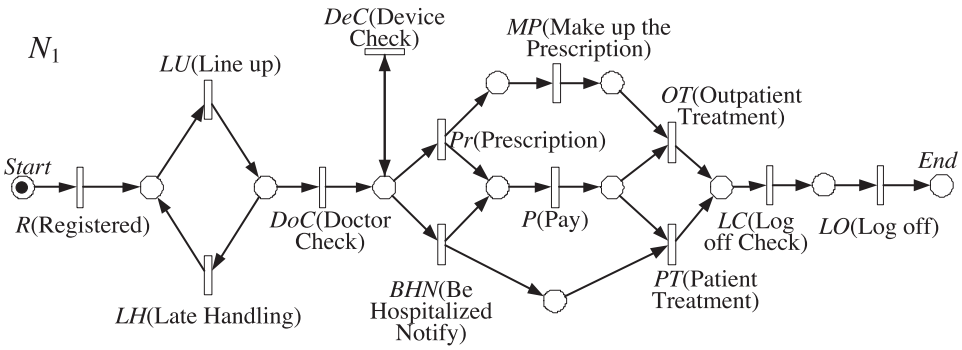


Fig. 2. The process model of the *Patient Registration System* used by many hospitals in China.

cannot do is made. Section 8 concludes and envisages our future work.

2. Modeling language: Petri nets and workflow nets

Petri nets are widely used in modeling and analyzing control systems because they offer both a graphical notation and a well-developed mathematical theory. There are Petri nets for continuous and hybrid discrete-continuous processes that are useful in discrete, continuous and hybrid control theory (David and Alla, 2005). Controllers in cybernetics related researches are often modeled with Petri nets. For example, in software cybernetics, the choice and iteration notations of Petri nets are very flexible in representing feedback-loop mechanism in software requirements or testing engineering. In the context of BPM, Petri nets are used for modeling the feedback loops in the controller and the choice, iteration and concurrency executions of process models. In this paper, Workflow nets (WF-nets), a subclass of Petri nets, are used for modeling the control-flow of business processes supporting software systems. Fig. 2 shows the designed software execution process model of a *Patient Registration System* and it is represented in WF-nets. The control units in this process, such as Late Handling, Device Check and the selection after Doctor Check, are well presented by Petri nets. In the following parts of the paper, an execution process model representing the running behavior of the implemented supporting software system will be discovered through our method for generating precise feedback in BPM from the software cybernetics point of view. This discovering process is used as a case study to show the overall discovery method proposed in this paper.

Petri net PN can be denoted by a 3-tuple, $PN = (P,T,F)$. P is the set of places in the net. T is the set of transitions and $P \cap T = \emptyset$. $F = (P \times T) \cup (T \times P)$ is the set of flow relations between places and transitions. As shown in Fig. 2, a place is represented with a circle and a transition is represented by a rectangle. The set of the pre elements of element $x \in P \cup T$ is denoted by $\bullet x = \{y \in P \cup T | (y, x) \in F\}$ and the set of the post elements of x is denoted by $x\bullet = \{y \in P \cup T | (x, y) \in F\}$. The snapshot of the distribution of tokens (represented by black dots) in the places is called a marking at one point of time and the marking is denoted by M. If the initial marking of the net PN is M_0 then all the possible markings of the net could be denoted by $M = [PN, M_0\rangle$. Let M be the marking of PN at one point, p be a place in PN and t be a transition, then definitions of other terms in Petri nets could be described as follows. The token number in PN under marking M is denoted by $M(p)$. Transition t is enabled under marking M *iff* $\forall p \in \bullet t$: $M(p) \geq 1$. If a transition is enabled it could change its state and this process is called fire. After t fires the marking of PN turns to M' and this transform is marked as $M \xrightarrow{t} M'$. All the enabled transitions under marking M is denoted by *enabled*(M).

In this paper, each transition or task in the Petri nets represents a function unit in the supporting software of the business process. The places serve as caches or storages of transit parameters or resources. A WF-net requires the Petri net to have (i) a single *Start* place, (ii) a single *End* place, and (iii) each node must be on one path from *Start* to *End*. The soundness property further enforces that (iv) there is no dead task, and (v) the process with only one token in the *Start* place can always terminate with only one token in the *End* place. If a WF-net is sound then there will be no dead-lock or live-lock in the model (Rozinat and van der Aalst, 2008). The initial and final states of a WF-net with input place i and output place o are denoted by $[i]$ and $[o]$.

The process in Fig. 2 only contains a part of special structures discussed in this paper, such as *implicit dependency, implicit place* and *one-loop*. All the special structures considered in this paper are defined in Section 3 and the ability of our method to mine them correctly is proved in Sections 4.4 and 5.1.

3. Process discovery challenges

The software discussed in this paper are the ones supporting the business processes which are collections of related, structured activities or tasks that produce a specific service or product. The activities may form simple choice, iteration or concurrent structures. But they may also form complex ones, such as *implicit dependencies, implicit places* and *non-SWF-nets*. It is difficult for the existing algorithms to discover these structures and our method is proposed to handle this problem. Following are definitions of these structures.

Definition 1. (Implicit Dependency (Wen et al., 2007)). Let $PN = (P,T,F)$ be a sound WF-net with input place i and output place o. $\forall a,b \in T$, there is an implicit dependency between a and b *iff*: (1)$a\bullet \cup \bullet b \neq \emptyset$, (2)$\nexists M \in [PN,[i]\rangle \Rightarrow a \in enabled(M) \wedge b \in enabled(M - \bullet a + a\bullet)$, (3)$\exists M \in [PN,[i]\rangle \Rightarrow a \in enabled(M)$, and $\exists M' \in [PN, M - \bullet a + a\bullet \rangle \Rightarrow b \in enabled(M')$.

Definition 2. (Implicit Place (van der Aalst et al., 2004)). Let $PN = (P,T,F)$ be a sound WF-net with an initial state $[i]$, $\forall p \in P$ is an implicit place *iff*: $\forall M \in [PN, [i]\rangle$, $\forall t \in p\bullet \Rightarrow$ *if* $M \geq \bullet t \setminus \{p\}$ then $M \geq \bullet t$.

Definition 3. (SWF-net (van der Aalst et al., 2004)). A WF-net $PN = (P,T,F)$ with initial state $[i]$ is a SWF-net (Structured Workflow Net) *iff*: (1)$\forall p \in P \wedge \forall t \in T \Rightarrow$ *if* $(p,t) \in F \wedge |p\bullet| > 1$ then $|\bullet t| = 1$, (2) $\forall p \in P \wedge \forall t \in T \Rightarrow$ *if* $(p,t) \in F \wedge |\bullet t| > 1$ then $|\bullet p| = 1$, (3) there are no implicit places.

To summarize, Fig. 3 shows all the sub-structures that influence the mining scope of existing algorithms including the structures of implicit dependency, implicit place and non-SWF-net.

The place p in Fig. 3(e) is an implicit place which would not affect the behavior of the net whether it exists or not. Fig. 3(f) is

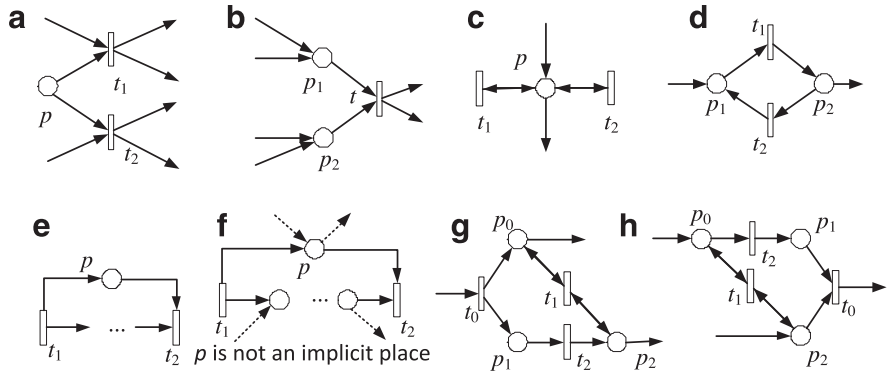


Fig. 3. These are the sub-structures that influence the mining scope of existing algorithms.

Table 1
The conclusion of the mining capability of algorithms α^{++}, β and λ.

Algorithm names	(a), (b)	(e)	(c), (d)	(f)	(g), (h)
α^{++}	Rule out	Cannot	Can	Can	Cannot
β	Rule out	Cannot	Can	Cannot	Not mentioned
λ	Partly	Partly	Can	Can	Cannot

a combined sub-structure of implicit dependency in which place p must be not an implicit place. Implicit places also imply one kind of implicit dependency, so if the algorithm can mine the implicit places correctly then it can mine that kind of implicit dependency correctly. The structure of Fig. 3(f) represents all the patterns of implicit dependency defined in paper (de Mederios et al, 2004). Fig. 3(a), (b) and (e) are typical sub-structures of non-SWF-nets. There are two one-loops in Fig. 3(c) and a two-loop in Fig. 3(d). Fig. 3(g) and (h) stand for a special kind of one length loop together with an implicit dependency. None of the existing algorithms can mine all of these sub-structures correctly. In the remainder parts of this paper, the mining ability of the new algorithm is discussed based on all sub-structures shown in Fig. 3.

Because algorithms β and α^{+} are both inherited from algorithm α and algorithm α^{++} is inherited from α^{+}, only the mining capability of α^{++}, β and λ are concluded in Table 1 according to structures in Fig. 3.

But with the augmented event logs where pre tasks are added, the scope of structures which can be discovered correctly by new mining algorithm λ^{+} is expended (i.e., the mining ability is improved through our new discovery method). Compared with other algorithms, the new λ^{+} algorithm is much more outstanding due to its ability of mining all the structures in Fig. 3. Here is the definition of the mining scope of λ^{+}:

Definition 4. (Mining Scope of λ^{+}). Let $PN = (P,T,F)$ be a sound WF-net, then λ^{+} can mine PN from its event logs with pre and post tasks correctly even though it contains all the sub-structures which are shown in Fig. 3.

The mining ability of λ^{+} algorithm defined here is proved by the case study after describing the details of the mining algorithm in Section 4.3 and the formal proofs in Section 5.1.

4. Process discovery approach

This section describes our new discovery method for deriving software execution process models which represents the real behavior of the running systems. Besides discovering the execution process of software system, the proposed method can be applied to discover software development, testing, maintenance or evolution processes. Taking software testing process as an example, the testing process which contains test planning, test cases design, activities of managing bugs and so on, can be derived with our discovery method. With the help of the intuitive testing process model, it is easy to improve the testing process itself. In the context of software cybernetics in BPM, the discovery method serves as a monitor of the running software system. It will help to generate information for mining potential requirements and bugs. This section is organized as follows. First, the basis of the new method, i.e., the augmented input log, is defined. Second, the strategy for augmenting the event logs is analyzed. Then the core of the discovery method, i.e., the mining algorithm, is proposed along with the definitions of the relations between tasks and helper data structures. At last, a case study based on the business process mentioned in Section 2 (i.e., Fig. 2) is described to show how our discovery method helps to generate intuitive software execution process models as feedback in software evolution process.

Table 2
A part of the augmented event log used in λ^{+} with both pre and post tasks.

Pre-tasks	Current-task	Post-tasks	Pre-tasks	Current-task	Post-tasks
	R	*LU*	*LH*	*LU*	*DoC*
R	*LU*	*DoC*	*LU*	*DoC*	*DeC*
LU	*DoC*	*Pr*	*DoC*	*DeC*	*DeC*
DoC	*Pr*	*MP, P*	*DeC*	*DeC*	*DoC*
Pr	*MP*	*OT*	*DeC*	*DoC*	*DoC*
Pr	*P*	*OT*	*DoC*	*DoC*	*BHN*
MP, P	*OT*	*LC*	*DoC*	*BHN*	*P, PT*
OT	*LC*	*LO, LO*	*BHN*	*P*	*PT*
LC, LC	*LO*		*BHN, P*	*PT*	*LC*
	R	*LU*	*PT*	*LC*	*LO, LO*
R	*LU*	*LH*	*LC, LC*	*LO*	
LU	*LH*	*LU*			

4.1. The augmented event logs: adding pre tasks and post tasks

Table 2 shows a part of the augmented event logs of the implemented *Patient Registration System*. The log will be used as the input of the discovery method for deriving software real execution process. The result process will be used as the feedback of the software evolution process from the software cybernetics point of view. Each line in the table represents an event entry. Each event entry consists of *pre-tasks, current-task* and *post-tasks*. Task is the term in business process and here it can be interpreted with *function* in software systems.

Task List is defined to contain the pre and post tasks in an event entry. The same task in the pre tasks list means that the current executed task needs two or more trigger signals or parameters (represented by *token* in WF-nets) transferred from the pre task in order to execute. The same task in the post tasks list means that

the current executed task will produce two or more tokens for the same post task. A task belonging to a Task List is denoted by '∈'. The Task List shall be defined prior to the events because it is a key element in the event. Below are the definitions of Task List, Event and Event Set:

Definition 5. (Task List). Let $PN = (P,T,F)$ be a sound WF-net. $TL = [t_1,t_2,\ldots,t_n]$ is a Task List based on T *iff*: $\forall t_i \in TL \Rightarrow t_i \in T$. $NUM(t,TL)$ denotes the occurrence of t in TL. Let TL_1,TL_2 be two Task Lists based on T, then $TL_1 \neq TL_2$ *iff*: if $(i = 1 j = 2) \vee (i = 2 j = 1)$ then $\exists t \in TL_i \Rightarrow (t \notin TL_j) \vee (t \in TL_j \wedge NUM(t,TL_i) \neq NUM(t,TL_j))$.

Definition 6. (Event). Let $PN = (P,T,F)$ be a sound WF-net and $t_0 \in T$, then entry $e = [TL_1]t_0[TL_2]$ is an event based on *PN iff:* TL_1 and TL_2 are Task Lists based on T. TL_1 is called the pre Task List and TL_2 is called the post Task List. $\forall e_i = [TL_{11}]t_i[TL_{12}]$, $e_j = [TL_{21}]t_j[TL_{22}] \Rightarrow e_i \neq e_j$ *iff:* $TL_{11} \neq TL_{21} \vee t_i \neq t_j \vee TL_{12} \neq TL_{22}$. $\forall e = [TL_1]t_e[TL_2] \Rightarrow pre(e) = \{t | t \in TL_1\}$, $task(e) = t_e$, $post(e) = \{t | t \in TL_2\}$.

Definition 7. (Event Set). Let $PN = (P,T,F)$ be a sound WF-net, then $ES = \{e_1,e_2,\ldots,e_n\}$ is called the event set based on *PN iff*: (1) $\forall e_i,e_j \in ES \Rightarrow e_i \neq e_j$ (2) $\forall e = [TL_1]t[TL_2] \in ES \Rightarrow t \in T \wedge TL_1,TL_2 \in T^*$.

For example, the event entry $e = [OT]LC[LO,LO]$ in Table 2 has two *LO*s in post task list. That means if task *LC* is executed, there would be two different tokens produced and placed in the post places of *LC*. The event set of the log shown in Table 1 can be formally represented as $ES = \{[]R\ [LU],[R]LU[DoC],[LU]DoC[Pr],[DoC]Pr[MP,P],[Pr]MP[OT],[Pr]P[OT],[MP,P]OT[LC],[OT]LC[LO,LO],[LC,LC]LO[]\}$. But if the pre or post tasks of an event are the same as the executed task, the pre or post tasks should show up only once. For example, the events $e_1 = [t_0,t_2]t_1[t_1,t_1]$ and $e_2 = [t_1,t_1]t_1[t_1,t_1]$ in the event log of Fig. 3(g) should be simplified as $e_1 = [t_0,t_2]t_1[t_1]$ and $e_2 = [t_1]t_1[t_1]$.

4.2. Strategies for making the augmented event logs

The ultimate goal of BPM is to improve the quality of the managed business process referring to that of software cybernetics. Interpreted with theories of software cybernetics, the quality of the supporting software system represents the quality of the business process itself. Practicing the feedback mechanism in the software evolution process is an effective way to upgrade the controlled software products. So, a fundamental question is raised: how to construct precise feedbacks from running software systems for efficient and effective software evolution? It is a common sense that system logs tell what the running software systems do exactly. Then the question is how to derive execution process model from the system logs. The first challenge is to find useful system logs. As analyzed in Section 3, the logs with the pre and post tasks will be more useful than traditional event logs and there are two ways to generate this kind of augmented event logs. The first is to construct the information of pre and post tasks from basic events defined in (van der Aalst et al., 2004). There are often other data fields in the event besides *case id* and *task name*, such as *variables, operators, inputs* and *outputs*. The relations between tasks can be derived from the relations between data fields belonging to different events in the same running cases. For instance, if in a same running instance, there is a same valued variable used in two different events (e_1, e_2) and the variable is a output of another event (e_3), then the tasks of e_1 and e_2 may both be the post tasks of the task of e_3. There has already been research using data fields in basic event logs to get the relationship between tasks, such as (Borrego and Barba, 2014) and (de Leoni and van der Aalst, 2013). This method is more appropriate for mining execution process models than old software systems that already lives for decades. For generating feedbacks for evolution process of other newer software systems supporting business processes, we recommend using the second strategy.

The second is to track and record the routing (i.e., the producers and consumers) of all kinds of resource used in the software system, such as control signals, parameters and even real-world materials. Then the pre and post function units of the current ones can be derived according to the producers and consumers of resources. For example, if two kinds of different resources are produced by the same task at the same time, then the different consumers of these resources must all be the post tasks of the producer. According to this logging method, there is no doubt that the producer and consumer of the resources passing the traditional defined implicit places are recorded in the log. Then, by constructing places between the resource's producer and consumer, the implicit place is mined. In other words, the traditional implicit places related information will always be exhibited in the augmented event logs if the software system runs normally. So it is the augmented event logs that make sure our new discovery method can mine all implicit places. The software for the previously mentioned *Patient Registration System* has been embedded the function of making the augmented event log according to this strategy. We have also developed a plug-in in PIPE (*Platform Independent Petri Net Editor*, a Java based, open source, graphical tool for drawing and analyzing Petri nets[1]) for the method that tracks resources used in the software systems supporting the business processes for making the augmented event logs.

The interesting question that people might ask is whether it deserves to make this kind of system log in process supporting software systems. First, we argue that the function for making the log by tracking the resources used in the system is easy to implement as analyzed above. Second, the efficiency of diagnosis in the governor system will be improved significantly by discovering models using this kind of system logs comparing to other discovery methods and this contributes a lot to improving the efficiency of the total evolution process. The comparisons will be illustrated in Section 5.2.

4.3. Mining algorithm

With the augmented event logs and its generating methods, here comes the most important part of our new discovery method, i.e., the mining algorithm. In software cybernetics, the key point of improving the software quality in evolution process is constructing precise feedbacks. So, the goal of mining algorithms is to discover all relationships between function units correctly and construct a proper software execution process model by these relations. The derived model will be used in detecting software errors and reengineering of business process supporting software systems. However, generating intuitive and interactive feedback for monitoring and controlling the executing business processes is not the only way in which controlled methods in software cybernetics support process discovery. Here, we stress that the controlled methods optimize the discovery method itself with the help of feedback evaluation. Taking our mining algorithm as an example, it cannot handle all special structures in Fig. 3 at first. But by analyzing the experiments feedbacks, we find the way to enhance its mining ability.

Before illustrating the details of the algorithms, the relations between tasks (i.e., function units in business process supporting software system) shall be defined. There are mainly casual dependency and parallel relationship between tasks. Here are the definitions of relations in the augmented event logs.

Definition 8. (Causal Dependency). Let *ES* be the event set of a sound WF-net $PN = (P,T,F)$ and '$\rightarrow_w$' be the label

[1] PIPE can be downloaded from http://sourceforge.net/projects/pipe2/.

for *causal dependency* between tasks. $\forall a,b\in T$, $a\rightarrow_w b$ *iff* $(\exists e = [\theta_1]a[\theta_2]\in ES\Rightarrow b\in\theta_2)\vee(\exists e = [\theta_1]b[\theta_2]\in ES\Rightarrow a\in\theta_1)$.

Definition 9. (PrePost Parallelism). Let *ES* be the event set of a sound WF-net $PN = (P,T,F)$ and '$||_w$' be the label for *per-post parallelism* between tasks. $\forall a,b\in T$, $a||_w b$ *iff* $\exists e = [\theta_1]t[\theta_2]\in ES\Rightarrow (a\in\theta_1\wedge b\in\theta_1)\vee(a\in\theta_2\wedge b\in\theta_2)$. And $a||_w b\Leftrightarrow b||_w a$.

For example, from the event $e = [MP,P]OT[LC]$ in Table 2, we can derive the causal dependencies $MP\rightarrow_w OT$, $P\rightarrow_w OT$ and $OT\rightarrow_w LC$. As defined in *parallelism*, every two tasks in the pre task list or the post task list is in parallel with each other. As to the event log shown in Table 2, the parallelisms derived are $MP||_w P$, $LO||_w LO$, $LC||_w LC$, $BHN||_w P$ and $P||_w PT$. If tasks a and b satisfy $a||_w b$ then there is no need to specify $a\neq b$ (i.e., the relations $LO||_w LO$ and $LC||_w LC$ in Table 2). But for the events whose pre task, current task and post task are the same task, all the parallel relations related to that task are ignored. For example, since there is $e = [t_1]t_1[t_1]$ in the task set of Fig. 3(g), the parallelism $t_1||_w t_2$ is ignored and it is the same in Fig. 3(h). This property is essential to the proof of the new method's mining ability in Section 5.1. Here are other basic and important definitions that will be used.

Definition 10. ($\in$, first, last). Let $ES = \{e_1,e_2,\ldots,e_n\}$ be the event set of sound WF-net $PN = (P,T,F)$ and TL_1,TL_2 are task lists based on T, then (1)$\forall a\in T,[TL_1]a[TL_2]\in ES$ *iff* $\exists e_i = [TL_1]a[TL_2]\in ES$, (2)$first(ES) = \{task(e)|e\in ES\wedge pre(e) = \emptyset\}$, (3)$last(ES) = \{task(e)|e\in ES\wedge post(e) = \emptyset\}$.

The *first*(*ES*) is the set of tasks without pre tasks and the *last*(*ES*) is the set of tasks without post tasks.

Like other mining algorithms, λ^+ also requires no noise in the event logs (i.e., the logs are correct) and the event logs must contain all the necessary information for mining the execution model correctly (i.e., the logs are complete) (van der Aalst et al., 2004). For example, Table 2 is a correct and complete event log for mining the software execution process of the process model in Fig. 2.

Definition 11. (Completeness of Event Set). Let $PN = (P,T,F)$ be a sound WF-net, the event set $ES = \{e_1,e_2,\ldots,e_n\}$ is called the complete event set of *PN* *iff* (1)$\forall ES'\neq ES\Rightarrow ES'\subseteq ES$, (2) $\forall t\in T$, $\exists e\in ES\Rightarrow task(e) = t$.

With the augmented event logs, the first thing to do before constructing the model is deriving causal dependencies between tasks. The causal dependencies will be stored in a list called *Causal Dependency List*. The list is enclosed in "[]". The causal dependencies derived from pre task lists are captured in the *pre* Causal Dependency List(*PreCL*) and the ones derived from post task lists are captured in the *post* Causal Dependency List(*PostCL*). We define the theorem as:

Theorem 1. *The PreCL and PostCL of the same process model will be the same.*

Proof. Let $PN = (P,T,F)$ be a sound WF-net, $ES = \{e_1,e_2,\ldots,e_n\}$ be the complete event set of *PN*. *PreCL* and *PostCL* are the lists of causal dependencies captured from the relationships between the current task and its pre and post tasks. Assume $\forall a\in T\wedge\forall b\in T\wedge a\rightarrow_w b\in PostCL\ \wedge a\rightarrow_w b\notin PreCL$. $a\rightarrow_w b\in PostCL$ means that $\exists e\in ES\wedge task(e) = a\wedge b\in post(e)\ \wedge a\bullet\cup\bullet b\neq\emptyset$, because the WF-net is sound, that is, there is no dead task in the net. And task b would be executed and recorded in the log, which means $\exists e'\in ES\wedge task(e) = b$, because $a\bullet\cup\bullet b\neq\emptyset$, there must exist the relation that $a\in pre(e')$. Thus $a\rightarrow_w b\in PreCL$ can be deduced from e' and it conflicts with our assumption. So $a\rightarrow_w b\in PostCL\Rightarrow a\rightarrow_w b\in PreCL$ and vice versa. That is to say $a\rightarrow_w b\in PostCL\Leftrightarrow a\rightarrow_w b\in PreCL$. Therefore *PreCL* is the same as *PostCL*. □

The *PostCL* will be used in λ^+. The output of mining algorithms is the set of pairs of pre and post tasks of all places. In former algorithms, Y_w is used to denote the set of pairs of pre and post tasks of all places. The entry in Y_w is called the *Max Pre-Post Task Set Pair* (*MTSP*) in this paper. The tasks are added step by step into the pair of pre and post tasks of places in λ^+ and Y_w is constructed at the end of the algorithm. The intermediate state of *MTSP* is defined first as following.

Table 3
Method *merge*.

Method: **Merge a causal dependency into a TSP**

Input: *PPPara*
TSP : (TS_1,TS_2),$TS_1 = [a_1,a_2,\ldots,a_n]$,$TS_2 = [b_1,b_2,\ldots,b_m]$
$c = (a', b')_0$

Output: *TSP* : (TS_1,TS_2)

```
Begin
  if a'∈TS1∧b'∉TS2
    if ∀i∈[1,m]bi∈TS2⇒(bi, b')∉PPPara
      TS2 = TS2 + b'
      If a'≠b'
        c = (a', b')1
      end if
    end if
  else if a'∉TS1∧b'∈TS2
    if ∀i∈[1,n]ai∈TS1⇒(ai, a')∉PPPara
      TS1 = TS1 + a'
      If a'≠b'
        c = (a', b')1
      end if
    end if
  else if a'∈TS1∧b'∈TS2
    If a'≠b'
      if ∀i∈[1,n]ai∈TS1⇒(ai, a')∉PPPara∧∀i∈[1,m]bi∈TS2⇒(bi, b')∉PPPara
        c = (a', b')1
      end if
    end if
  end if
End
```

Definition 12. (Pre-Post Task Set Pair (*TSP*)). Let $PN = (P,T,F)$ be a sound WF-net and $TS_1,TS_2\subseteq T$, then (TS_1,TS_2) is the *TSP* of place $p(p\in P)$ *iff* $\forall a\in TS_1\wedge\forall b\in TS_2\Rightarrow(a,p)\in F\wedge(b,p)\in F$.

Definition 13. (Max Pre-Post Task Set Pair (*MTSP*)). Let $PN = (P,T,F)$ be a sound WF-net and $TS_1,TS_2\subseteq T$. Then (TS_1,TS_2) is the *MTSP* of place $p(p\in P)$ *iff* (1)$\forall a\in TS_1\wedge\forall b\in TS_2\Rightarrow(a,p)\in F\wedge(b,p)\in F$, (2)$\forall(a,p)\in F\wedge\forall(b,p)\in F\Rightarrow a\in TS_1\wedge b\in TS_2$.

Based on the definition of *MTSP*, $Y_w = \{p_{(TS1,TS2)}|\exists p\in P:\forall a\in TS_1\wedge b\in TS_2\Leftrightarrow(a,p)\in F\ \wedge(p,b)\in F\}$. The key step in λ^+ is to add appropriate tasks into the *TSP*s to generate the *MTSP*s of the places. Whether a task should be added or not is determined based on the parallelism relations between tasks. All the parallelism relations are captured in the *Pre-Post Parallelism Set* (*PPPara*). For instance, the *PPPara* derived from Table 1 is $\{MP||_w P,LC||_w LC,LO||_w LO\}$. The process of adding tasks into *TSP* is accomplished by method *Merge*. The inputs of *Merge* are *TSP* = (TS_1,TS_2), *causal dependency* between task a' and b' and the *PPPara* of the entire process. Each *causal dependency* might be encountered by the method *Merge* for more than once. But once it is added into the *TSP* there is no need to insert it again. This is enabled by the state of *causal dependency*. If the causal dependency between a pair of tasks is inserted into the *TSP*, its state is turned into *1* and *0* otherwise. Table 3 is method *Merge*. *Merge* determines the state of a task pair based on the following Theorem.

Theorem 2. *If different tasks both have the same causal dependency with the same task then they must be connected with the same task by a single place if they are not in parallel with each other.*

Table 4
Method *core*.

Method: **Build up Y_w according to *PostCL* and *PPPara***
Input: $PostCL = [(a_1, b_1)_0,(a_2, b_2)_0,\ldots,(a_i, b_i)_0]$ $PPPara = \{(a_1', b_1')_0,(a_2', b_2')_0,\ldots,(a_j', b_j')_0\}$ $Y_w = \emptyset$
Output: $Y_w = \{(TS_{11}, TS_{12}), (TS_{21}, TS_{22}), \ldots, (TS_{k1}, TS_{k2})\}$

Begin
for each $c = (a, b)_{tag\_c} \in PostCL$ do
if $a \neq b \wedge tag\_c$ is 0
TSP: (TS_1, TS_2), $TS_1 = [a]$, $TS_2 = [b]$;
$tag\_c = 1$;
for each $c' = (a', b')_{tag\_c'} \in PostCL$ from the next of $c = (a, b)_{tag\_c}$ in $PostCL$ do
if tag_c' is 0
$Merge(PPPara, (TS_1, TS_2), c')$
end if
end for
Put (TS_1, TS_2) into Y_w
end if
end for
End

Proof. If *a,b,c* are any three tasks in the same WF-net *PN* with $a \rightarrow_w b, a \rightarrow_w c$, and $a\bullet \cup \bullet b \neq \emptyset \wedge a\bullet \cup \bullet c \neq \emptyset$, then there can be only two legal situations: (1) There is a place connecting *a* and *b* and another place connecting *a* and *c*; (2) There is only one place connecting *a,b* and *c*. If there are two places then $\bullet b \cup \bullet c = \emptyset$ that is to say $b||_w c$. And if there is only one place then $\bullet b \cup \bullet c \neq \emptyset$ that is to say $b||_w c$ is not in the *PPPara* of *PN*. If $b||_w c$ is not in the *PPPara* of *PN*, there is only one place between *a,b* and *c*.

Definition 14. (Mining Algorithm λ^+). Let *W* be the sound WF-net representing the execution process model to be mined, *ES* be the event set of *W*, i_w be the input place and o_w be the output place of *W*, T_w be the task set and $\lambda^+(W)$ be the result process model generated by λ^+. Then λ^+ is defined as following.

1. $T_w = \{t|\exists e \in ES \Rightarrow t = task(e)\}$
2. $T_i = \{t|t \in first(ES)\}$
3. $T_o = \{t|t \in last(ES)\}$
4. $PostCL = [(a,b)||TL_1]t[TL_2] \in ES, a = t \wedge b \in TL_2]$,
   $PPPara = \{(a_i,a_j),(b_k,b_l)||a_1,a_2,\ldots,a_n]t[b_1,b_2,\ldots,b_m] \in ES \wedge i,j \in [1,n] \wedge k, l \in [1,m] \wedge i \neq j, k \neq l\}$
5. Let $Y_w = \emptyset, Y_w = Core(PostCL, PPPara, Y_w)$. (Method Core is defined in Table 4.)
6. $P_w = \{p_{(TS1,TS2)} \ |(TS_1,TS_2) \in Y_w\} \cup \{i_w, o_w\}$
7. $F_w = \{(a,p_{(TS1,TS2)})|(TS_1,TS_2) \in Y_w \wedge a \in TS_1\} \cup \{(p_{(TS1,TS2)},b)|(TS_1,TS_2) \in Y_w \wedge b \in TS_2\} \cup \{(i_w,t)|t \in T_i\} \cup \{(t,o_w)|t \in T_o\}$
8. $\lambda^+ (W) = (P_w, T_w, F_w)$

Steps 1–3 mine T_w, T_i and T_o from *ES* (*Event Set*). Step 4 retrieves *PostCL* and *PPPara* from *ES*. And step 5 is the key step in the algorithm which constructs the set of all *MTSP* (Max Pre-Post Task Set Pair of places) based on the *PostCL* and *PPPara*. The details of step 5 are shown in the method *Core* in Table 4. The inputs of *Core* include an empty set used to store the *MTSP*s, *PostCL*, and *PPPara*. Each *causal dependency* in *PostCL* would be processed in *Core*. If the entry to be added is $c = (a,b)_0$ then the $TSP = (\{a\},\{b\})$ will be generated and all remaining entries in *PostCL* will be iteratively compared with the existing entries in the *TSP* by method *Merge* with the input of $TSP = (\{a\},\{b\})$. Step 6 of λ^+ is to generate the set of places with Y_w and step 7 is to generate the set of flows between places and tasks. In the end the formal workflow net is generated with P_w, T_w and F_w.

4.4. Case study

The process of discovering real execution model of the supporting software system of the business process showing in Fig. 2 will be used as a case study to illustrate how our discovery method helps software evolution in software cybernetics. We firstly make a resource tracking log (Fig. 4(a) shows a part of it) by simulating three patients registering in the hospital with the *Patient Registration System*. Then transform this log into the augmented event log with pre and post tasks shown in Fig. 4(b). Since the discovery method is conducted based on the augmented event logs, the first two steps are the key work in applying our discovery method to find software execution process model. The details of λ^+ algorithm will be shown by generating the model representing the real software behavior from the log in Fig. 4(b). (The efforts used in transforming the logs are taken into account in the efficiency calculating in Section 5.2.)

Based on *ES* constructed from Fig. 4(b), λ^+ mines the execution process model in the following steps:

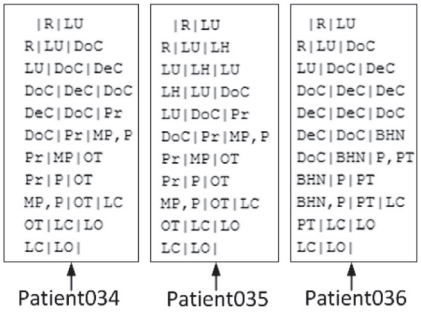
1. $T_w = \{R, LU, LH, DoC, DeC, Pr, MP, P, OT, LC, LO, BHN, PT\}$
2. $T_i = \{R\}$
3. $T_o = \{LO\}$
4. PostCL (i.e., Post Causal Dependency List, the list of causal dependencies derived from the current task and its post tasks in event entries) is $[R \rightarrow_w LU, DoC \rightarrow_w Pr, Pr \rightarrow_w MP, Pr \rightarrow_w P, MP \rightarrow_w OT, P \rightarrow_w OT, OT \rightarrow_w LC, LC \rightarrow_w LO, \ LC \rightarrow_w LO, LH \rightarrow_w LU, LU \rightarrow_w LH, LU \rightarrow_w DoC, DoC \rightarrow_w DeC, DeC \rightarrow_w DeC, DeC \rightarrow_w DoC, DoC \rightarrow_w BHN, BHN \rightarrow_w P, BHN \rightarrow_w PT, P \rightarrow_w PT, PT \rightarrow_w LC]$.
   PPPara (i.e., Pre-Post Parallelism Set, the set of all parallel relationships derived from the event entries) is $\{MP||_w P, P||_w PT, BHN||_w P\}$.
5. $Y_w = \{(\{R,LH\},\{LU\}),(\{LU,DeC,DoC\},\{DoC,LH,DeC,Pr,BHN\}),(\{Pr\},\{MP\}),(\{Pr,BHN\},\{P\}),(\{BHN\},\{PT\}),(\{MP\},\{OT\}),(\{P\},\{OT,PT\}),(\{OT,PT\},\{LC\}),(\{LC\},\{LO\})\}$.
6. P_w is all the places in Y_w adding input and output places.
7. F_w is the set of all flows between t in T_w and p in P_w.
8. $\lambda^+(W) = (P_w, T_w, F_w)$

We have already implemented the λ^+ algorithm as a plug-in in *ProM* (*Process Mining*, the framework for process mining[2]). Fig. 5

[2] ProM is described at http://www.processmining.org/ and can be downloaded from http://www.promtools.org/prom6/.

```
Resource_ID;Resource_Type;Patient_ID;Pre_DateStamp;Pre_Incident;Post_DateStamp;Post_Incident;
RE0008;Control_Sig;Patient034;NONE;NONE;11-09-2014 09:13:34;Registered;
RE0009;Control_Sig;Patient034;11-09-2014 09:13:34;Registered;11-09-2014 09:13:44;Line up;
RE0010;Control_Sig;Patient035;NONE;NONE;11-09-2014 09:13:54;Registered;
RE0011;Control_Sig;Patient035;11-09-2014 09:13:54;Registered;11-09-2014 09:14:04;Line up;
RE0012;Ticket;Patient034;11-09-2014 09:13:44;Line up;11-09-2014 09:14:14;Doctor Check;
RE0013;Doc_Permit;Patient034;11-09-2014 09:14:14;Doctor Check;11-09-2014 09:14:24;Device Check;
RE0014;Control_Sig;Patient035;11-09-2014 09:14:04;Line up;11-09-2014 09:14:34;Late Handling;
RE0015;Control_Sig;Patient036;NONE;NONE;11-09-2014 09:14:44;Registered;
RE0016;Control_Sig;Patient036;11-09-2014 09:14:44;Registered;11-09-2014 09:14:54;Line up;
RE0017;Control_Sig;Patient035;11-09-2014 09:14:34;Late Handling;11-09-2014 09:15:04;Line up;
RE0018;Dec_Result;Patient034;11-09-2014 09:14:24;Device Check;11-09-2014 09:15:14;Doctor Check;
RE0019;Ticket;Patient036;11-09-2014 09:14:54;Line up;11-09-2014 09:15:24;Doc_Check;
RE0020;Doc_Permit;Patient036;11-09-2014 09:15:24;Doctor Check;11-09-2014 09:15:34;Device Check;
RE0021;Ticket;Patient035;11-09-2014 09:15:04;Line up;11-09-2014 09:15:44;Doc_Check;
RE0022;Doc_Decision;Patient034;11-09-2014 09:15:14;Doctor Check;11-09-2014 09:15:54;Prescription
RE0023;Prescription;Patient034;11-09-2014 09:15:54;Prescription;11-09-2014 09:16:04;Make up Pre;
RE0024;Prescription;Patient034;11-09-2014 09:15:54;Prescription;11-09-2014 09:16:14;Pay;
RE0025;Doc_Decision;Patient035;11-09-2014 09:15:44;Doc_Check;11-09-2014 09:16:24;Prescription;
RE0026;Dec_Result;Patient036;11-09-2014 09:15:34;Device Check;11-09-2014 09:16:34;Device Check;
RE0027;Control_Sig;Patient034;11-09-2014 09:16:04;Make up Pre;11-09-2014 09:16:44;Outpatient;
RE0028;Control_Sig;Patient034;11-09-2014 09:16:14;Pay;11-09-2014 09:16:44;Outpatient;
RE0029;Prescription;Patient035;11-09-2014 09:16:24;Prescription;11-09-2014 09:16:54;Make up Pre;
RE0030;Prescription;Patient035;11-09-2014 09:16:24;Prescription;11-09-2014 09:17:04;Pay;
```

(a) Part of the resource tracking log generated in the supporting software of *Patient Registration System*



(b) The augmented event log generated from the resource tracking log

Fig. 4. The logs used in discovering real behavior of the supporting software of *Patient Registration System*.

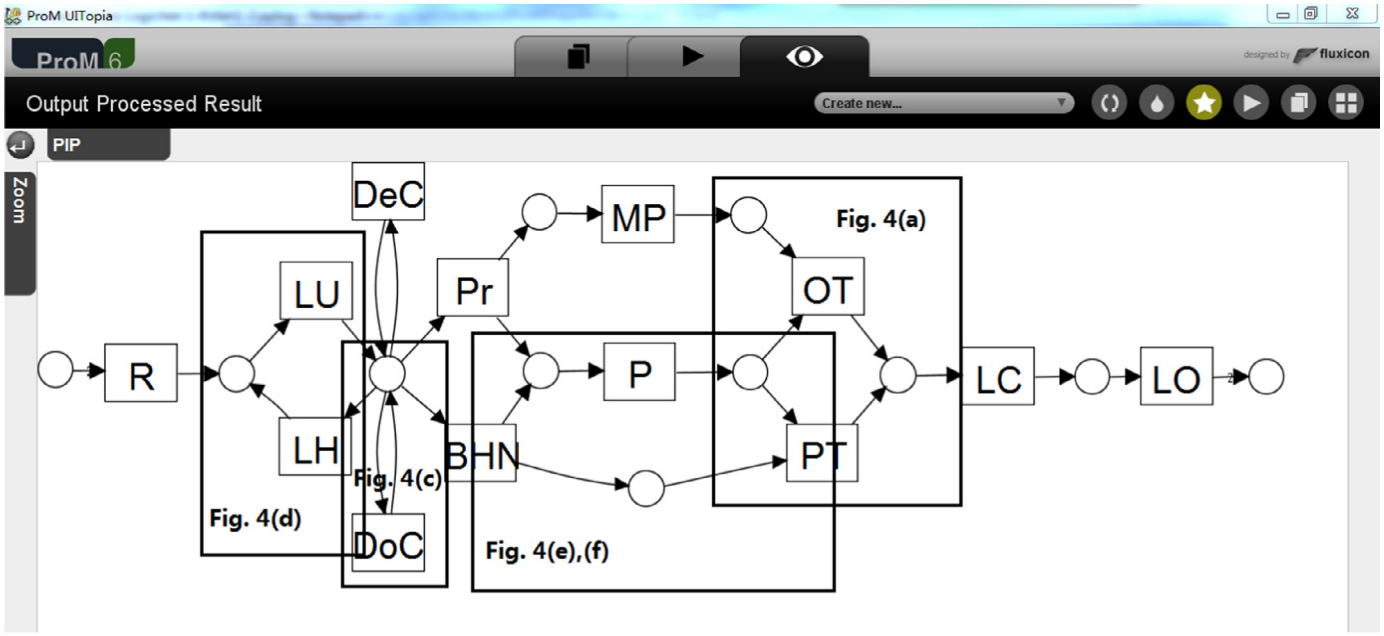


Fig. 5. Result model of handling the log in Table 2 in ProM.

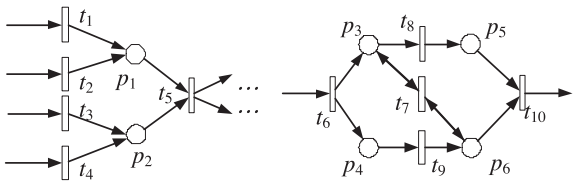


Fig. 6. The result model of extending and merging the sub-structures in Fig. 3(b), (g) and (h).

shows the resulting model mined by *ProM* with the input event log shown in Table 2.

By comparing the mined execution process model in Fig. 5 with the original designed software model shown in Fig. 2, it shows that the model of the implemented process supporting software is not consistent with the original requirements because the task *DoC* is also a one-length loop as the task *DeC*. According to the difference between the pre-defined execution process and the running one, the potential problem of the business process is pointed out. This shows that a complete feedback loop from process discovery to process adjustment is formed in software evolution process from the software cybernetics point of view in the context of BPM. With the precise feedback of real software behavior, the evolution process will be accelerated.

The result shown in Fig. 5 demonstrates the correctness of λ^+ in mining five special (i.e., Fig. 3(a), (c), (d)–(f)) structures in the process model. Next, we will evaluate the algorithm λ^+ by proving the mining ability formally and by comparing the mining efficiency with other algorithms.

5. Evaluations

5.1. Formal proofs of mining ability

In the research of software cybernetics, exploring feedback mechanism is a very important activity to make better reuse of software experience, improve software process as well as the quality of controlled software itself. The accuracy of the feedback is the key factor to measure the quality of the feedback mechanism. For example, in software evolution process discussed in this paper, to what extent the mined execution process model representing the behavior of the running software system determines to what extent the evolution process would benefit the supporting software of the business process. That is to say the mining ability of our proposed discovery method leads a very important role in the evolution process from the software cybernetics point of view. So, we firstly prove that our proposed discovery method can mine all the sub-structures shown in Fig. 3 correctly. Since the case study in Section 4.4 has proved the ability of mining five of them correctly, we only prove the mining ability for other three sub-structures (i.e., Fig. 3(b), (g) and (h)) formally here. Actually, the proofs for different sub-structures are similar to each other. In order to describe clearly, we extend and merge the sub-structures in Fig. 3(b), (g) and (h) as shown in Fig. 6.

Then the problem is to prove the new algorithm can mine place p_1, p_2, p_3, p_4, p_5 and p_6 in Fig. 6. Here is the proof:

Proof. According to the definition of the event logs with pre and post tasks, there must be events:

$e_1 = [t_1,t_3]t_5[\ldots]$,$e_2 = [t_2,t_4]t_5[\ldots]$,$e_3 = [t_1,t_4]t_5[\ldots]$,$e_4 = [t_2,t_3]t_5[\ldots]$,$e_5 = [\ldots]t_6[t_7,t_9]$,$e_6 = [t_6]t_9[t_7]$,$e_7 = [t_6,t_9]t_7[t_7]$,$e_8 = [t_7]t_7[t_7]$,$e_9 = [t_7]t_7[t_8,t_{10}]$,$e_{10} = [t_7]t_8[t_{10}]$,$e_{11} = [t_7,t_8]t_{10}[\ldots]$

Mine p_1 and p_2:

From e_1, e_2, e_3 and e_4, causal dependencies, $t_1 \rightarrow_w t_5$, $t_2 \rightarrow_w t_5$, $t_3 \rightarrow_w t_5$, $t_4 \rightarrow_w t_5$ and parallelisms, $t_1 \|_w t_3$, $t_1 \|_w t_4$, $t_2 \|_w t_3$, $t_2 \|_w t_4$, can be derived. Then $t_1 \rightarrow_w t_5$ can be merged with $t_2 \rightarrow_w t_5$ and $t_3 \rightarrow_w t_5$ can be merged with $t_4 \rightarrow_w t_5$ according to method *Merge* in Table 3. So the *MTSP* (i.e., Max Pre-Post Task Set Pair) of p_1 and p_2 are generated: $(\{t_1,t_2\},\{t_5\})$ and $(\{t_3,t_4\},\{t_5\})$.

Mine p_3, p_4, p_5, and p_6:

According to events e_i, $i \in [5,11]$, there are causal dependencies $t_6 \rightarrow_w t_7$, $t_6 \rightarrow_w t_9$, $t_9 \rightarrow_w t_7$, $t_7 \rightarrow_w t_8$, $t_7 \rightarrow_w t_{10}$, $t_8 \rightarrow_w t_{10}$ and $t_7 \rightarrow_w t_7$, and parallelisms $t_6 \|_w t_9$, $t_8 \|_w t_{10}$, $t_7 \|_w t_8$ and $t_7 \|_w t_9$. But according to the property mentioned after Definition 9, the parallelisms $t_7 \|_w t_8$ and $t_7 \|_w t_9$ need to be ignored since there is an event $e_8 = [t_7]t_7[t_7]$. Then $t_6 \rightarrow_w t_7$, $t_7 \rightarrow_w t_8$ and $t_7 \rightarrow_w t_7$ are merged into the *MTSP* $(\{\{t_6,t_7\},\{t_7,t_8\}\})$ for p_3. $t_9 \rightarrow_w t_7$, $t_7 \rightarrow_w t_{10}$ and $t_7 \rightarrow_w t_7$ are merged into the *MTSP* $(\{\{t_9,t_7\},\{t_7,t_{10}\}\})$ for p_6. $t_6 \rightarrow_w t_9$ and $t_8 \rightarrow_w t_{10}$ are remained for constructing the *MTSPs* $(\{\{t_6\},\{t_9\}\})$ and $(\{t_8\},\{t_{10}\}))$ for p_4 and p_5 separately.

Based on the proofs of algorithm λ^+'s mining ability, it is obvious that λ^+ can handle more sub-structures than other existing mining algorithms (i.e., α, α^+, λ, α^{++}, $\alpha^{\#}$ and β) by using the augmented input event logs in discovering running behavior of software systems for the usage in software evolution in software cybernetics.

5.2. Comparisons of mining efficiency

It was implied in Liu et al. (2006) that the efficiency of online system analysis and evolution in software cybernetics is greatly important. So it is very important to improve the efficiency of software execution process discovery techniques which helps to analyze software systems. Here, we aim at proving that our new method does better than others in mining efficiency.

For comparing the mining efficiency of different mining methods, we have done experiments using PIPE and ProM, where we discover the execution models of the running systems with each method from its corresponding event logs and the executing times are recorded (the experiments were conducted for many times and only the average execution times are used). Among the mining methods based on traditional event logs, the mining efficiency of algorithm α is higher than that of others. Therefore, algorithm α is chosen as a representation of traditional event-log-based mining algorithms in comparison. Since the core of method, i.e., algorithm λ^+, is inherited from λ, the mining efficiency of the method using algorithm λ is also used for comparison. Therefore, the three discovery methods are represented by algorithms α, λ and λ^+.

Firstly, we make a comparison between the methods based on the *Patient Registration System* and the result is shown in Fig. 8(a) (the execution times are recorded in Table 5). Table 6 shows the results of paired *T*-Tests between times pairs used by (α, λ^+) and (λ, λ^+) for proving the statistic significances of the differences. It is obvious that our new method is much more efficient than the others.

In order to prove either the number of tasks or the type of the structures the process has cannot influence the differences between the executing times of the three methods, we also make comparisons based on the models having the same structures (only choices) but different number of tasks (the models in Fig. 7(d), the result in Fig. 8(b)) and the models having the same number of tasks but different structures (i.e., sequence (the model in Fig. 7(a), the result in Fig. 8(c)), parallelisms (the model in Fig. 7(b), the result in Fig. 8(d)), and choices (the model in Fig. 7(c), the result in Fig. 8(e))). We claim that all the used data in the comparison have passed the statistic significance tests.

The result shown in Fig. 8 tells that no matter how many tasks or no matter what kinds of structures the software system contains, the method we propose in this paper discovers that the execution process model from the corresponding system logs is much faster than the other methods. That means our method

Table 5
Executing Times used by for mining *Patient Registration System* with different number of cases.

Method	The execution times of the corresponding number of cases (*ns*)									
	100	200	300	400	500	600	700	800	900	1000
λ^+	2,014,860	2,428,977	3,048,000	3,775,974	4,287,138	5,014,821	5,644,837	6,228,891	6,743,951	7,427,982
λ	76,289,454	79,243,732	84,424,420	90,488,233	97,305,230	107,885,704	119,651,707	129,685,919	142,579,556	159,897,400
α	43,018,485	76,509,473	105,498,484	140,476,442	176,117,768	203,566,483	244,181,743	269,447,660	305,809,976	342,610,001

Table 6
Paired *T*-tests of execution times: (λ^+, α) and (λ^+, λ) on *Patient Registration System*.

Method pairs	Paired differences			*P*-values
	Mean	Std. deviation	Std. error mean	
$\lambda^+-\alpha$	−1.04084E8	26659667.02481	8430526.94601	.000
$\lambda^+-\lambda$	−1.86062E8	98365097.63166	31105775.07809	.000

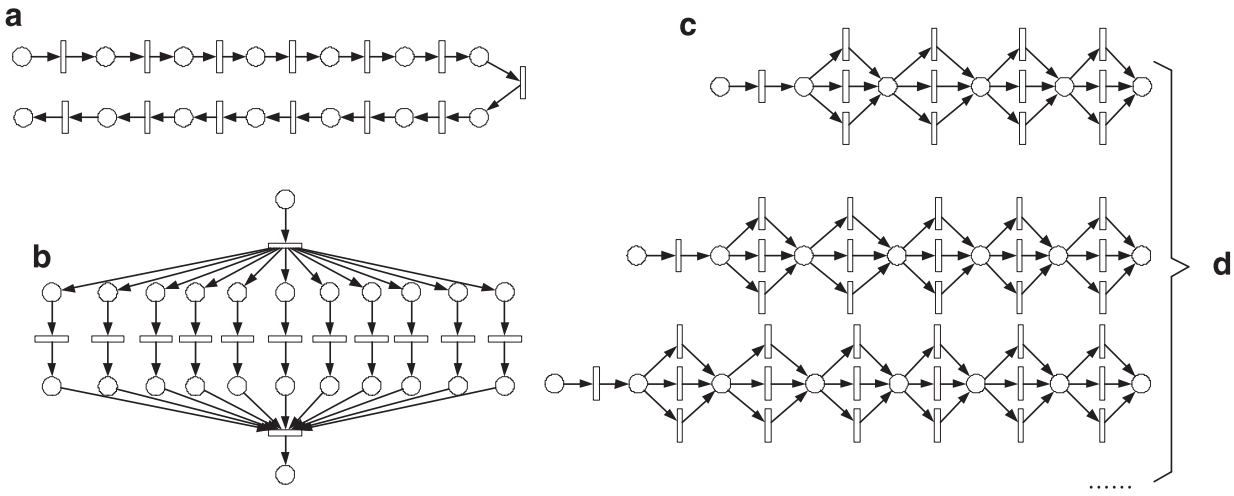
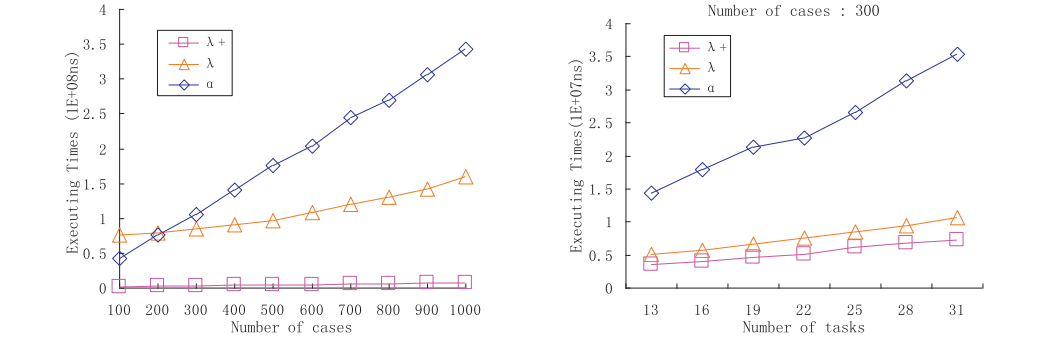
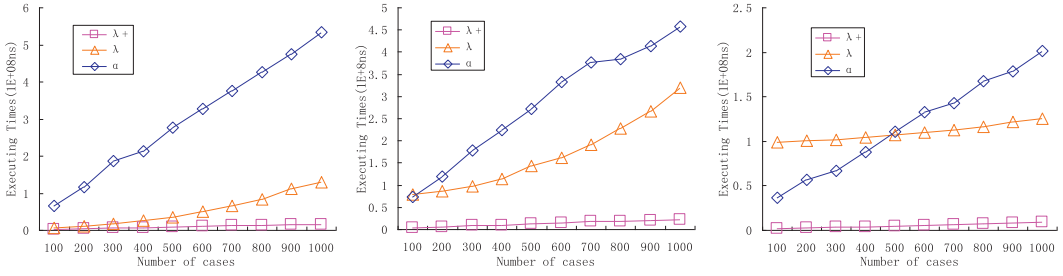


Fig. 7. The models used in the experiment.



(a). Comparison based on *Patient Registration System* (b). Different number of tasks (models of Fig. 7(d))



(c). Sequence (model of Fig. 7(a)) (d). Parallelism (model of Fig. 7(b)) (e). Choice (Fig. 7(c))

Fig. 8. Comparison between three mining methods in handling different models with different number of tasks or different structures.

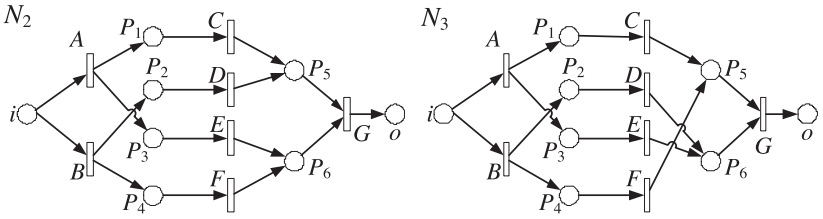


Fig. 9. The possible mined results of λ^+ in handling the logs of N_2.

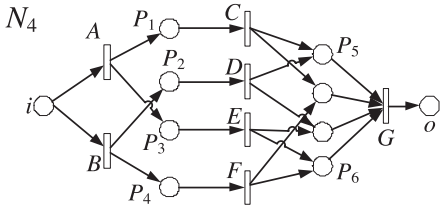


Fig. 10. The discovered result of other mining algorithms in handling the logs of N_2.

is more suitable for generating software behavior feedback for evolution in software cybernetics.

5.3. Limitations

Even though it has been said that the new algorithm can mine almost all structures in software systems whose execution process models can be represented by sound WF-nets correctly, it cannot rediscover N_2 in Fig. 9 accurately since N_3 may also come out as the result, which means if the execution process model of the software contains the following structures, our discovery method may generate inaccurate feedback for its evolution process.

N_3 is sound and its event logs are completely the same as N_2's. The λ^+ algorithm has no idea to distinguish N_3 from N_2 because they are not *Well-structured* WF-nets. The definition of *Well-structured* is:

Definition 15. (Well-handled/Well-structured van der Aalst et al., 1998). Petri net $PN = (P,T,F)$ is well-handled *iff* $\forall x,y \in P \cup T$ and one of them is a place, another is a transition: $\forall C_1,C_2$ are two paths from x to y, if $\alpha(C_1) \cup \alpha(C_2) = \{x, y\}$ then $C_1 = C_2$. And the Workflow net extended from PN is well-structured *iff* PN is well-handled.

The method α here is getting the names of all places and transitions on a path. The definition of Well-structured tells that the *and-split* structure must be followed by an *and-join* and the *or-split* must be followed by an *or-join* in workflow nets.

The limitation of the new mining algorithm λ^+ is that if the model contains the symmetrical non-well-structured subnet, it might fail to rediscover the model. For example, while handling the event logs of N_2, the process model that the new algorithm discovered may be N_2 or N_3 instead the one shown in N_4 (Fig. 10).

6. Related works

Cybernetics and Software Cybernetics: Cybernetics was used by Norbert Wiener as an umbrella term to cover control and communication in the animal and the machine (Norbert, 1948). It is much more about governance than control. A governor should be a server rather than a controller (Filev et al., 2013). In Yamalidou et al. (1996), a method for constructing a Petri net controller for discrete event system modeled by a Petri nets is proposed. To govern a system or an organization, it relies more on the feedback of the system or organization. As a result, what cybernetics strengthens is to make the system or organization much more effective and efficient. In the context of software cybernetics, software problems are treated as a control problem (Cai et al., 2002). So, sub-processes in software process, such as software requirements engineering (Xu et al., 2006; White, 2013) and software testing (Miller et al., 2006), are interpreted from the cybernetics point of view. In (Xu et al., 2006), Requirements Process Control (RPC) is proposed and 66 key practices of Requirements Engineering Good Practice Guide (REGPG) are mapped to different components in the control system. It finds that there are only 5 good RE practices that have been recommended in measuring elements with 18 in actuators and series compensation, 17 in feedback compensation, and 51 in organizational support. In (Miller et al., 2006), the Model Predictive Control (MPC) technique is applied to the software system test phase (STP) with an improved calibration algorithm. The results of the experiments conducted on a STP that lasted 120 days over which 95% defects were discovered and removed show that the proposed control method successfully achieves the schedule and quality objectives of the STP that achieves 90% defect reduction at day 70. In Kenett (2011), it is suggested to expand the original scope of Software Cybernetics and turn it into a natural platform for integrating a wide range of disciplines and methodologies addressing challenges of current and future software systems.

Cybernetics in BPM: Business Process Management (BPM) is a study of supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information (van der Aalst et al., 2003). Cybernetics in BPM concerns how to improve the performance of the process by the feedback of the process. The most common style of feedbacks of the running processes is the system logs and process mining is the technique for making the feedbacks easier to use. Besides, the information generated by process discovery can help to understand the software requirements, design the software architecture and model the entire software system in early software process.

Process Discovery: Process discovery is used for interpreting the system logs into visualized process models (van der Aalst, 2012a; van der Aalst et al., 2012b). It is the basic technique serving cybernetics in BPM. There have been many algorithms for process discovery using event logs made by workflow systems and they are separated into two types: 1). the event logs only contain tasks, such as α, α^+, α^{++}, $\alpha^{\#}$ and β, and 2). the event logs contain the time information, such as adding time constraints on transitions (Li et al., 2004) or on places (Zeng et al., 2008). But these methods have limitations in mining structures in Fig. 3. There are also methods based on the concept of region used in Petri-net (van der Werf et al., 2009). The region-based methods which have the same destination with genetic methods (van der Aalst et al., 2005) aim at breaking restricts on completeness or non-noise of event logs (Carmona et al., 2010) to improve the tolerance of low quality logs. But their mining efficiency is even relatively lower than that of traditional algorithms. Although algorithm λ improves the mining efficiency by adding post tasks into the event logs, it cannot make sure to discover all implicit places and implicit dependencies. Compared with these existing algorithms,

our method makes progresses in (1) extending mining scope to cover all special structures shown in Fig. 3, (2) improving mining efficiency by more than 90% in handling event logs with 1000 running cases and thirteen tasks.

7. Comparison between Petri nets and software cybernetics

A Petri net is a mathematical modeling language for stepwise processes that include choice, iteration and concurrent execution and it was invented by Carl Adam Petri for the purpose of describing chemical processes (Reisig, 1985). There are many extensions of standard Petri nets, such as Colored Petri nets (Jensen, 1997), Object Petri nets (Valk, 2004) and Prioritized Petri nets (Guan et al., 1998).

The motivation of the research of software cybernetics is exploring whether or not and how software behavior can be controlled (Cai et al., 2004). Here software stands for software development process, software maintenance process, software evolution process and all related artifacts delivered during these processes including the developed software system. Cybernetics means the control and communication in the pre-mentioned software including software processes and software systems.

In this section, we mainly try to answer the open question where the connections between Petri nets and software cybernetics are or what they can do and they cannot do. First as a useful concurrency process modeling language, Petri nets can provide basic modeling services (Yu et al., 2014) for processes or software systems concerned in software cybernetics, including structures of sequence, choice, parallelisms and loops. What is more important is the special ability of Petri nets in modeling the system with control conception in software cybernetics. For example, there are specialized Petri nets for usage in control theory, such as the Petri nets used in discrete, continuous and hybrid control theory (David and Alla, 2005). The control systems are separated into two types, the open-loop ones and the closed-loop ones, according to the existence of feedback mechanism. No matter there is feedback or not, Petri nets can do well in modeling the control system. Second, the Petri nets related process theory can be used in analyzing, testing and verifying the benefits brought by software cybernetics to their studied processes or systems. For example, in (Miller et al., 2006), the process model related theory of Model Predictive Control (MPC) was introduced in calculating the impacts of control actions on the software test phase. However, the control theory discussed in software cybernetics is applied to processes and software systems talked about in software engineering, which means Petri nets cannot contribute to software cybernetics with software engineering related theories, such as definitions of software requirements, development, testing, maintenance and evolution processes as well as the good practices which could improve these software processes. For example, Petri nets related theory may tell whether or not or even in what extent a given software practice principle can influence the given software process, but it cannot create any practice principle that can influence software processes. In software cybernetics, the software engineering related process knowledge (e.g., practice principle, target deadline, software quality et al.) should be transformed into parameters which could be utilized by Petri nets analyzing techniques so as to serve the studied software processes better.

As to software cybernetics, the main benefited body is software engineering related studies and the instruction tool is control theory. So basically, what software cybernetics can do is to enhance the quality and efficiency of software engineering by bringing exact control knowledge (Hu et al., 2013) as well as Petri nets theory to the research domain. Further more, considering the characteristics of software system and engineering, research of software cybernetics may help to invent novel control techniques suitable for software processes or systems (Liu et al., 2012; Zhu et al., 2013). For example, there may be closed-loop control with more than one kind of feedbacks or with special feedback analyzing techniques. The discovery method proposed in this paper (i.e., a novel feedback generating technique for software evolution process) can just be viewed as an example of contribution of software cybernetics made to control theory. Till now, most discussed and benefited software processes in software cybernetics are requirements process and testing process. More work should be done in applying control theory to software development, maintenance and evolution processes.

In general, Petri nets serve more as a modeling language and an analyzing or verifying tool in software cybernetics and its main contribution is to cybernetics, while software cybernetics itself can both benefit software engineering and control theory. Considering the connection between Petri nets and software cybernetics, more works on how to apply behavior inheritance theory, soundness checking technique (Liu and Jiang, 2015) and similarity measuring methods (Dijkman et al., 2011) of Petri nets to software cybernetics, how to apply control theory to more software processes and how to develop creative control systems for software engineering research could be conducted.

8. Conclusions and future work

In this paper, we propose a novel software execution process discovery method to model software behavior from system logs for improving the system in the evolution process. From the software cybernetics point of view, the discovery technique is very useful in contributing to the evolution of business processes in the feedback mechanism. Besides, it can also contribute to the early phases in software process, such as requirements, design and modeling. Specifically, the mined execution process model derived from the works discussed in this paper contributes as a more understandable feedback of the running software system. The method we proposed can be split into two steps: (1) generate the augmented event logs with pre and post tasks, and (2) mine a proper execution process model from the augmented event logs. We have proved the feasibility of making the augmented event logs through discussing the generating strategies. Besides the real world *Patient Registration System*, we also developed a plug-in in PIPE for tracking the producers and consumers of resources used in supporting software systems of business processes to validate the feasibility of producing the augmented event logs.

Compared to other existing discovery algorithm, the one we proposed is not only more effective but also more efficient. As for effectiveness, it can mine more special structures (i.e., implicit dependency, non-SWF-nets, one-loop and two-loop) and this is proved through the case study in Section 4.4 and the formal proof in Section 5.1. As for efficiency, while handling the same process of a running system, the time used by our method is almost only one percent of others as showing in Table 5. The results of *T*-tests in Table 6 illustrate that the efficiency of the new method is always significantly higher than those of other methods as the log's capacity increases. Apart from that, the time increasing trends in Fig. 8 also show that no mater what kinds of structures the model has, the more tasks the process has the more time will be saved by our discovery method.

The novel mining method improves the mining efficiency and ability in mining the process's control flow with the help of the augmented event logs. But this is not the only challenge in process mining techniques. There are other challenges not considered in the proposed method, such as mining data flow or resource usage in the process efficiently and promoting the expression of the mined information, etc. However, the use of the augmented event logs will also create new research directions for process mining,

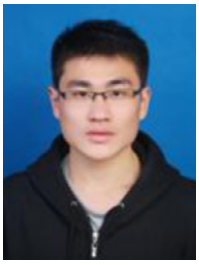
such as whether the new log would help to simplify conformance checking and similarity measuring between process models. According to them, our further research will be done in the following directions: (1) make the discovery method more effective in mining the special structure like the one in Fig. 9, (2) discover more data and resources related information of the process and link them to the control flow model of the execution process for better compliance checking, (3) do more research on how to make the augmented event logs useful in other process mining related techniques, such as conformance checking, similarity measuring between process models and process enhancement.

Acknowledgments

This work was supported by the National 863 Program (2015AA01A203), the National 973 Program (2013CB329102), the National Natural Science Foundation, China (nos. 61572251, 61272188, 61572162, 61100039), the Natural Science Foundation of Jiangsu Province (no. BK20131277), the Open Foundation of State key Laboratory of Networking and Switching Technology (Beijing University of Posts and Telecommunications) (SKLNST-2013-1-14), the Fundamental Research Funds for the Central Universities, the Open Foundation of State Key Laboratory for Novel Software Technology of Nanjing University (KFKT2014B15), U.S. National Science Foundation (NSF Awards CNS-1126747).

References

Borrego, D., Barba, I., 2014. Conformance checking and diagnosis for declarative business process models in data-aware scenarios. Expert Syst. Appl. 41 (11), 5340–5352.
Cai, K.Y., 2002. Optimal software testing and adaptive software testing in the context of software cybernetics. Inf. Softw. Technol. 44 (14), 841–855.
Cai, K.Y., Cangussu, J.W., DeCarlo, R.A., Mathur, A.P., 2004. An overview of software cybernetics. In: Proceedings of the 12th IEEE Annual International Workshop on Software Technology and Engineering Practice, pp. 77–86.
Cai, K.Y., Chen, T.Y., Tse, T.H., 2002. Towards research on software cybernetics. In: Proceedings of the 7th IEEE International Symposium on High-Assurance Systems Engineering (HASE 2002), pp. 240–241.
Carmona, J., 2012. Projection approaches to process mining using region-based techniques. Data Min. Knowl. Discov. 24 (1), 218–246.
Carmona, J., Cortadella, J., Kishinevsky, M., 2010. New region-based algorithms for deriving bounded petri nets. IEEE Trans. Comput. 59 (3), 371–384.
David, R., Alla, H., 2005. Discrete, Continuous, and Hybrid Petri Nets. Springer, Berlin Heidelberg.
de Leoni, M., van der Aalst, W.M.P., 2013. Data-aware process mining: discovering decisions in processes using alignments. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC 2013), pp. 1454–1461.
de Mederios, A.K.A., van Dongen, B.F., van der Aalst, W.M.P., Weijters, T., 2004. Process Mining: Extending the α;-Algorithm to Mine Short Loops. BETA Working Paper Series, WP 113. Eindhoven University of Technology, Eindhoven.
Dijkman, R.M., Dumas, M., van Dongen, B., Käärik, R., Mendling, J., 2011. Similarity of business process models: metrics and evaluation. Inf. Syst. 36 (2), 498–516.
Fescioglu, U.N., Kokar, M.M., 2011. Self controlling tabu search algorithm for the quadratic assignment problem. Comput. Ind. Eng. 60 (2), 310–319.
Filev, D.P, Zhao, Q., Brine, J., 2013. Cybernetics: where shall we go? In: Proceedings of the IEEE International Conference on Cybernetics 2013 (CYBCONF 2013), pp. 25–31.
Guan, S., Yu, H., Yang, J., 1998. A prioritized Petri net model and its application in distributed multimedia systems. IEEE Trans. Comput. 47 (4), 477–481.
Harmon, P., 2015. The scope and evolution of business process management. Handbook on Business Process Management 1, Part of the series International Handbooks on Information Systems. Springer, pp. 37–80.
He, T., John, A.S., Michael, M., Lu, C., Lu, Y., Tarek, A., Sang, S., Tao, G., 2007. Feedback control-based dynamic resource management in distributed real-time systems. J. Syst. Softw. 80 (7), 997–1004.
Hu, H., Jiang, C.H., Cai, K.Y., Wong, W.E., Mathur, A.P., 2013. Enhancing software reliability estimates using modified adaptive testing. Inf. Softw. Technol. 55 (2), 288–300.
Jensen, K., 1997. A brief introduction to coloured Petri nets. Tools and algorithms for the construction and analysis of systems. Lect. Notes Comput. Sci. 1217, 203–208.
Kenett, R.S., 2011. Future directions of software cybernetics: A position paper. In: Proceedings of the IEEE 35th Annual Computer Software and Applications Conference Workshops (COMPSACW 2011), pp. 43–44.
Li, J., Fan, Y., Zhou, M., 2004. Performance modeling and analysis of workflow. IEEE Trans. Syst. Man Cybern. 34 (2), 229–242.
Liu, C., Zhang, W., Zhao, H.Y., Jin, Z., 2012. A problem oriented approach to modeling feedback loops for self-adaptive software systems. In: Proceedings of 19th Asia-Pacific Software Engineering Conference (APSEC), vol. 1, pp. 440–445.
Liu, G.J., Jiang, C.J., 2015. Co-NP-hardness of the soundness problem for asymmetric-choice workflow nets. IEEE T. Syst., Man, Cybern. Syst. 45 (8), 1201–1204.
Liu, Y., Bojan, C., Edgar, F., Sampath, Y., Srikanth, G., 2006. Monitoring techniques for an online neuro-adaptive controller. J. Syst. Softw. 79 (11), 1527–1540.
Miller, S.D., DeCarlo, R.A., Mathur, A.P., Cangussu, J.W., 2006. A control-theoretic approach to the management of the software system test phase. J. Syst. Softw. 79 (11), 1486–1503.
Norbert, W., 1948. Cybernetics: Or Control and Communication in the Animal and the Machine. John Wiley & Sons, Inc., New York; and Hermann & Cie, Paris 1948.
Peng, X., Chen, B., Yu, Y., Zhao, W., 2012. Self-tuning of software systems through dynamic quality tradeoff and value-based feedback control loop. J. Syst. Softw. 85 (12), 2707–2719.
Ravindran, K., 2014. Software cybernetics to manage adaptation behavior of complex network systems. In: Proceedings of 23rd International Conference on Computer Communication and Networks (ICCCN2014), pp. 1–8.
Ravindran, K., Rabby, M., 2013. Software cybernetics to infuse adaptation intelligence in networked systems. In: Proceedings of Fourth International Conference on the Network of the Future (NOF2013), pp. 1–6.
Reisig, W., 1985. Petri Nets: An Introduction. Springer, Verlag New York, Inc. New York, NY, USA.
Rosemann, M., Brocke, J., 2015. The six core elements of business process management. Handbook on Business Process Management 1, Part of the series International Handbooks on Information Systems. Springer, pp. 105–122.
Rozinat, A., van der Aalst, W.M.P., 2008. Conformance checking of process based on monitoring real behavior. Inf. Syst. 33 (1), 64–95.
Valk, R., 2004. Object Petri nets. Lect. Concurr. Petri nets, Lect. Notes Comput. Sci. 3098, 819–848.
van Eck, M.L., Buijs, J.C.A.M., van Dongen, B.F., 2015. Genetic process mining: alignment-based process model mutation. Lect. Notes Bus. Inf. Process. 202, 291–303.
van der Aalst, W.M.P., Alves de Medeiros, A.K., Weijters, A.J.M.M., 2005. Genetic process mining. Lect. Notes Comput. Sci. 3536, 48–69.
van der Aalst, W.M.P., Weijters, T., Maruster, L., 1998. The application of petri nets to workflow management. J. Circ. Syst. Comput. 8 (1), 21–66.
van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M., 2003. Business process management: a survey. In: Proceedings of the International Conference on Business Process Management (BPM 2003), pp. 1–12.
van der Aalst, W.M.P., Weijters, T., Maruster, L., 2004. Workflow mining: Discovering process models from event logs. IEEE Trans. Knowl. Data Eng. 16 (9), 1128–1142.
van der Aalst, W.M.P., 2012. Decomposing process mining problems using passages. Lect. Notes Comput. Sci. 7347, 72–91.
van der Aalst, W.M.P., Adriansyah, A., de Medeiros, A.K.A., et al., 2012. Process mining manifesto. Proceedings of the Business Process Management Workshops 1, 169–194. 2011.
van der Werf, J.M., van Dongen, B., van Hee, K., Hurkens, C., Serebrenik, A., 2009. Process discover using integer linear programming. Fundam. Inf. 94 (3-4), 387–412.
Vázquez-Barreiros, B., Mucientes, M., Lama, M., 2014. ProDiGen: Mining complete, precise and minimal structure process models with a genetic algorithm. Inf. Sci. 294, 315–333.
Wang, D., Ge, J., Hu, H., Luo, B., Huang, L., 2012. Discovering process models from event multiset. Expert Syst. Appl. 39 (15), 11970–11978.
Wen, L., van der Aalst, W.M.P., Wang, J., Sun, J., 2007. Mining process models with non-free-choice constructs. Data Min. Knowl. Discov. 15 (2), 145–180.
Wen, L., Wang, J., van der Aalst, W.M.P., Huang, B., Sun, J., 2009. A novel approach for process mining based on event types. J. Intell. Inf. Syst. 32 (2), 163–190.
Wen, L., Wang, J., van der Aalst, W.M.P., Huang, B., Sun, J., 2010. Mining process models with prime invisible tasks. Data Knowl. Eng. 69 (10), 999–1021.
White, A.S., 2013. A control model of the software requirements process. Kybernetes 42 (3), 423–447.
Xu, H., Pete, S., Ian, S., 2006. Requirement process establishment and improvement from the viewpoint of cybernetics. J. Syst. Softw. 79 (11), 1504–1513.
Yamalidou, K., Moody, J., Lemmon, M., Antsaklis, P., 1996. Feedback control of petri nets based on place invariants. Automatica 32 (1), 15–28.
Yang, H., Zhang, L., 2014. Controlling and being creative: software cybernetics and creative computing. In: Proceedings of IEEE 38th Annual International Computers, Software and Applications Conference Workshops (COMPSACW), pp. 19–24.
Yang, Z.J., Yin, B.B., Lv, J., Cai, K.Y., Yau, S.S., Yu, J., 2014. Dynamic random testing with parameter adjustment. In: Proceedings of IEEE 38th International Computer Software and Applications Conference Workshops (COMPSACW2014), pp. 37–42.
Yu, W.Y., Yan, C.G., Ding, Z.J., Jiang, C.J., Zhou, M.C., 2014. Modeling and validating e-commerce business process based on Petri nets. IEEE Trans. Syst. Man, Cybern. Syst. 44 (3), 327–341.
Zeng, Q., Wang, H., Xu, D., Duan, H., Han, Y., 2008. Conflict detection and resolution for workflows constrained by resources and non-determined durations. J. Syst. Softw. 81 (9), 1491–1504.
Zhu, W.L., Yin, B.B., Cai, K.Y., Huang, D., Yau, S.S., 2013. An adaptive control strategy for managing concurrent service requests in service-based systems. In: Proceedings of the 37th IEEE Annual Computer Software and Applications Conference Workshops (COMPSACW), pp. 35–40.



Chuanyi Li is a PhD candidate at the Software Institute, Nanjing University, under the supervision of Prof. Jidong Ge and Prof. Bin Luo. His research interests include software engineering, systems and software quality assurance, process modeling, simulation and improvement, process mining.

Jidong Ge is an Associate Professor at Software Institute, Nanjing University. He received his PhD degree in Computer Science from Nanjing University in 2007. His current research interests include software engineering, systems and software quality assurance, process modeling, simulation and improvement, process mining. His research results have been published in more than 40 papers in international journals and conference proceedings including JASE, ESA, ICSE, APSEC, ICSSP, HPCC, SEKE etc.

LiGuo Huang is an Associate Professor, Department of Computer Science and Engineering at the Southern Methodist University (SMU), Dallas, TX, USA. She received both her PhD (2006) and M.S. from the Computer Science Department and Center for Systems and Software Engineering (CSSE) at the University of Southern California (USC). Her current research centers around process modeling, simulation and improvement, systems and software quality assurance and information dependability, mining systems and software engineering repository, stakeholder/value-based software engineering, and software metrics. Her research has been supported by NSF, U.S. Department of Defense (DoD) and NSA. She had been intensively involved in initiating the research on stakeholder/value-based integration of systems and software engineering. She can be contacted at lghuang@smu.edu.

Haiyang Hu is a full Professor at School of Computer, Hangzhou Dianzi University, Hangzhou, China. He received the BS, MS, and PhD degrees in computer science from Nanjing University, Nanjing, China, in 2000, 2003, and 2006, respectively. His research interests include mobile computing and distributed computing. His research results have been published in more than 20 papers in international journals and conference proceedings.

Budan Wu is an Associate Professor of computer science in State Key Laboratory of Networking and Switching Technology at Beijing University of Posts and Telecommunications. Her current research interests include software engineering, service-oriented computing and business process management.

Hao Hu is an Associate Professor at Department of Computer Science and Technology, Nanjing University. He received his PhD degree in Computer Science from Nanjing University in 2009. His current research interests include software engineering, systems and software quality assurance, process modeling, simulation and improvement, process mining. His research results have been published in more than 30 papers in international journals and conference proceedings including ICSE, ICSM, ICSSP etc.

Bin Luo is a full Professor at the Software Institute, Nanjing University. His main research interests include software engineering, systems and software quality assurance, process modeling, simulation and improvement, process mining. His research results have been published in more than 40 papers in international journals and conference proceedings. He is leading the institute of applied software engineering at Nanjing University.