Модел. и анализ информ. систем. Т. **22**, № **3** (2015) 392–403
©*Begicheva A. K., Lomazova I. A.*, 2015

DOI: 10.18255/1818–1015–2015–3–392–403
UDC 517.9

Does Your Event Log Fit the High-Level Process Model?[1]

Begicheva A. K., Lomazova I. A.

National Research University Higher School of Economics
Myasnitskaya str. 20, Moscow, 101000, Russia

e-mail: ilomazova@hse.ru, akbegicheva@edu.hse.ru

received June 28, 2015

Keywords: Petri nets, high-level process models, event logs, process mining, conformance checking

Process mining is a relatively new field of computer science, which deals with process discovery and analysis based on event logs. In this paper we consider the problem of models and event logs conformance checking. Conformance checking is intensively studied in the frame of process mining research, but only models and event logs of the same granularity were considered in the literature. Here we present and justify the method of checking conformance between a high-level model (e.g. built by an expert) and a low-level log (generated by a system).

The article is published in the author's wording.

Introduction

Process mining [1] is a new technology, that provides variety of methods to discover, monitor and improve real processes by extracting knowledge from event logs. Conformance checking [1, 8, 7, 3] is one of the most prominent process mining tasks. It is needed for diagnosis and quantifying discrepancies between observed and modeled behavior. There are many software products which allow us to use methods of Process Mining. ProM [6] is an open-source tool supporting many techniques of Process Mining, which are represented as plug-ins. Due to a flexibility of this environment it can be used both for research and applications.

Conformance checking uses both an event log and a model, and compares observed behavior written in the log with the behavior produced by the model. The general goal is to find discrepancies between them to improve a model. Conformance checking techniques can also be used for measuring the performance of process discovery algorithms (that

[1]This study was carried out within the National Research University Higher School of Economics' Academic Fund.

restores a model on the basis of a known log) and to repair models that have not got a well alignment with the real behavior of the process.

There are four evaluation criteria in conformance checking: fitness, precision, generalization and simplicity. Fitness shows at what extent traces from the event log can be reproduced by the model. Among the other quality criteria, fitness is the most related to the conformance. An obvious approach to measure fitness is to count the fraction of cases that can be "parsed completely"(i.e. the proportion of cases corresponding to firing sequences leading from *[start]* to *[end]*). Fitness can range from 0 to 1. It is supposed that fitness is equal to 1, if the log *perfectly fits* the model.

More subtle methods for measuring fitness are based on the *replay* approach [8]. When measuring fitness by replaying, we do not stop replaying a trace when we face a problem, but continue replaying the trace, and record a count of all missing tokens and all tokens that are pending at the end.

When working with business processes we typically use detailed logs, which present the full report about sequences of executed activities. Since in most information systems logs are generated automatically, keeping detailed records is not a problem. However, large and detailed models are not good to deal with. Such models are not clear and readable for experts. Experts prefer working with more abstract (high-level) models. More abstract models are easier to construct, understand and analyze. Process models developed by people are, as a rule, not very large and abstract from technical details. The problem has been studied in the literature only for discovering abstract models from low-level event logs. In [4, 5] methods for discovering abstract models based on finding behavior patterns in event logs were proposed. Thus checking conformance of an abstract model and a low-level event log generated by an information system is an important and challenging problem.

In this paper we consider process models represented by workflow nets – a special class of Petri nets for workflow modeling [2]. In an abstract model each separate activity represents a sub-process built from a set of more refined activities. A history of a detailed process behavior is recorded in low-level logs. We present a algorithm for checking conformance between an abstract model and a low-level event log. The algorithm is proved to be correct for perfectly fitted logs. Software implementation of the algorithm is developed as a plug-in for ProM. The algorithm was tested on groups of input data with different characteristics and types of noise.

The paper is organized as follows. In Section II we give a motivating example of handling a request for a compensation within airline in terms of Petri nets. Section III contains some basic definitions and notions, including Petri nets, event log, perfect fitness and refinement. In Section IV we present a method for checking conformance between an abstract model and a low-level log. We also give a justification of this method by proving its correctness in the case of perfect fitness. Experiments that confirm the adequacy of our algorithm on imperfect data are described in Section V. Section VI contains some conclusions.

1. Motivating Example

Let us consider a toy model from [1], which describes handling a request for a compensation from the airline. Here customers may request compensations for various reasons. An

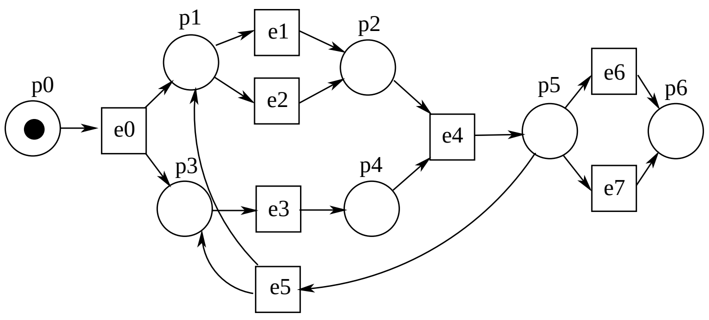

Fig. 1. An abstract model $\mathcal{N}_1$ for handling compensation requests

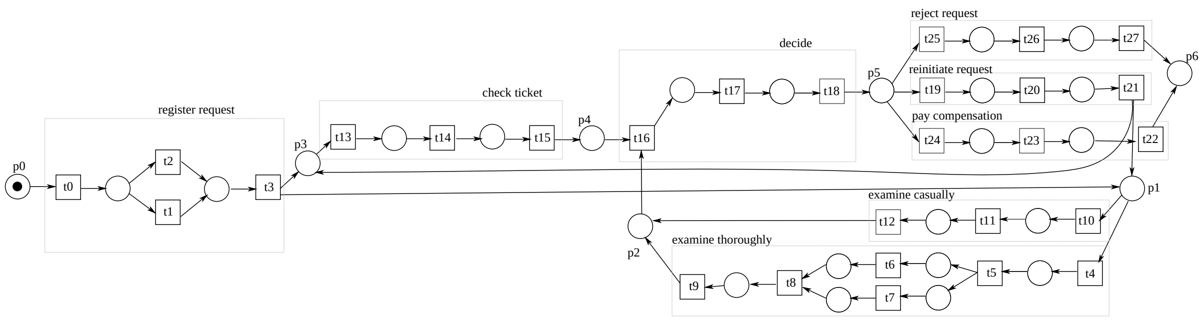

Fig. 2. A low-level model $\mathcal{N}_2$ refined from the model $\mathcal{N}_1$ in Fig. 1

abstract model of this process (expressed in terms of a *Petri net*) is presented in Fig.1. *Transitions*, corresponding to activities, are pictorially represented by squares and connected through other types of elements — *places*. Each place is represented as a circle and models a local state of the process. A distribution of tokens in net places indicates a total state of the system and is called a *marking*. A transition is *enabled* when each of its input places contains a *token*. For example the transition $e0$ can fire since the initial place $p0$ contains a token. When *firing*, a transition consumes one token from each its input place and produces one token for each of its output places.

The process begins with registering the request (transition $e0$). The meaning of other transitions is as follows: $e1$ — examine thoroughly, $e2$ — examine casually, $e3$ — check ticket, $e4$ — decide, $e5$ — re-initiate request, $e6$ — pay compensation and $e7$ — reject request.

Fig.2 presents a refined model of the same process, obtained by substitution of detailed descriptions for abstract activities. For example, 'register request' action is to read the request (transition $t0$) and then to record it in one of two possible ways (transition $t1$, or $t2$). To avoid congestion of activities' names in the low-level model in Fig.2 only places inherited from the abstract model are labeled in the picture. Low-level transitions in Fig.2 are grouped into blocks according to the high-level activities.

We study the situation when a low-level model is not available (or even does not exist). Given an abstract (high-level) model constructed by experts or software developers, and an event log generated by an information system, we would like to know whether the model conforms the real system behavior represented by the event log. A sample of such

$$
\begin{aligned}
L = \{ & < t0, t1, t3, t4, t5, t6, t13, t14, t7, t8, t15, t9, t16, t17, t18, t25, t26, t27 >, \\
& < t0, t1, t3, t4, t13, t14, t5, t7, t15, t6, t8, t9, t16, t17, t18, t24, t23, t22 >, \\
& < t0, t1, t3, t13, t10, t11, t14, t15, t12, t16, t17, t18, t24, t23, t22 >, \\
& < t0, t2, t3, t13, t4, t14, t15, t5, t6, t7, t8, t9, t16, t17, t18, t25, t26, t27 >, \\
& < t0, t2, t3, t4, t13, t14, t15, t5, t7, t6, t8, t9, t16, t17, t18, t24, t23, t22 >, \\
& < t0, t1, t3, t10, t11, t13, t12, t14, t15, t16, t17, t18, t19, t20, t21, t10, t11, \\
& t13, t14, t15, t12, t16, t17, t18, t25, t26, t27 >, \\
& < t0, t2, t3, t13, t10, t14, t15, t11, t12, t16, t17, t18, t24, t23, t22 >, \\
& < t0, t2, t3, t13, t10, t11, t12, t14, t15, t16, t17, t18, t19, t20, t21, t10, t13, \\
& t14, t11, t15, t12, t16, t17, t18, t24, t23, t22 >, \\
& < t0, t2, t3, t13, t14, t10, t11, t15, t12, t16, t17, t18, t25, t26, t27 >\}.
\end{aligned}
$$

Fig. 3. An event log $\mathcal{L}_2$, generated by the refined model $\mathcal{N}_2$ in Fig. 2

an event log for our example is shown in Fig.3. This log is generated by the low-level model in Fig.2, and thus perfectly fits it. An abstract model and a low-level event log cannot be directly compared, since in the model high-level events are used, and the log contains low-level events. So, a one-to -many correspondence between high- and low-level events is needed for checking conformance. In our example the abstract event $e0$ corresponds to the set $\{t0, t1, t2\}$ of low-level events, $e1$ — to the set $\{t13, t14, t15\}$, etc. This correspondence will be used in the algorithm for measuring conformance between a high-level model and a low-level event log.

2. Preliminaries

In this section we give some basic notions and definitions used the paper.

Let S be a set. By S^* we denote the set of all finite sequences (words) over S. $S = S_1 \cup S_2 \cup \ldots \cup S_n$ is a *partition* of S iff $\forall i, j \in [1, n] : S_i \subseteq S$ and $S_i \cap S_j = \emptyset$.

A *multiset* m over a set S is a mapping $m : S \to Nat$, where Nat is the set of natural numbers (including zero), i.e. a multiset may contain several copies of the same element.

Definition 1 (Petri net). *Let P and T be disjoint finite sets of* places *and* transitions *and $F \colon (P \times T) \cup (T \times P) \to Nat$. Then $N = (P, T, F)$ is a* Petri net. *Let A be a finite set of activities. A labeled Petri net is a Petri net with a labeling function $\lambda : T \to A \cup \{\epsilon\}$ which maps every transition to an activity (a transition label) from A, or a special label ϵ, corresponding to an invisible action.*

A *marking* in a Petri net is a function $m \colon P \to Nat$, mapping each place to some natural number (possibly zero).

For a transition $t \in T$ a *preset* ${}^\bullet t$ and a *postset* $t^\bullet$ are defined as the multisets over P such that ${}^\bullet t = \{p | F(p, t) \neq 0\}$ and $t^\bullet = \{p | F(t, p) \neq 0\}$ for each $p \in P$.

A transition $t \in T$ is *enabled* in a marking m iff $\forall p \in P \; m(p) \geq F(p, t)$. An enabled transition t may *fire* yielding a new marking i. e. $m'(p) = m(p) - F(p, t) + F(t, p)$ for

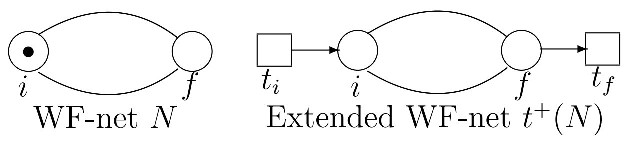

Fig. 4. An extending WF-net with initial and final transitions

each $p \in P$ (denoted $m \xrightarrow{t} m'$).

A *Workflow net* is a (labeled) Petri net with two special places: i and f. These places are used to mark the beginning and the ending of a workflow process.

Definition 2 (Workflow net)**.** *A (labeled) Petri net $N = (P, T, F, \lambda)$ is called a* workflow net (WF-net) *iff*

1. *There is one source place $i \in P$ and one sink place $f \in P$ s. t. ${}^\bullet i = f^\bullet = \emptyset$;*
2. *Every node from $P \cup T$ is on a path from i to f.*
3. *The initial marking in N contains the only token in its source place.*

By abuse of notation we denote by i both the source place and the initial marking in a WF-net. Similarly, we use f to denote the final marking in a WF-net N, defined as a marking containing the only token in the sink place f.

Let $N = (P, T, F, \lambda)$ be a WF-net. Then we define the *extended WF-net (EWF-net)* $N' = (P', T', F', \lambda')$ as follows: $P' = P, T' = T \cup \{t_i, t_f\}$, and $F' = F \cup \{\langle t_i, i\rangle, \langle f, t_f\rangle\}$, where t_i, t_f are new (not occurring in P, T) nodes. The new transitions t_i, t_f are labeled with invisible activity ϵ in N', all other transitions in N' have the same labels as in N. The initial marking in an extended WF-net contains no tokens. Thus an extended WF-net may start a new case at any moment (cf. Fig.4).

Event logs keep a history of process executions.

Definition 3 (Event log)**.** *Let A be a finite set of activities. A* trace *σ (over A) is a finite sequence of activities, i.e., $\sigma \in A^*$. An* event log *L (over A) is a finite multi-set of traces, i.e. $L \in \mathcal{M}(A^*)$.*

In this paper we study conformance checking. Given a model and an event log we would like to compare the process model behavior and the behavior recorded in the event log. Several metrics for conformance checking were defined in the literature [1]. Among the most important metrics is *fitness*. Informally speaking, fitness measures the proportion of behavior in the event log possible according to the model.

Definition 4 (Perfect fit)**.** *Let N be a WF-net with transition labels from A, an initial marking i, and a final marking f. Let σ be a trace over A. We say that a trace $\sigma = a_1, \ldots, a_k$* perfectly fits *$N$ iff there exists a sequence of firings $i = m_0 \xrightarrow{t_1} \ldots \xrightarrow{t_k} m_{k+1} = f$ in N, s.t. the sequence of activities $\lambda(t_1), \lambda(t_2), \ldots, \lambda(t_k)$ after deleting all invisible activities ϵ coincides with σ. A log L* perfectly fits *N iff every trace from L perfectly fits N.*

Petri nets can be extended by adding a hierarchy as it is done e.g. in Colored Petri nets (CPN) [9]. Hierarchy allows to develop more compact and readable models. In the case of two-level hierarchy there are two models of one process: a high-level (*abstract*) model and a low-level (*refined*) model. The high-level model is a model with abstract transitions. An abstract transition refers to a Petri net sub-process, which refines the activity represented by this transition. The low-level model can be obtained from an abstract model by substituting subprocess models for abstract transitions.

Definition 5 (Substitution). *Let $N_1 = (P_1, T_1, F_1, \lambda_1)$ be a WF-net, $t \in T$ be a transition in N_1. Let also $N_2 = (P_2, T_2, F_2, \lambda_2)$ be an EWF-net with the initial and final transitions t_i, t_f correspondingly. We say that a WF-net $N_3 = (P_3, T_3, F_3, \lambda)$ is obtained by a* substitution $[t \to N_2]$ *of N_2 for t in N_1 iff $P_3 = P_1 \cup P_2$, $T_3 = T_1 \cup T_2 \setminus \{t\}$, $F_3 = F_1 \cup F_2 \setminus \{(p,t) \mid p \in {}^\bullet t\} \setminus \{(t,p) \mid p \in t^\bullet\} \cup \{(p,t_i) \mid p \in {}^\bullet t\} \cup \{(t_f,p) \mid p \in t^\bullet\}$,*

Definition 6 (Refinement). *Let N, N_r be two WF-nets with sets of activities A, A_r correspondingly. Let $A = a_1, a_2, \ldots, a_n$, and $A_r = A_r^1 \cup A_r^2 \cup \ldots \cup A_r^n$ be a partition of A_r into n subsets, and $N^1, N^2, \ldots N^n$ be EWF-nets with sets of activities $A_r^1, \ldots, A_r^n$ correspondingly. We say that N_r is a refinement of N via substitutions $[a_1 \to N_r^1, a_2 \to N_r^2, \ldots a_n \to N_r^n]$ iff N_r can be obtained from N by simultaneous substitutions of N_r^i for all t s.t. $\lambda(t) = a_i$.*

3. Algorithm for Checking Conformance between an Abstract Model and a Low-Level Event Log

The idea of the algorithm is as follows. To check conformance between an abstract model and a low-level log, we first transform the given log into a log over abstract activities. For this purpose, each low-level activity in the log is replaced by a name of the sub-process (an abstract activity) it belongs to. As a result we get a log over the set of abstract activities.

Traces in this log may contain several sequential occurrences of the same abstract activity, since there are several steps corresponding to this activity in the low-level trace. Then the next step is to get rid of “stuttering"by replacing several sequential occurrences of the same activity with just one.

However, this is still not enough to start checking conformance using known methods. Note, that when we have two concurrent sub-processes, represented by two concurrent abstract activities in an abstract model, stuttering sequences may interleave. To overcome this problem we transform an abstract model into a model allowing stuttering of each abstract activity. This is done by adding loops to transitions in the abstract model.

Now we describe the algorithm for checking conformance between an abstract model and a low-level log more precisely.

Algorithm 1. *(Checking conformance between a high-level model and a low-level event log).*

Let $N = (P, T, F, \lambda)$ be a WF-net corresponding to an abstract model of a process over a set of activities A, and let L_r be an event log (a finite multiset of traces) over a set A_r of low-level activities. Let also $\delta : A_r \to A$ be a function that maps each low-level activity to a certain high-level activity from A.

$$\begin{aligned} L = \{ & < e0, e1, e3, e1, e3, e1, e4, e7 >, < e0, e1, e3, e1, e3, e1, e4, e6 >, \\ & < e0, e3, e2, e3, e2, e4, e6 >, < e0, e3, e1, e3, e1, e4, e7 >, < e0, e1, e3, e1, e4, e6 >, \\ & < e0, e2, e3, e2, e3, e4, e5, e2, e3, e2, e4, e7 >, < e0, e3, e2, e3, e2, e4, e6 >, \\ & < e0, e3, e2, e3, e4, e5, e2, e3, e2, e3, e2, e4, e6 >, < e0, e3, e2, e3, e2, e4, e7 > \}. \end{aligned}$$

Fig. 5. The 'abstract' event log $\mathcal{L}_1$ obtained by converting the log $\mathcal{L}_2$ in Fig. 3

Step 1. Convert L_r into an event log L over the set of activities A by replacing each activity $a \in L_r$ in each trace with the activity $\delta(a)$.

Step 2. Get rid of 'stuttering' in L by replacing for each trace each substring consisting of the same repeating activity with one occurrence of this activity.

Step 3. Check whether there are traces in L with with more than one (not consecutive) occurrences of the same activity. If there are no such traces, go to the Step 5, otherwise go to the Step 4.

Step 4. For a trace $\sigma \in L$, let B_σ be the set of activities which have more than one occurrence in σ, and let $B \subseteq A$ be the set of all activities with multiple occurrences in L, i.e. $B = \cup_{\sigma \in L} B_\sigma$.

For each transition t labeled with an activity from B in the net N add a loop, as follows:

- add two new transitions to T in N: a transition t_ϵ labeled by the invisible action ϵ and a transition t' labeled by $\lambda(t)$;
- add a new place p' to P in N;
- add new arcs $(t_\epsilon, p'), (p', t'), (t', p'), (p', t)$, and for each place $p \in {}^\bullet t$ add an arc (p, t_ϵ).

Step 5. Measure the conformance between N (with the added loops) and L (without stuttering) by applying one of existing algorithms (e.g. the replay algorithm).

We illustrate this algorithm by applying it to the WF-net $\mathcal{N}_1$ in Fig. 1 and the log $\mathcal{L}_2$ in Fig. 3. First, we convert the event log $\mathcal{L}_2$ into a high-level log. The log obtained as the result of this is denoted by $\mathcal{L}_1$ and is shown in Fig. 5.

Then we add loops to the abstract model $\mathcal{N}_1$ and obtain the new model $\mathcal{N}_1'$, shown in Fig. 6. Here the invisible transitions are colored in black. The model N_a' allows stuttering of activities $e1, e2$, and $e3$.

And finally we check conformance between the model $\mathcal{N}_1'$ and the log $\mathcal{L}_1$ by replaying traces from $\mathcal{L}_1$ in $\mathcal{N}_1'$. It turns out, that all traces from $\mathcal{L}_1$ can be exactly replayed in $\mathcal{N}_1'$, i.e. the log $\mathcal{L}_1$ perfectly fits $\mathcal{N}_1'$. This is not by chance. The following theorem states, that the proposed conformance checking method is correct under perfect fitness.

Theorem 1. *Let N be a WF-net with transitions labeled by activities from a set A, and let N_r be a WF-net, obtained by refining N with transitions labeled by activities from A_r.*

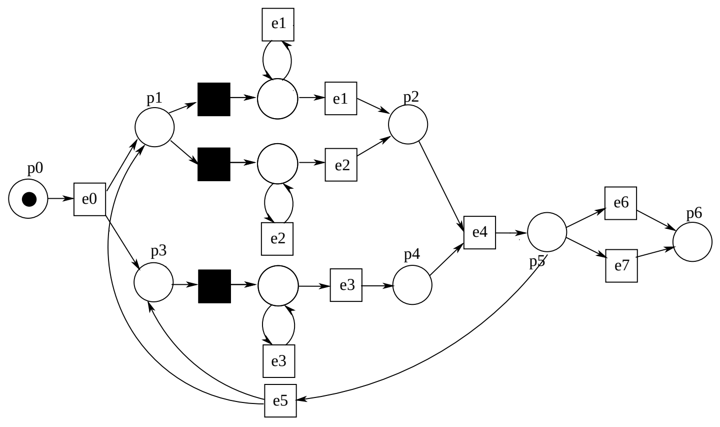

Fig. 6. The abstract model $\mathcal{N}_1$ after adding loops

Let also $L_r \in \mathcal{M}(A_r^*)$ be an event log over the set A_r. Then if L_r perfectly fits N_r, the output of the Algorithm 1 will be equal to 1.

The proof of this theorem is based on checking (by induction on the trace length) that in the theorem conditions each trace from L with removed stutterings can be exactly replayed in N with added loops. We omit the detailed proof of the theorem, since it is rather technical and straightforward.

4. Experimental Validation of Algorithm

We have proven, that our method recognizes perfect fitness between an abstract model and a low-level log correctly. This can be considered as a justification of the proposed approach. However, it is not enough, since it is very important to check the method on logs and/or models with deviations.

To ensure that the algorithm is suitable for checking conformance not only in the case of perfect fitness, we have implemented the algorithm and run it on different input data: event logs with noise, models with some small modifications, and models with significant deviations.

The proposed algorithm was implemented as a plug-in named "Conformance Checking for High-Level model and Event log, Transformation"within the ProM 6 framework [11]. Our plug-in receives a high-level process model in PNML format and a low-level event log in XES format as an input. During the plug-in execution first the Petri net model is visualized, then the user is asked to define a partition for the set of actions occurring in the log and to match an abstract event (occurring in the model) to each subset of the partition. After running the plug-in the user can apply any known algorithm for the standard conformance checking. There are several such algorithms in ProM.

For measuring fitness the plug-in for replay-based conformance analysis (described in [3]) was used.

The first group of experiments concerns event logs with some noise. We consider several kinds of noise:

Kind of noise	Level of noise	Low-level fitness	High-level fitness
adding an event	10	0,888	0,878
	10	0,905	0,894
	10	0,874	0,888
	10	0,888	0,864
adding a new event	10	0,894	0,892
	10	0,885	0,878
skipping an event	10	0,93	1
	20	0,876	1
	30	0,782	0,986
	40	0,706	0,967
	50	0,629	0,920
	60	0,533	0,894

Table 1. Fitness for event logs with noise

Type of routing	Low-level trace fitness	High-level trace fitness
sequential	0,95	0,96
AND-split	0,97	0,96
AND-join	0,97	0,96
OR-split	0,96	0,97
OR-join	0,97	0,96

Table 2. Fitness for event logs with specific noise

1. Adding an extra record for the event occurrence in a random place in the event log for an event presented already in the model.

2. Adding a record for the event occurrence in a random place in the event log for some new event.

3. Skipping randomly a record for the event occurrence.

4. Adding some specific noise, significant for the conformance of the high-level model, while not so important for the low-level conformance.

Experiments were carried out for the artificial model in Fig. 1 and its refinement in Fig. 2. Logs with different kinds and levels of noise were generated with the help of Log Generator plug-in [10], where the level of noise is measured as the probability of applying a certain kind of change to records in the event log.

The obtained results are presented in Table 1 and Table 2. Here the low-level fitness is the fitness for an event log and the refined model, measured by direct replay-based conformance analysis, and the high-level fitness is the fitness for the same event log and the abstract model, computed using our algorithm.

Changing the model	Average fitness
adding a transition	1
adding an arc	0,912
adding a place	0,977
removing a transition	0,884
removing a place	0,963
removing an arc	0,969
swapping two transitions	0,702

Table 3. Fitness for models with minor structural changes

In most cases the high-level fitness values are greater than the corresponding low-level values, since deviations on the low-level may be not noticeable on the level of abstract model.

The cases when the high-level fitness is a little less then the low-level one can be also explained. This can be caused by an artificial random noise inserting a low-level event into a group of events from some different abstract activity. We believe, this case is very rare in real-life logs.

Some information systems record not just event execution, but starting and ending of an event. Then deviations in fixing timestamps may lead to violations in fixing events order. Such cases we consider as a specific noise (connected with fixing timestamps), which could be supposed to crucially influence the high-level fitness values. For imitating this kind of noise we added noise affecting concrete routing constructions. The experiment results in Table 2 shows that such noise is not really important.

The second group of experiments concerns small deviations in the abstract model. Experiments were done for the same artificial abstract model. The question was the extent to which local changes in the model affect the fitness results. To answer this question we have measured fitness of originally perfectly fitting logs for models, obtained from the original abstract model with local structural changes. The results are presented in Table 3. They show the resistance of our algorithm to small structural changes in the model. Note also, that the replay-based algorithm allows not just measure the fitness, but also to localize log deviations in the model. The experiments show that our algorithm localizes deviations in the same way as the original replay-based algorithm. We made some changes in the plug-in, it allows us to verify that after our transformation the location of noise is recognized correctly. This test also was successfully passed.

One more group of experiments concerns measuring fitness, when the level of conformance between a model and an event log is far from being perfect. To check the stability of our algorithm we compare the high-level fitness for an abstract model and an event log, measured by our method, and the low-level fitness, i.e. the fitness of the same event log and a refined model measured by direct replaying the log.

The results for different degrees of conformance are shown in Table 4. As expected, here also the high-level fitness values are greater than the corresponding low-level values. The reason is in more rough granularity of the abstract model, when some subprocess deviations cannot reflected in the model. However, there is a good correspondence between high-level and low-level (used as a reference) fitnesses.

	High degree of conformance		Middle degree of conformance		Low degree of conformance	
	High-level fitness	Low-level fitness	High-level fitness	Low-level fitness	High-level fitness	Low-level fitness
case 1	0.928	0.872	0.655	0.431	0.608	0.412
case 2	0.903	0.866	0.691	0.509	0.633	0.361
case 3	0.866	0.827	0.568	0.312	0.524	0.279
case 4	0.885	0.787	0.599	0.398	0.573	0.346

Table 4. Fitness for models with different degrees of conformance

5. Conclusion

Abstract models are much more clear and more understandable than low-level models. But information systems generate only low-level event logs, which cannot be used for direct conformance checking. So, checking conformance of a high-level business model to a low-level event log is very important for facilitating the work of experts on analysis and enhancement of business information systems.

In this paper we provide a robust technique for solving this problem, which is based on transforming both the model and the log and then applying one of already known methods for measuring fitness. We've proved our method to be correct in the case of perfect conformance between a model and an event logs. Also we had developed a ProM plug-in which implements the proposed algorithm and tested applicability of our technique on artificial imperfect data with noise and deviations.

References

[1] van der Aalst W. M. P., *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, Springer-Verlag, Berlin, 2011.

[2] van der Aalst W. M. P., van Hee K. M., *Workflow Management: Models, Methods and Systems*, MIT Press, Cambridge, MA, 2002.

[3] Adriansyah A., van Dongen B. F., van der Aalst W. M. P., “Conformance Checking Using Cost-Based Fitness Analysis”, *IEEE 15th International Enterprise Distributed Object Computing Conference*, 2011, 55–64.

[4] Bose R. P. J. C., van der Aalst W. M. P., “Abstractions in Process Mining: A Taxonomy of Patterns. In Business Process Management”, *Lecture Notes in Computer Science*, **5701** (2009), 159–175.

[5] Bose R. P. J. C., Verbeek H. M. W., van der Aalst W. M. P., “Discovering Hierarchical Process Models Using ProM”, *CAiSE Forum (Selected Papers)*, 2011, 33–48.

[6] van Dongen B. F., Alves de Medeiros A. K., Verbeek H. M. W., Weijters A. J. M. M., van der Aalst W. M. P., “The ProM framework: A New Era in Process Mining Tool Support”, *Lecture Notes in Computer Science*, **3536** (2005), 444–454.

[7] Rozinat A., *Process mining: conformance and extension*, Technische Universiteit, Eindhoven, 2010.

[8] Rozinat A., van der Aalst W. M. P., “Conformance Testing: Measuring the Alignment Between Event Logs and Process Models”, *BETA Working Paper Series*, **144** (2005), 203–210.

[9] Jensen K., Kristensen L. M., “Coloured Petri Nets: Modelling and Validation of Concurrent Systems”, 2009.

[10] Shugurov I., Mitsyuk A., “Generation of a Set of Event Logs with Noise”, *Proc. of the 8th Spring/Summer Young Researchers’ Colloquium on Software Engineering (SYRCoSE 2014)*, 2014, 88–95.

[11] Verbeek H. M. W., Buijs J. C. A. M., van Dongen B. F., van der Aalst W. M. P., “Prom 6: The process mining toolkit”, *Proc. of BPM Demonstration*, **615**, 2010, 34–39.

Соответствует ли Ваш журнал событий высокоуровневой модели процесса?

Бегичева А.К., Ломазова И.А.

Национальный Исследовательский Университет «Высшая Школа Экономики» , 101000, Россия, Москва, ул. Мясницкая, 20

Ключевые слова: сети Петри, высокоуровневые модели процессов, журналы событий, Process Mining, проверка соответствия

Process mining – это технология, которая посредством извлечения данных из журнала событий предоставляет различные методы для исследования реального процесса, его улучшения и контроля над ним. В данной статье мы рассматриваем проблему проверки соответствия между высокоуровневой моделью процесса и журналом событий. Проверка соответствия интенсивно изучается в рамках process mining, но в литературе можно найти только методы, позволяющие измерить этот показатель между логом и моделью одного уровня. В статье мы представляем алгоритм проверки соответствия между высокоуровневой моделью процесса (построенной экспертами) и низкоуровневым журналом событий (сгенерированным системой), а также доказываем его применимость.

Статья публикуется в авторской редакции.

Сведения об авторах:

Бегичева Антонина Константиновна,
Национальный исследовательский университет «Высшая школа экономики»,
Научно-учебная лаборатория ПОИС, стажер-исследователь

Ломазова Ирина Александровна,
Национальный исследовательский университет «Высшая школа экономики»,
доктор физ.-мат. наук, профессор