*Моделирование и анализ информационных систем.* Т. 24, № 4 (2017), с. 459–480
*Modeling and Analysis of Information Systems.* Vol. 24, No 4 (2017), pp. 459–480



**DOI:** 10.18255/1818-1015-2017-4-459-480

УДК 519.682.6+004.021

# Использование журналов событий для локальной корректировки моделей процессов

**Мицюк А. А.**[1], **Ломазова И. А.**[1], **ван дер Аалст В. М. П.**

*получена 17 июля 2017*

**Аннотация.** В ходе жизненного цикла информационной системы (ИС) ее реальное поведение может перестать соответствовать исходной модели системы. Между тем для поддержки системы очень важно иметь актуальную модель, отражающую текущее поведение системы. Для корректировки модели можно использовать информацию из журнала событий системы. Журналы событий процессно-ориентированных информационных систем содержат запись истории исполнения поддерживаемых процессов в виде более или менее детальных списков событий. Такие журналы, как правило, записываются всеми современным ИС. Эта информация может использоваться для анализа реального поведения ИС и ее усовершенствования. В работе рассматривается задача корректировки (исправления) модели процесса на основе информации из журнала событий. Исходными данными для этой задачи являются первоначальная модель процесса в виде сети Петри и журнал событий. Результатом корректировки должна быть новая модель процесса, лучше отображающая реальное поведение ИС, чем исходная модель. Актуальная модель может быть построена и полностью заново, например, с помощью одного из известных алгоритмов автоматического синтеза модели процесса по журналу событий. Однако структура исходной модели при этом может полностью измениться. Полученную модель будет трудно сопоставить с прежней моделью процесса, что затруднит ее понимание и анализ. Поэтому при корректировке модели важно по возможности сохранить ее прежнюю структуру. Предлагаемый в настоящей работе алгоритм корректировки модели основан на принципе «разделяй и властвуй». Исходная модель процесса декомпозируется на фрагменты. Для каждого из фрагментов проверяется, соответствует ли он актуальному журналу событий. Фрагменты, для которых выявлены несоответствия, заменяются на заново синтезированные. Новая модель собирается из фрагментов путем слияния переходов. Проведенные эксперименты показывают, что наш алгоритм корректировки дает хорошие результаты, если применяется для исправления локальных несоответствий. Работа содержит описание алгоритма, формальное обоснование его корректности, а также результаты экспериментального тестирования на искусственных примерах.

**Ключевые слова:** извлечение и анализ процессов, исправление моделей процессов, сети Петри, декомпозиция моделей процессов, разделяй и властвуй

**Для цитирования:** Мицюк А. А., Ломазова И. А., ван дер Аалст В. М. П., "Использование журналов событий для локальной корректировки моделей процессов", *Моделирование и анализ информационных систем*, **24**:4 (2017), 459–480.

**Об авторах:**
Мицюк Алексей Александрович, orcid.org/0000-0003-2352-3384, науч. сотр.,
Национальный исследовательский университет «Высшая школа экономики», лаборатория ПОИС
ул. Мясницкая, 20, г. Москва, 101000 Россия, e-mail: amitsyuk@hse.ru

Ломазова Ирина Александровна, orcid.org/0000-0002-9420-3751, д-р физ.-мат. наук, профессор
Национальный исследовательский университет «Высшая школа экономики», лаборатория ПОИС
ул. Мясницкая, 20, г. Москва, 101000 Россия, e-mail: ilomazova@hse.ru

Вил М.П. ван дер Аалст, orcid.org/0000-0002-0955-6940, профессор
Технический университет Эйндховена (TU/e), группа «Архитектура информационных систем»,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, e-mail: w.m.p.v.d.aalst@tue.nl

**Благодарности:**
[1]Исследование выполнено в рамках Программы фундаментальных исследований Национального исследовательского университета «Высшая школа экономики» (НИУ ВШЭ).

# Введение

Журналы событий процессно-ориентированных информационных систем (ИС) содержат запись истории исполнения поддерживаемых процессов в виде более или менее детальных списков событий. Такие журналы, как правило, записываются всеми современным ИС. Эта информация может использоваться для анализа реального поведения ИС и ее усовершенствования [3].

В данной работе рассматривается задача корректировки (исправления) модели процесса на основе информации из журнала событий. Исходными данными для этой задачи являются (1) модель процесса для корректировки и (2) журнал (лог) событий. Необходимость в корректировке модели возникает в том случае, когда разработанная ранее модель процесса уже не соответствует его текущему исполнению. Такая ситуация возникает достаточно часто, так как в ходе жизненного цикла системы в нее могут вноситься изменения.

Для моделирования процессов могут использоваться различные формализмы. В данной работе мы рассматриваем модели в виде классических сетей Петри. В свою очередь, журнал (лог) событий представляется как набор последовательностей записей о событиях, где запись о событии содержит информацию о выполняемом действии и включает отметку о времени его выполнения. Результатом корректировки должна быть новая модель процесса, лучше отображающая реальное поведение ИС, чем исходная модель.

В литературе описаны различные методы автоматического синтеза моделей процессов на основе журналов событий [6,7,10,20,23,25,27,41,42]. Однако синтез модели по журналу событий «с нуля» может дать модель, которую трудно сопоставить с исходной моделью — это затруднит ее понимание и анализ. Поэтому в задаче корректировки нужно не только исправить несоответствие между моделью и реальным поведением системы, но и по возможности сохранить прежнюю структуру модели.

Общая идея подхода, применяемого в данной работе, состоит в корректировке модели *по частям*. Предлагается сначала *декомпозировать* исходную модель на фрагменты, выявить фрагменты, не соответствующие поведению, представленному в логе, и заменить их на новые, исправленные фрагменты. Если изменения в поведении системы являются локальными, и их не много, то большая часть корректируемой модели остается неизменной. Общая схема такого подхода и критерии его применимости были представлены нами в [28].

Данная статья имеет следующую структуру. Общая постановка задачи исправления моделей процессов и иллюстрирующий пример приводятся в разделе 1. Раздел 3. содержит используемые в работе определения и предварительные сведения. В разделе 4. описывается базовый алгоритм корректировки модели путем разбиения ее на фрагменты. Усовершенствованная версия алгоритма представлена в разделе 5.

Раздел 6. содержит результаты экспериментов по исправлению моделей. Обзор других работ по исправлению моделей процессов дан в разделе 2.

# 1. Исправление моделей процессов

Описание постановки задачи по корректировке модели процесса начнем с мотивирующего примера.

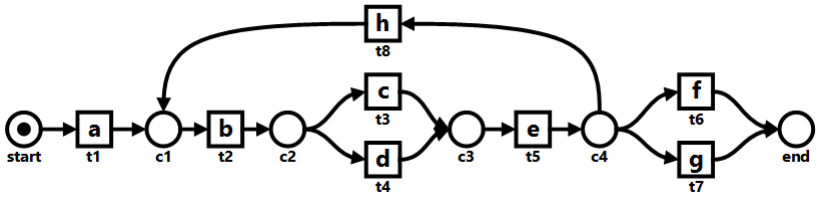


Рис. 1. Пример: обработка запросов на компенсацию затрат *(имена переходов: a = Зарегистрировать запрос; b = Проверка билета; c = Рядовая проверка; d = Экстраординарная проверка; e = Принятие решения; f = Отправка сообщения об отказе; g = Выплата компенсации затрат; h = Повторное рассмотрение запроса)*

Fig. 1. Example: the compensation requests processing *(transition labels: a = Register request; b = Check ticket; c = Examine casually; d = Examine thoroughly; e = Decide; f = Send rejection letter; g = Pay compensation; h = Re-initiate request)*

Рассмотрим модель процесса в виде сети Петри на Рис. 1. Данный пример модели процесса приводится в [3] и часто используется для иллюстрации методов синтеза моделей процессов. Сеть Петри на этом рисунке представляет процесс обработки запросов на компенсацию затрат в некоторой компании. Такая модель может быть, в частности, автоматически синтезирована по следующему логу событий: $[\langle a, b, c, d, e, f, g\rangle, \langle a, b, d, c, e, f, g\rangle, \langle a, b, c, d, e, h, b, d, c, g, f\rangle]$. Этот лог состоит из трех трасс, каждая из которых есть последовательность событий, представленных именами выполняемых действий. Время выполнения событий используется только для их упорядочения и может быть опущено.

Предположим, что данный лог событий был дополнен трассой $\langle a, c, b, e, f, g\rangle$, которая не может быть воспроизведена в модели. В этой новой трассе действие «Проверка билета» $b$ выполняется после действия «Рядовая проверка» $c$, что не соответствует порядку их выполнения в исходной модели на Рис. 1, где любые проверки выполняются строго после проверки билета. Таким образом, исходная модель не соответствует *реальности*, представленной для нас логом событий.

Для того, чтобы получить модель, отражающую реальное исполнение процесса, можно применить два подхода: заново синтезировать модель по логу событий или выполнить корректировку имеющейся модели на основании информации о новом поведении процесса. Второй подход предпочтительней в тех случаях, когда исходная модель имеет определенную *ценность*, например ясную компактную структуру. Пример возможного исправления модели показан на Рис. 2. Большая часть графической структуры исходной модели осталась прежней. Далее мы представим алгоритм, который позволит выполнять подобную корректировку автоматически.

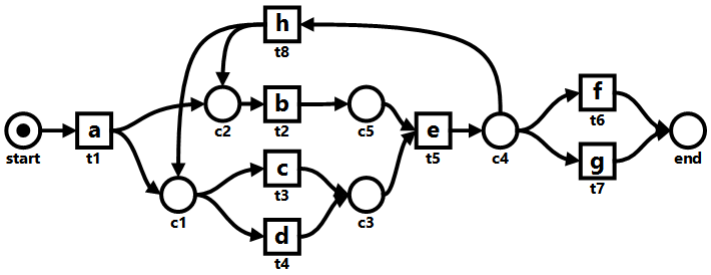


Рис. 2. Пример: модель после корректировки
Fig. 2. Example: the corrected model

**Формулировка задачи исправления модели процесса**. Даны сеть Петри $N$ — это исходная модель процесса, и лог событий $L$, представляющий текущее поведение процесса. Нужно построить сеть Петри $N^r$ такую, что

– модель $N^r$ позволяет выполнить все трассы из $L$ (идеальное соответствие модели и лога);

– модель $N^r$ допускает не «слишком много» исполнений, не представленных в логе (высокая точность модели);

– модели $N$ и $N^r$ имеют похожую графовую структуру (сходство моделей).

## 2. Обзор работ по теме исследования

Идея исправления моделей процессов на основании использования информации из логов событий была предложена в [16] и развита в [17], где авторы предлагают способ повышения степени соответствия (*fitness*) модели процесса в виде сети потоков работ (подкласс сетей Петри) логу событий. Для этого все трассы лога событий разделяются на соответствующие и не соответствующие модели. Трассы, не соответствующие модели, рассматриваются как специальный вариант исполнения, для которого строится модель подпроцесса, встраиваемого затем в исходную модель.

Ещё один способ исправления моделей бизнес-процессов, называемый «автоматизированной коррекцией ошибок», был предложен в [19]. Авторы используют метод симуляции отжига для поиска оптимального набора операций исправления, что позволяет им находить одну или несколько сетей потоков работ, моделирующих заданный процесс без ошибок. Отметим, что авторы не используют при этом информации из логов событий.

А. Поливаный и др. предложили в [31] метод исправления моделей процессов, называемый «impact-driven repair». Метод основан на учете влияния, оказываемого различными операциями исправления на свойства итоговой модели. Этот метод может применяться, когда количество допустимых операций исправления ограничено, а различные операции имеют неодинаковое влияние на ценность итоговой модели. В такой постановке задача исправления сети Петри сводится к оптимизационной задаче. Авторы приводят результаты большого количества экспериментов, произведенных для выбора наиболее эффективного алгоритма.

В [12] предлагается способ усовершенствования *хорошо структурированных* (well-structured) моделей процессов на основе специального генетического алгоритма, в

котором наряду с четырьмя классическими для process mining метриками качества: соответствие (fitness), точность (precision), обобщенность (generalization), простота (simplicity) учитывается также и *сходство* (similarity) синтезируемой модели с исходной.

Основная идея применения принципа *разделяй и властвуй* для синтеза моделей процессов и проверки согласованности модели с логами событий описана в [2, 39]. Практические вопросы применения декомпозиции моделей процессов рассматриваются в [1, 38] и других работах. В них исследуется метод эффективного автоматического синтеза моделей процессов, использующий разбиение лога событий на части.

В работах [21, 22] рассматривается задача так называемой *ре-композиции* (recomposition) и исследуется выбор наиболее подходящего размера фрагмента для задачи автоматического синтеза модели процесса. Декомпозиция модели процесса и лога событий может применяться также для повышения производительности оценки согласованности модели и лога [29, 30, 35]. Применение модульных подходов при решении задач анализа процессов также рассматривалось в некоторых работах, в частности, в [5, 26, 33].

Задача автоматического *упрощения* модели родственна задаче исправления. Части модели, используемые в малом количестве трасс, могут удаляться из модели. Это упрощает модель при некотором снижении соответствия логу событий. Методы упрощения модели предложены, например, в [18, 32]. В [20] также рассматривается вопрос упрощения *гибкой* (fuzzy) модели процесса путём удаления несущественных частей модели с использованием частотных метрик.

Предлагаемый в данной работе метод отличается от подходов, описанных выше. В нашем подходе декомпозиция модели используется для последующей замены некоторых фрагментов. Если расхождения между логом и моделью локальны, то исправленная модель будет отличаться от исходной только замененными фрагментами. Это особенно важно в случаях, когда исходная модель была построена экспертом, а потому хорошо воспринимается человеком. Заново синтезированная модель может быть лишена такого достоинства. Другой особенностью нашего подхода является то, что набор действий процесса не изменяется при исправлении. Это может быть как минусом (если в реальности он изменялся), так и плюсом (не будут добавлены никакие искусственные действия).

## 3. Предварительные сведения

В этом разделе приводятся основные определения и понятия, используемые в работе.

**Мультимножества, функции, последовательности.** Через $\mathbb{N}$ обозначим множество натуральных чисел с нулем. Пусть $S$ — некоторое множество. Мультимножеством над $S$ называется функция $B : S \to \mathbb{N}$, которая сопоставляет каждому элементу из $S$ число его вхождений в $B$. Через $\mathcal{B}(S)$ будем обозначать множество всех мультимножеств над $S$. Конечное мультимножество можно задать перечислением его элементов с указанием их кратности, например, $B = [e, e, e, c, d, d] = [e^3, c, d^2]$ обозначает мультимножество, содержащее три экземпляра элемента $e$, один элемент

$c$ и два экземпляра элемента $d$. Обычные операции над множествами распространяются и на мультимножества естественным образом с сохранением стандартной нотации.

Для функции $f\colon X \to Y$ через $dom(f)$ будем обозначать область ее определения. Нотация $f\colon X \nrightarrow Y$ используется для частичных функций. Для функции $f\colon X \nrightarrow Y$ ее проекция на множество $Q \subseteq X$ определяется как функция $f\restriction_Q\colon Q \nrightarrow Y$ такая, что $\forall x \in Q\colon f\restriction_Q(x) = f(x)$. Определение проекции распространяется и на мультимножества: $[e^3, c, d^2]\restriction_{\{c,d\}} = [c, d^2]$.

Множество всех конечных последовательностей над множеством $X$ будем обозначать через $X^*$. Для записи последовательностей будем использовать треугольные скобки: $\sigma = \langle x_1, x_2, ..., x_n\rangle$ — последовательность длины $n$. Конкатенацию двух последовательностей $\sigma_1$ и $\sigma_2$ будем обозначать как $\sigma_1 \cdot \sigma_2$, а проекцию последовательности $\sigma$ на множество $Q$ — как $\sigma\restriction_Q$.

**Модель процесса.** В данной работе для моделирования процессов используются классические сети Петри, а точнее, более узкий класс моделей — сети потоков работ (WF-сети).

*Сеть Петри* — это тройка $(P, T, F)$, где $P$ и $T$ — непересекающиеся множества позиций и переходов, а $F\colon (P \times T) \cup (T \times P) \to \mathbb{N}$ — функция потока. Через $\bullet t = \{p | F(p, t) \neq 0\}$ будем обозначать множество позиций, входных для перехода $t \in T$, а через $t\bullet = \{p | F(t, p) \neq 0\}$ — множество выходных для $t$ позиций.

Помеченная сеть Петри — это четверка $(P, T, F, l)$, где $l\colon T \to \mathcal{U}_A \cup \{\tau\}$ — функция пометки, ставящая каждому переходу из сети Петри имя действия из некоторого заранее заданного множества $\mathcal{U}_A$. Если $l(t) = \tau$, переход $t$ называют *невидимым*. В противном случае переход $t$ называют *видимым*.

Текущее состояние сети Петри задается *разметкой* $M\colon P \to \mathbb{N}$. Переход $t \in T$ *активен* в разметке $M$ тогда и только тогда, когда $\forall p \in P\colon M(p) \geq F(p, t)$. Активный переход $t$ может *сработать*, изменив разметку сети на новую $M'(p) = M(p) - F(p, t) + F(t, p)$ для каждого $p \in P$. Такое срабатывание перехода будем обозначать как $M \xrightarrow{t} M'$.

**Определение 1.** ***Сеть потоков работ*** *(WF-сеть) — это помеченная сеть Петри $N = (P, T, F, l)$ с выделенными начальной разметкой $M_i$ и конечной разметкой $M_o$, в которой*
*(1) есть ровно одна позиция-исток $i \in P$ такая, что $\bullet i = \emptyset$ и $M_i = [i]$,*
*(2) есть ровно одна позиция-исток $o \in P$ такая, что $o\bullet = \emptyset$ и $M_o = [o]$,*
*(3) каждая позиция или переход $n \in P \cup T$ лежит на пути из $i$ в $o$.*

На Рис. 1 приводится пример сети потоков работ. Позиции изображаются кругами, переходы — квадратами, а начальная разметка — черным токеном в позиции-истоке.

Множество всех видимых переходов в сети $N$ будем обозначать через $T_v(N)$. Будем говорить, что имя перехода *уникально*, если оно помечает в сети ровно один переход. Множество видимых переходов в сети $N$ с уникальными именами будем обозначать через $T_v^u(N)$. Сеть $N^U = N^1 \cup N^2$ — это объединение двух сетей потоков работ, которая строится путем обычного объединения множеств позиций, переходов и отношения потока: $N^1 \cup N^2 = (P^1 \cup P^2, T^1 \cup T^2, F^1 \cup F^2, l^u)$, где $l^u \in (T^1 \cup T^2) \nrightarrow \mathcal{U}_A$

— объединение $l^1$ и $l^2$, $dom(l^u) = dom(l^1) \cup dom(l^2)$, а $l^u(t) = l^1(t)$, если $t \in dom(l^1)$, и $l^u(t) = l^2(t)$, если $t \in dom(l^2) \setminus dom(l^1)$. В данной работе мы рассматриваем только модели с уникальными именами переходов.

**Лог событий.** Множество всех действий процесса обозначим через $A \subseteq \mathcal{U}_A$. Поскольку для построения модели потока работ нам важны только имена действий и их последовательность, будем считать, что событие в логе — это имя действия, и далее будем использовать термины события и (имени) действия как синонимы.

Определим *трассу* $\sigma$ как конечную последовательность событий из $A$, то есть $\sigma \in A^*$. Конечное мультимножество трасс $L$ будем называть *логом событий* или простро *логом*, $L \in \mathcal{B}(A^*)$. Например, $L = [\langle a, b, c, d, e\rangle^3, \langle a, b, a, d, e\rangle^5, \langle a, d, c, b, e\rangle]$ — лог событий с тем же множеством действий, что и модель на Рис. 1. Проекция лога событий $L = [\sigma_1, \sigma_2, \ldots, \sigma_n]$ на множество действий $B$ — это лог событий $L\restriction_B = [\sigma_1\restriction_B, \sigma_2\restriction_B, \ldots, \sigma_n\restriction_B]$, где $\sigma_i\restriction_B$ — проекция трассы $\sigma_i$ на $B$. Такой лог событий будем называть *суб-логом* лога $L$.

**Проверка согласованности.** При оценке качества модели обычно применяются четыре критерия: *соответствие* (fitness), *точность* (precision), *обобщенность* (generalization), *простота* (simplicity) [3]. Два первых критерия используются для измерения степени согласованности модели и лога. Мера соответствия определяет, в какой степени трассы лога могут быть исполнены в модели. Точность определяет, в какой мере модель может порождать поведение, не представленное в логе. Обобщенность и простота характеризуют качество собственно модели. Обобщенность определяет меру абстрактности модели, а простота — ее компактность. Обсуждение смысла и значения указанных критериев можно найти, например, в [3, 11].

При исправлении моделей процессов прежде всего важны критерии соответствия между моделью и логом. Важен также критерий простоты — модель не должна слишком усложняться после ее корректировки. Помимо этого будет оцениваться степень различия между исходной и скорректированной моделями.

Пусть $N$ — сеть потоков работ с именами переходов из $A$, начальной разметкой $[i]$ и конечной разметкой $[o]$, а $\sigma$ — трасса над $A$. Будем говорить, что трасса $\sigma = a_1, \ldots, a_k$ *идеально соответствует* (perfectly fits) модели $N$, если существует последовательность срабатываний переходов сети $[i] = m_0 \xrightarrow{t_1} \ldots \xrightarrow{t_k} m_{k+1} = [o]$ такая, что последовательность действий $\lambda(t_1), \lambda(t_2), \ldots, \lambda(t_k)$ после удаления всех невидимых переходов совпадает с $\sigma$. Лог событий $L$ идеально соответствует сети $N$, если каждая трасса из $L$ идеально соответствует $N$. Например, набор трасс $[\langle a, b, c, e, f\rangle, \langle a, b, c, e, h, b, d, e, g\rangle, \langle a, b, d, e, g\rangle]$ идеально соответствует модели, показанной на Рис. 1.

В литературе представлено достаточно большое количество методов проверки согласованности [4, 8, 9, 13, 15, 29, 30, 40]. В данной работе для оценки соответствия мы используем методику, основанную на выравниваниях [8]. *Выравнивание* — это специальное соответствие между трассой и возможной последовательностью срабатываний переходов сети Петри. Пример двух выравниваний показан на Рис. 3. Верхний ряд каждого выравнивания соответствует трассе из лога событий, тогда как нижний ряд соответствует исполнению модели. Действию в трассе должен соответствовать переход, помеченный этим действием. Если действию в трассе не

удается сопоставить соответствующий переход в исполнении сети, то либо в трассе, либо в исполнении сети добавляется $\gg$ («пропуск»).

| log | a | b | c | d | e | f | g |
|---|---|---|---|---|---|---|---|
| model | a | b | c | d | e | f | g |
| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ |

| log | a | $\gg$ | d | c | b | e | f | g |
|---|---|---|---|---|---|---|---|---|
| model | a | b | d | c | $\gg$ | e | f | g |
| | $t_1$ | $t_2$ | $t_4$ | $t_3$ | | $t_5$ | $t_6$ | $t_7$ |

Рис. 3. Выравнивания (слева: идеальное соответствие; справа: несоответствие)
Fig. 3. Alignments (left: perfectly fitting; right: unfitting)

Левое выравнивание на Рис. 3 представляет идеальное соответствие. В правом выравнивании добавлены два пропуска, так как трасса не может быть исполнена на модели.

Очевидно, что для одной и той же пары трасса — исполнение можно построить различные выравнивания. Хорошее (*оптимальное*) выравнивание — это выравнивание с минимальным числом пропусков. В [8] рассматриваются способы построения оптимальных выравниваний, которые используются для оценки соответствия модели и лога событий.

**Автоматический синтез моделей процессов.** Для исправления фрагментов модели, не соответствующих логу, мы применяем методы автоматического синтеза моделей процессов (process discovery). Достаточно большое число таких алгоритмов с разными характеристиками были описаны ранее, а именно [6,7,10,20,23–25,41,42] и другие.

В данной работе используется метод *индуктивного синтеза* (Inductive Miner) [24]. Этот алгоритм, запущенный с определенными параметрами (в данной работе мы используем настройку *inductive miner incompleteness*), гарантирует идеальное соответствие получающейся модели. Мы также используем для тестирования нашего подхода алгоритм автоматического синтеза, сводящий задачу построения модели к задаче целочисленного линейного программирования (ILP Miner) [41]. Этот алгоритм также гарантирует идеальное соответствие построенной модели логу событий, если запущен с правильными параметрами.

# 4. Корректировка моделей процессов с разбиением на фрагменты

В [28] нами была предложена обобщенная модульная схема исправления моделей процессов, в которой используется подход *разделяй и властвуй*. Исправление модели процесса делится на несколько шагов: (1) декомпозиция, (2) выбор, (3) починка, (4) композиция, (5) оценка. В данной работе мы описываем конкретную реализацию этой схемы, определяемую следующим образом.

Пусть $\mathcal{U}_A$ — множество действий процесса, а $\mathcal{U}_N$ — множество всех помеченных сетей потоков работ, пометки на переходах которых из $\mathcal{U}_A$.

Определим **алгоритм исправления** как процедуру, получающую на вход лог событий $L \in \mathcal{B}(\mathcal{U}_A^*)$ и помеченную сеть потоков работ $N \in \mathcal{U}_N$, и выдающую в каче-

стве результата своей работы сеть Петри $N^r = ModularRepair(L, N)$, которая идеально соответствует логу $L$, т.е. $fitness(L, N^r) = 1$.

В качестве процедурных параметров схемы починки используются:

- $eval \in (\mathcal{B}(\mathcal{U}_A^*) \times \mathcal{U}_N) \to [0; 1]$ — функция оценки, основанная на использовании выравниваний [8],
- $repair \in \mathcal{B}(\mathcal{U}_A^*) \times \mathcal{U}_N) \to \mathcal{U}_N$ — индуктивный алгоритм автоматического синтеза модели процесса [24], либо алгоритм, основанный на решении задачи линейного программирования [41],
- $split \in \mathcal{U}_N \to \mathcal{P}(\mathcal{U}_N)$ — алгоритм максимальной декомпозиции, описанный в [28],
- а $compose \in \mathcal{P}(\mathcal{U}_N) \to \mathcal{U}_N$ — парный ему алгоритм композиции, объединяющий фрагменты сети путем склейки через граничные переходы с одинаковыми пометками.

Неформально алгоритм исправления можно описать следующим образом. Исходная модель декомпозируется на несколько фрагментов. Для каждого фрагмента с помощью проекции получается соответствующий ему суб-лог событий. Каждая пара (подсеть; суб-лог) оценивается с точки зрения соответствия друг другу. Фрагменты, для которых уровень соответствия недостаточен, заменяются с использованием алгоритма автоматического синтеза модели процесса. Результирующая модель получается путём объединения фрагментов по граничным переходам.

В работе [28] описан алгоритм, строящий т. н. «максимальную» декомпозицию, предложенную в [2]. Максимальная декомпозиция предназначена для разбиения сети на подсети следующим образом. Все узлы фрагмента делятся на граничные и внутренние. Внутренними узлами могут быть позиции, невидимые переходы, а также переходы, имеющие в исходной сети одинаковые пометки. Граничными узлами могут быть только видимые переходы с уникальными (в исходной сети) пометками. Максимальной является такая декомпозиция, в которой для каждого из фрагментов выполняется следующее правило. Внутренний узел фрагмента может быть связан дугами либо с граничными узлами фрагмента, либо с аналогичным внутренним узлом, тогда как граничный узел может быть связан только с внутренними узлами одного фрагмента. Все видимые переходы с уникальными (в исходной сети) пометками делятся пополам между двумя или более фрагментами (по ним происходит разбиение). Начальная разметка сети разбивается очевидным образом. Позиция-исток попадает ровно в один из фрагментов разбиения вместе с содержащимся в ней токеном. Максимально декомпозированная сеть может быть собрана (композирована) путём слияния граничных переходов с одинаковыми именами из разных фрагментов. На Рис. 5 показана максимальная декомпозиция модели, изображенной на Рис. 1.

Самое важное свойство максимальной декомпозиции состоит в том, что это *валидная декомпозиция* (valid decomposition). Валидная декомпозиция для сетей Петри определяется в работе [2]. Мы используем версию определения для сетей потоков работ.

**Определение 2.** *Пусть $N \in \mathcal{U}_N$ — сеть потоков работ с функцией пометки $l$. Будем называть $D = \{N^1, N^2, \ldots, N^n\} \subseteq \mathcal{U}_N$ её* ***валидной декомпозицией****, если*

- $N^i = (P^i, T^i, F^i, l^i)$ *— помеченные сети Петри для каждого* $1 \leq i \leq n$,
- $l^i = l\restriction_{T^i}$ *для каждого* $1 \leq i \leq n$,
- $P^i \cap P^j = \emptyset$ *для* $1 \leq i < j \leq n$,
- $T^i \cap T^j \subseteq T^u_v(N)$ *для* $1 \leq i < j \leq n$, *а*
- $N = \bigcup_{1 \leq i \leq n} N^i$.

*Множество всех валидных декомпозиций сети $N$ будем обозначать $\mathcal{D}(N)$.*

Заметим, что максимальная декомпозиция единственна для сети. Более того, можно показать, что это валидная декомпозиция с минимально возможным размером фрагментов (т.е. разбиение *максимально*).

В [2] показано, что процедура проверки соответствия (fitness checking) может быть декомпозирована с использованием любой валидной декомпозиции. Базируясь на этих результатах, мы сформулировали в [28] условия, при которых модульная схема исправления модели процесса обеспечивает идеальное соответствие исправленной модели логу событий.

**Утверждение 1.** *Модульная схема исправления ModularRepair обеспечивает идеальное соответствие, если*

1. *split — функция валидной декомпозиции;*
2. *repair — идеальная функция автоматического синтеза модели, т. е. такая функция, которая для любого лога событий возвращает идеально соответствующую ему сеть Петри;*
3. *compose — функция композиции фрагментов, в которой используется простое слияние переходов с одинаковыми именами.*

Доказательство данного утверждения содержится в [28]. Заметим, что алгоритм исправления полностью удовлетворяет условиям (1), (2) и (3). Действительно,
(1) используется максимальная декомпозиция, являющаяся валидной,
(2) используются алгоритмы автоматического синтеза, которые гарантируют идеальное соответствие конструируемой модели,
(3) для объединения подсетей используется простое слияние переходов.

Рассмотрим пример применения данного алгоритма. Допустим, некоторый лог событий содержит трассы $\langle a, d, b, e, f\rangle$ и $\langle a, b, c, e, g\rangle$, а трасс, в которых $b$ предшествует $d$, в логе нет. Таким образом, $d$ всегда предшествует $b$, если оба события есть в логе. Такой лог не соответствует модели, показанной на Рис. 1, в которой переход с именем $b$ срабатывает строго раньше переходов с именами $c$ и $d$.

В результате применения нашего модульного подхода будет получена модель, показанная на Рис. 4. Данная модель идеально соответствует логу, состоящему из

трасс $\langle a, d, b, e, f\rangle$ и $\langle a, b, c, e, g\rangle$. Результирующая модель уже не является сетью потоков работ, так как содержит дополнительные позиции *s-start* и *s-end*, добавленные алгоритмом автоматического синтеза при замене исправляемого фрагмента. Заметим, что такая модель может быть трансформирована в эквивалентную сеть потоков работ путём добавления искусственных начальной (конечной) позиций, соединенных с позициями *source* и *s-start* (*sink* и *s-end*) через невидимые переходы.

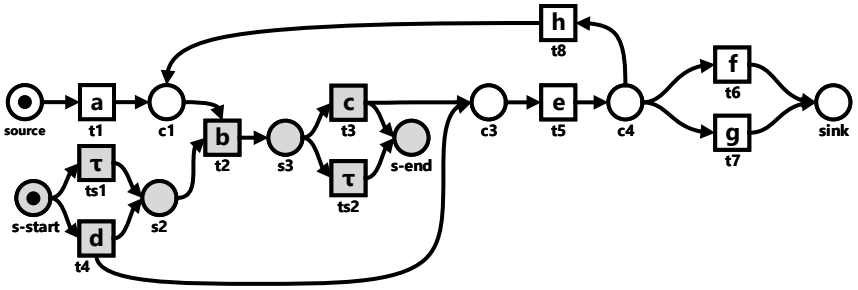


Рис. 4. Исправленная модель
Fig. 4. The repaired model

Позиции *s-start* и *s-end* существенным образом ограничивают поведение исправленной модели. В частности, формируется блокировка (deadlock), не позволяющая исполнять цикл через переход $t_8$ более одного раза. Впрочем, это не противоречит логу событий, по которому исправлялась модель, но, тем не менее, может понизить её ценность. Для решения данной проблемы мы удаляем позиции *s-start* и *s-end* как излишние. Данная операция не изменяет характеристики соответствия модели и лога событий, но сильно увеличивает количество возможных исполнений модели. Действительно, переходы $t_4$ и $ts_1$ теперь могут на любом шаге добавить токен в позиции $s_2$ и $c_3$, а переход $ts_2$ может удалить токен из позиции $s_3$. Таким образом, существенно понижается точность исправленной сети Петри по отношению к логу событий.

Для того, чтобы справиться с данным недостатком, мы предлагаем усовершенствовать наивный алгоритм. В следующем разделе будет описано предлагаемое усовершенствование.

## 5. Усовершенствованный алгоритм исправления модели процесса

Рассмотрим подробнее построение модели для исправляемого фрагмента. Первым делом к исходной модели, показанной на Рис. 1, применяется максимальная декомпозиция, в результате чего получаются шесть фрагментов (см. Рис. 5).

Затем подсчитывается соответствие каждого фрагмента и соответствующего фрагмента лога событий. Оказывается, что ровно один фрагмент $SN3$ не соответствует своему суб-логу. Данный фрагмент заменяется на модель, синтезированную с при-

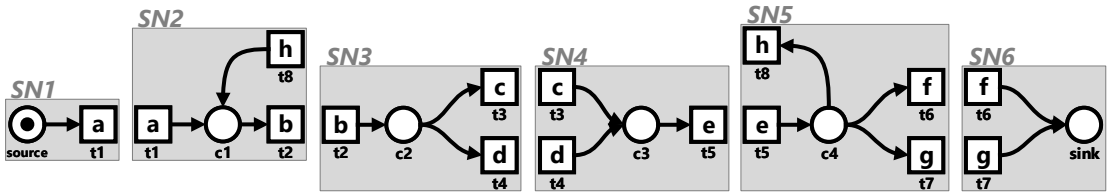


Рис. 5. Максимальная декомпозиция модели, изображенной на Рис. 1
Fig. 5. A maximal decomposition for the model shown in Fig. 1

менением индуктивного алгоритма (см. Рис. 6). Затем все фрагменты композируются в итоговую исправленную модель (см. Рис. 4).

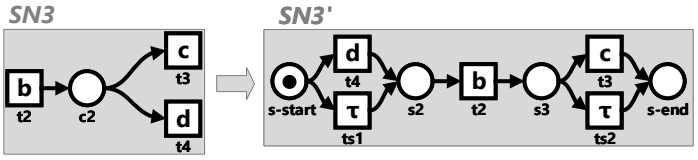


Рис. 6. Замена фрагмента $SN3$ (индуктивный алгоритм)
Fig. 6. The replacement of $SN3$ fragment (inductive miner)

При пристальном рассмотрении примера становится понятно, что в ходе процедуры исправления велась работа в том числе и с граничными переходами исправляемого фрагмента, по которым были разделены фрагменты. В результате причинно-следственные связи на границах фрагмента могут нарушаться, что снижает точность итоговой модели.

В качестве решения данной проблемы мы предлагаем *увеличивать* исправляемый фрагмент. Такое увеличение должно происходить по определенному правилу. К каждому фрагменту, требующему замены, предлагается присоединять все соседние фрагменты, идеально соответствующие своим суб-логам. В результате такого присоединения гарантируется, что внешняя граница изменяемого фрагмента не будет меняться при синтезе сети для замены фрагмента.

Действительно, пусть имеется последовательность срабатываний переходов в сети потоков работ, проходящая через большой фрагмент $m_i \xrightarrow{t_0} \dots \xrightarrow{t_{n-1}} m_n \xrightarrow{t_n} \dots \xrightarrow{t_{n+k}} m_{n+k+1} \xrightarrow{t_{n+k+1}} \dots \xrightarrow{t_{n+k+l}} m_o$, где $t_n, t_{n+1}, \dots, t_{n+k-1}, t_{n+k}$ — переходы, принадлежащие фрагменту, причем $t_n$ и $t_{n+k}$ — граничные для большого фрагмента, а остальные $t_{n+1}, \dots, t_{n+k-1}$ — внутренние. Так как к заменяемому фрагменту были присоединены все соседние фрагменты, то переходы $t_n$ и $t_{n+k}$ в первоначальном разбиении гарантированно принадлежали одному из соседних фрагментов с идеальным соответствием логу. Таким образом, фрагменты последовательности срабатываний $\dots \xrightarrow{t_{n-1}} m_n \xrightarrow{t_n} \dots$ и $\dots \xrightarrow{t_{n+k}} m_{n+k+1} \xrightarrow{t_{n+k+1}} \dots$ также соответствуют логу и не изменятся, если применяемый алгоритм автоматического синтеза гарантирует идеальное соответствие конструируемой модели.

Рис. 7 демонстрирует пример присоединения соседей к фрагменту $SN3$ из рассматриваемого нами примера. С этим фрагментом в декомпозиции граничат только фрагменты $SN2$ и $SN4$, которые и объединяются с ним.

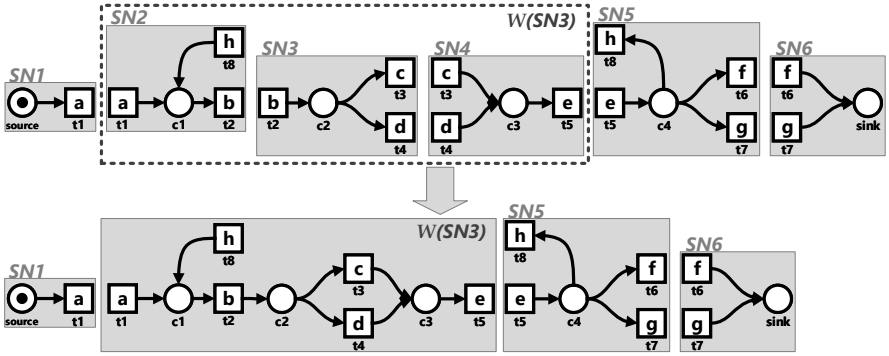


Рис. 7. Присоединение соседей к фрагменту
Fig. 7. Neighbours are connected to the fragment

**Определение 3.** *Пусть $N^i \in D_N$ — это фрагмент декомпозиции $D$ сети потоков работ $N$. Соседними фрагментами для $D$ будем называть фрагменты, в которых имеются переходы, общие с $D$. Множество соседей для фрагмента $N^i$ определим так $\mathcal{N}(N^i) = \{N^j \mid N^j \in D_N \wedge T_i \cap T_j \neq \emptyset\}$.*

Укрупненный фрагмент для фрагмента $N^i$, к которому присоединены все соседние фрагменты, может быть определен так $\mathcal{W}(N^i) = N^i \cup \mathcal{N}(N^i)$.

Определим процедуру укрупнения фрагментов для случая, когда в декомпозиции модели больше одного фрагмента, который требуется заменять при исправлении.

**Определение 4.** *Пусть $D$ — декомпозиция сети потоков работ $N$, и пусть $D = D_b \cup D_c$, где $D_b$ — множество фрагментов сети, не соответствующих своим сублогам, а $D_c$ — множество остальных фрагментов. Очевидно, что $D_b \cap D_c = \emptyset$.* ***Процедура укрупнения фрагментов*** *состоит из следующих шагов:*

1. *Каждой фрагмент из $D_b$ объединяется со своими соседями $\forall N^k \in D_b : D^i_w = \mathcal{W}(N^k)$, $D^i_n = \mathcal{N}(N^k)$; Множество всех укрупненных фрагментов обозначим $D_w = \bigcup\limits_i D^i_w$, а множество всех фрагментов, затронутых процедурой, обозначим $D_n = \bigcup\limits_i D^i_n$; заметим, что множества соседей для каждого фрагмента могут пересекаться;*

2. *Все фрагменты, затронутые процедурой, удаляются из исходной декомпозиции $D_r = D_c \setminus D_n$;*

3. *Собирается новая декомпозиция, содержащая бо́льшие фрагменты: $D^w_N = D_w \cup D_r$.*

Можно показать, что после применения процедуры укрупнения фрагментов получается валидная декомпозиция.

**Утверждение 2.** *Пусть* $D_N = \{N^1, \ldots, N^n\} \in \mathcal{D}(N)$ *— любая валидная декомпозиция сети* $N$*. Пусть* $D_N^w$ *— укрупненная декомпозиция, в которой объединены два фрагмента, то есть* $\exists i, j$ *такие, что* $1 \leq i < j \leq n$ *и* $D_N^w = \{N^i \cup N^j\} \cup D_N \setminus \{N^i, N^j\}$*. Тогда* $D_N^w$ *— валидная декомпозиция, т. е.* $D_N^w \in \mathcal{D}(N)$*.*

*Доказательство.* Необходимо показать, что для $D_N^w = \{N_w^1, \ldots, N_w^{n-1}\}$ выполняются все пять свойств валидной декомпозиции.

(1) Все подсети $N^1, \ldots, N^n$ являются помеченными сетями Петри, $N^i \cup N^j$ — также помеченная сеть Петри по определению операции объединения сетей. Один фрагмент итоговой декомпозиции получается объединением, т. е. $\exists k$ такое, что $1 \leq k \leq n-1$ и $N_w^k = N^i \cup N^j$. Остальные фрагменты не изменяются, т.е. $\forall f$ такого, что $1 \leq f \leq n-1$ и $f \neq k$: $\exists h$ такое, что $1 \leq h \leq n$ и $N_w^f = N^h$.

(2) Для всех подсетей, кроме двух объединяемых, ничего не меняется. При объединении подсетей $N^i$ и $N^j$ по определению $dom(l^{N^i \cup N^j}) = dom(l^i) \cup dom(l^j)$, а $l^{N^i \cup N^j}(t) = l^i(t)$, если $t \in dom(l^i)$, и $l^{N^i \cup N^j}(t) = l^j(t)$, если $t \in dom(l^j) \setminus dom(l^i)$. Так что $l^k = l\restriction_{T_w^k}$ для каждого $1 \leq k \leq n-1$.

(3) Каждая позиция остается в своем фрагменте $P^k \cap P^f = \emptyset$ для $1 \leq k < f \leq n-1$, так как в исходной декомпозиции каждая позиция находилась ровно в одном фрагменте, а множество позиций объединяемых фрагментов — $P^i \cup P^j$, $i < j$.

(4) На границах фрагментов остаются только видимые переходы с уникальными именами. Действительно, при объединении набор граничных переходов уменьшается, $\left\{t \mid t \in T_w^k \cap T_w^f, 1 \leq k < f \leq n-1\right\} \subset \left\{t \mid t \in T^h \cap T^g, 1 \leq h < g \leq n\right\} \subseteq T_v^u(N)$.

(5) $N = \bigcup_{1 \leq h \leq n} N^h = \bigcup_{1 \leq h < i} N^h \cup \bigcup_{i < h < j} N^h \cup \bigcup_{j < h \leq n} N^h \cup N^i \cup N^j = \bigcup_{1 \leq f < k} N_w^f \cup \bigcup_{k < f \leq n-1} N_w^f \cup N_w^k$, что следует из того, каким образом объединялись фрагменты. □

Мы показали, что объединение двух фрагментов не портит валидности декомпозиции. Очевидно, что объединение любого количества фрагментов сводится к последовательному объединению пар фрагментов, причем на каждом шаге валидность получающейся декомпозиции сохраняется. В ходе процедуры укрупнения фрагментов выполняется только объединение некоторых фрагментов. Следовательно, декомпозиция, конструируемая процедурой укрупнения фрагментов, валидна.

Таким образом, базовый (*наивный*) алгоритм исправления может быть усовершенствован путем добавления одного промежуточного шага. После декомпозиции исходной модели и выявления фрагментов, которые подлежат замене, выполняется процедура построения оберток. В результате получается новая декомпозиция, содержащая более крупные фрагменты. Замена фрагментов с использованием алгоритма автоматического синтеза выполняется для этой модифицированной декомпозиции. Такой алгоритм будем называть **усовершенствованным алгоритмом исправления**.

Рассмотрим пример работы усовершенствованного алгоритма. Допустим, что необходимо привести модель, показанную на Рис. 1, в идеальное соответствие логу событий, состоящему из таких трасс: $\langle a, d, b, e, f\rangle$, $\langle a, b, c, e, g\rangle$, $\langle a, b, c, e, h, d, b, e, g\rangle$, $\langle a, c, b, e, f\rangle$. Необходимо заменить фрагмент $SN3$ из максимальной декомпозиции, показанной на Рис. 6. Процедура присоединения соседей к данному фрагменту показана на Рис. 7. Затем с помощью простого проецирования получается суб-лог,

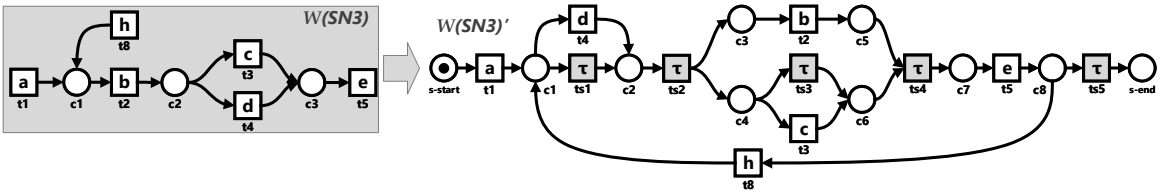


Рис. 8. Замена укрупненного фрагмента $\mathcal{W}(SN3)$ (индуктивный алгоритм)
Fig. 8. The replacement of enlarged fragment $\mathcal{W}(SN3)$ (inductive miner)

соответствующий укрупненному фрагменту: $\langle a, d, b, e\rangle$, $\langle a, b, c, e\rangle$, $\langle a, b, c, e, h, d, b, e\rangle$, $\langle a, c, b, e\rangle$. Для полученного суб-лога с помощью индуктивного алгоритма автоматически строится модель (см. Рис. 8).

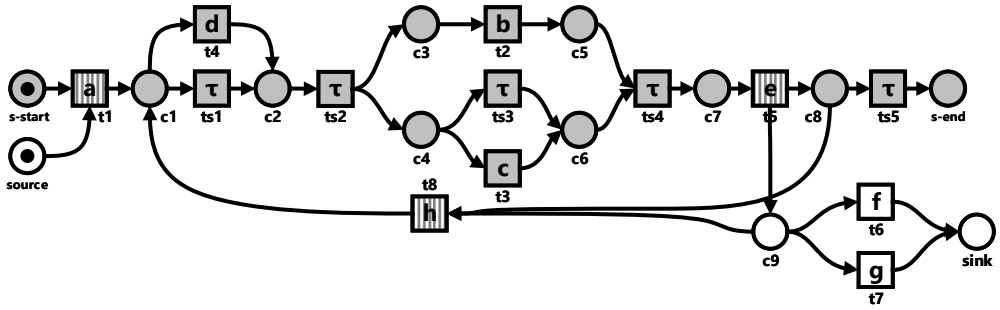


Рис. 9. Исправленная модель (усовершенствованный подход)
Fig. 9. The repaired model (improved approach)

В результате объединения всех фрагментов получается модель, показанная на Рис. 9. Нетрудно проверить, что эта модель идеально соответствует логу событий, который использовался при исправлении. Более того, она точнее отражает лог событий, чем модель, которая могла бы быть получена без укрупнения фрагментов.

Заметим, что позиции *s-start* и *s-end*, добавленные индуктивным алгоритмом автоматического синтеза модели, также могут быть удалены из модели. Более того, отметим, что весь фрагмент, состоящий из позиций $c_8$, *s-end* и невидимого перехода $ts_5$ был добавлен во фрагмент при исправлении только по той причине, что цикл через переход $t_8$ оказался разорван между несколькими фрагментами.

## 6. Экспериментальное тестирование

Описанный подход был протестирован не некотором количестве искусственных примеров. Тестовая установка была реализована в виде расширения для среды *ProM 6 Framework* [37]. Программная архитектура этого расширения и другие детали реализации описаны нами в [34].

С использованием генератора тестовых логов событий Gena [36] автоматически подготовлены идеально соответствующие логи для тестовых сетей потоков работ.

Таблица 1. Некоторые экспериментальные результаты *(Соотв. — соответствие, Точн. — точность, Сходство — сходство моделей, измеренное с помощью расширения* `Calculate Graph Edit Distance Similarity` *для среды ProM [21], N-f — количество фрагментов в декомпозиции, N-bf — количество замененных фрагментов, Размер — количество узлов в сети, Изм. — количество узлов, затронутых изменениями, Время — время работы программы в миллисекундах, тестовая конфигурация:* `Intel Core i7-3630QM, 2.40 ГГц, 4 Гб RAM, Windows 7 x64`*)*

Table 1. Some experimental results *(Similarity is measured with ProM plug-in* `Calculate Graph Edit Distance Similarity` *[21], N-f — the number of fragments in decomposition, N-bf — the number of replaced fragments, Size — the number of nodes in the model, Ch. — the number of nodes in changed fragments, Time — the working time in Milliseconds, test configuration:* `Intel Core i7-3630QM, 2.40 GHz, 4 Gb RAM, Windows 7 x64`*)*

| Модель Model | Метод Method | Алгоритм синтеза Discovery Algorithm | Соотв. Fitness | Точн. Precision | Сходство Similarity | N-f | N-bf | Размер Size | Изм. Ch. | Время Time |
|---|---|---|---|---|---|---|---|---|---|---|
| SM1 | Исходная модель Initial model | | 1 | 0,91 | - | - | - | 40 | - | - |
| | Синтез новой модели New Model Discovery | Ind | 1 | 0,91 | 0,51 | - | - | 47 | 47 | 829 |
| | | ILP | 1 | 0,91 | 0,57 | - | - | 38 | 38 | 10069 |
| SM1-BL-1 | Измененная модель Changed model | | 0,94 | 0,86 | - | - | - | 40 | - | - |
| | Наивный Naive | Ind | 1 | 0,57 | 0,97 | 19 | 1 | 40 | 3 | 2531 |
| | | ILP | 1 | 0,57 | 0,97 | 19 | 1 | 40 | 3 | 2531 |
| | Улучшенный Improved | Ind | 1 | 0,91 | 0,95 | 19 | 1 | 40 | 7 | 2365 |
| | | ILP | 1 | 0,91 | 0,95 | 19 | 1 | 40 | 7 | 2842 |
| SM1-BL-2 | Измененная модель Changed model | | 0,89 | 0,89 | - | - | - | 40 | - | - |
| | Наивный Naive | Ind | 1 | f | 0,89 | 19 | 3 | 45 | 9 | 2983 |
| | | ILP | 1 | 0,5 | 0,94 | 19 | 3 | 39 | 9 | 3215 |
| | Улучшенный Improved | Ind | 1 | 0,91 | 0,82 | 19 | 1 | 47 | 15 | 2426 |
| | | ILP | 1 | 0,91 | 0,88 | 19 | 1 | 41 | 15 | 4030 |
| LM2 | Исходная модель Initial model | | 1 | 0,59 | - | - | - | 133 | - | - |
| | Синтез новой модели New Model Discovery | Ind | 1 | f | f | - | - | 166 | 166 | 2920 |
| | | ILP | 1 | 0,53 | f | - | - | 137 | 137 | 1816898 |
| LM2-BL-1 | Измененная модель Changed model | | 0,98 | 0,59 | - | - | - | 133 | - | - |
| | Наивный Naive | Ind | 1 | 0,28 | f | 63 | 2 | 133 | 7 | 10745 |
| | | ILP | 1 | 0,28 | f | 63 | 2 | 133 | 7 | 9588 |
| | Улучшенный Improved | Ind | 1 | 0,59 | f | 63 | 1 | 133 | 12 | 9205 |
| | | ILP | 1 | 0,59 | f | 63 | 1 | 133 | 12 | 10339 |
| LM2-BL-2 | Измененная модель Changed model | | 0,99 | 0,59 | - | - | - | 133 | - | - |
| | Наивный Naive | Ind | 1 | 0,38 | f | 63 | 1 | 133 | 3 | 8048 |
| | | ILP | 1 | 0,38 | f | 63 | 1 | 133 | 3 | 12847 |
| | Улучшенный Improved | Ind | 1 | 0,59 | 1 | 63 | 1 | 133 | 8 | 12960 |
| | | ILP | 1 | 0,59 | 1 | 63 | 1 | 133 | 8 | 8619 |
| LM2-BL-3 | Измененная модель Changed model | | 0,97 | 0,59 | - | - | - | 133 | - | - |
| | Наивный Naive | Ind | 1 | 0,23 | f | 63 | 3 | 133 | 10 | 8097 |
| | | ILP | 1 | 0,23 | f | 63 | 3 | 133 | 10 | 8855 |
| | Улучшенный Improved | Ind | 1 | 0,59 | f | 63 | 2 | 134 | 21 | 9149 |
| | | ILP | 1 | 0,59 | f | 63 | 2 | 134 | 21 | 12410 |

Затем исходные тестовые модели были немного изменены так, чтобы испортить идеальное соответствие соответствующим логам. Алгоритм исправления тестировался на этих примерах. Некоторые результаты приводятся в Таблице 1.

Модели `SM1` и `LM2` — две тестовые сети потоков работ, состоящие из 40 и 133 узлов соответственно — внесены изменения. В каждом случае переставлены одна или больше пар переходов в исходной сети. Таким образом получены модели `-BL1` и т. д.

Отметим, что в модели специально вносились не слишком существенные изменения, ведь именно для таких случаев и предназначается описываемых подход.

Таблица 1 содержит результаты исправления модели процесса тремя основными способами: синтез новой модели по логу (без использования информации об исходной модели), наивный и улучшенный методы корректировки. Каждый способ опробован для двух алгоритмов автоматического синтеза модели, гарантирующих, что результирующая модель сможет исполнять все трассы из лога — индуктивного `Ind` и использующего целочисленное линейное программирование `ILP`. Обратим внимание, что плагин оценки сходства моделей не всегда может успешно справиться с задачей, по причине высокой ресурсоемкости. Имеющееся количество оперативной памяти (4 Гб) в некоторых случаях недостаточно.

Заметим, что результаты, содержащиеся в Таблице 1, получены с использованием процедуры удаления начальной и конечной позиции для заменяемых фрагментов, описанной в разделе 4.

В данной работе подробно разбиралось исправление нашим методом модели небольшого размера, изображенной на Рис. 1. Это удобно для иллюстративных соображений, но заменяемый фрагмент в таком случае включает большую часть модели. Заметим, что эффективность предлагаемого метода тем выше, чем меньший фрагмент от исходной модели изменяется. Рис. 10 демонстрирует результат работы подхода тестовой модели (`LM2-BL3`). Затронутый процедурой фрагмент выделен цветом. Модель, автоматически синтезированная индуктивным алгоритмом по логу событий, показана на Рис. 11.

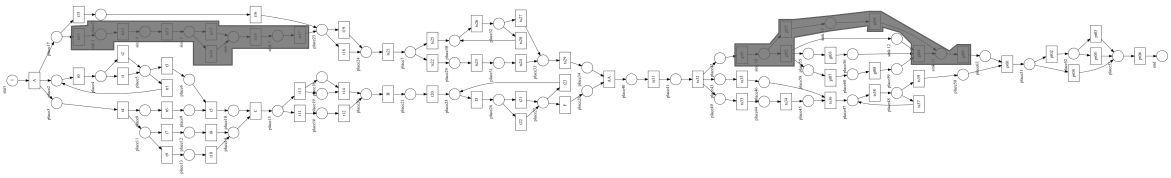

Рис. 10. Исправленная модель `LM2-BL3` (усовершенствованный подход)
Fig. 10. The repaired model `LM2-BL3` (improved approach)

Для всех демонстрационных примеров, результаты работы с которыми представлены в Таблице 1, наш алгоритм демонстрирует ожидаемые результаты. Соответствие *исправленных* моделей логу улучшается до идеального во всех примерах. Точность модели падает при исправлении наивным способом. Более того, в некоторых случаях поведение модели получается настолько разнообразным, что имеющихся 4 Гб оперативной памяти недостаточно для успешной работы реализации алгоритма подсчета точности модели. При применении усовершенствованного способа в рассмотренных случаях достигается точность исходной модели.

Изменяемый фрагмент для всех примеров составил небольшую часть модели. В частности, для самого масштабного из представленных примеров, изменялись 2 фрагмента из 63, содержащие вместе 21 переход и позицию из 133 имеющихся в модели.

Отметим, что для рассматриваемых примеров исправление с использованием алгоритма `ILP` выполняется существенно быстрее, чем построение полностью новой модели даже с учетом затрат времени на декомпозицию и дополнительную провер-

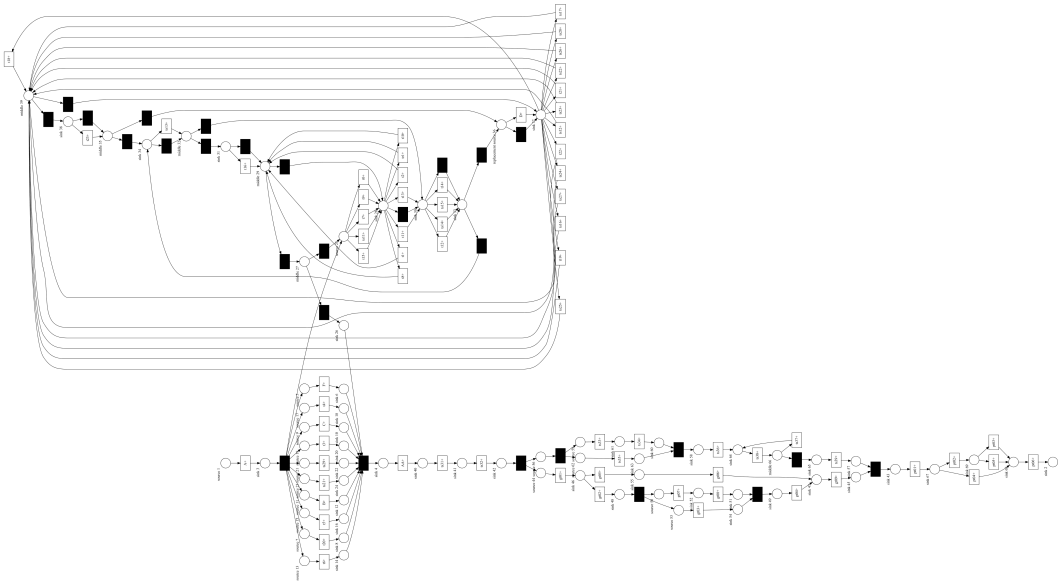

Рис. 11. Новая модель, синтезированная индуктивным алгоритмом по логу событий для модели `LM2`

Fig. 11. The new model discovered using inductive miner from the event log of `LM2` model

ку соответствия. Это ожидаемый результат для такого рода плохо масштабируемых алгоритмов. При использовании индуктивного алгоритма, устроенного иначе, исправление ожидаемо выполняется дольше. Впрочем, оптимизация эффективности не была целью данной работы.

Заметим, что конкретные настройки используемых алгоритмов автоматического синтеза и проверки соответствия модели и лога могут изменить вид получаемой модели. Например, алгоритм проверки соответствия, основанный на выравниваниях, позволяет задавать различную цену штрафа за разные виды несоответствий. Таким образом, некоторые из несоответствий можно игнорировать, если настроить алгоритм соответствующим образом.

Описанный в работе метод корректировки модели не является универсальным. В частности, открытым остается вопрос об исправлении моделей с нелокальными изменениями. При существенном изменении модели автоматический синтез новой модели на основании лога может быть более полезен, чем применение того или иного алгоритма исправления. Как и для поношенной одежды, заплатки работают только до определенного момента.

# Заключение

В данной работе предлагается метод исправления (корректировки) модели процесса, использующий информацию из журнала событий. Метод использует принцип «разделяй и властвуй» для того, чтобы поделить модель на фрагменты с ясными границами. Фрагменты, которые не соответствуют данному журналу событий, заменяются новыми, построенными с помощью алгоритма автоматического синтеза. Затем собирается итоговая исправленная модель, которая имеет сходство с исходной, но соответствует журналу событий.

Кроме формального обоснования корректности метода, работа содержит результаты тестирования метода на нескольких искусственных примерах. Результаты тестирования демонстрируют применимость метода. Особенно полезным он может быть в случаях, когда исходная модель была разработана экспертом и, в результате, легко воспринимается человеком. Применение алгоритма автоматического синтеза может привести к построению корректной, но плохо воспринимаемой модели. Предлагаемый метод исправления меняет только часть модели, в результате чего читаемость сохраняется.

Тем не менее, предложенный метод не является универсальным. Метод хорошо зарекомендовал себя в случае, когда несоответствия локальны, то есть, расхождения с логом могут быть устранены путем замены отдельных фрагментов. В настоящее время авторы работают над усовершенствованным методом исправления, который бы позволял *подстраивать* размер фрагмента разбиения под конкретное несоответствие. Однако не стоит забывать, что если поведение системы изменилось кардинально, то может быть проще построить модель системы полностью заново.

Данная работа продолжает и конкретизирует работу [28], в которой описана общая модульная схема исправления моделей процессов, а именно, представлена одна из возможных реализаций такой схемы. В будущем планируется рассмотреть другие возможные реализации общей схемы на основе других способов декомпозиции модели процесса. В [17] описан метод исправления процессов, позволяющий добавлять в новую модель действия, которые не встречались в исходной модели. Мы планируем исследовать возможности комбинации этого метода с методом, предложенным в данной статье.

## Список литературы / References

[1] van der Aalst W. M. P., “Decomposing Process Mining Problems Using Passages”, Lecture Notes in Computer Science, **7347**, Springer-Verlag, 2012, 72–91.

[2] van der Aalst W. M. P., “Decomposing Petri Nets for Process Mining: A Generic Approach”, *Distributed and Parallel Databases*, **31**:4 (2013), 471–507.

[3] van der Aalst W. M. P., *Process Mining — Data Science in Action, Second Edition*, Springer, 2016.

[4] van der Aalst W. M. P., Adriansyah A., van Dongen B. F., “Replaying History on Process Models for Conformance Checking and Performance Analysis”, *WIREs Data Mining and Knowledge Discovery*, **2** (2012), 182–192.

[5] van der Aalst W. M. P., Bolt A., van Zelst S. J., “RapidProM: Mine Your Processes and Not Just Your Data”, *CoRR*, **abs/1703.03740** (2017).

[6] van der Aalst W. M. P., Kalenkova A. A., Rubin V. A., Verbeek H. M. W., “Process Discovery Using Localized Events”, LNCS, **9115**, Springer, 2015, 287–308.

[7] van der Aalst W. M. P, Weijters A. J. M. M., Maruster L., “Workflow Mining: Discovering Process Models from Event Logs”, *IEEE Transactions on Knowledge and Data Engineering*, **16** (2004), 1128–1142.

[8] Adriansyah A., *Aligning observed and modeled behavior*, Ph.D. Thesis, Technische Universiteit Eindhoven, 2014.

[9] Begicheva A. K., Lomazova I. A., Does Your Event Log Fit the High-level Process Model? *Modeling and Analysis of Information Systems*, **22**:3 (2015), 392–403.

[10] Begicheva A. K., Lomazova I. A., “Discovering High-Level Process Models from Event Logs”, *Modeling and Analysis of Information Systems*, **24**:2 (2017), 125–140.

[11] Buijs J. C. A. M., van Dongen B. F., van der Aalst W. M. P., "On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery", *On the Move to Meaningful Internet Systems: OTM 2012*, Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2012 (Rome, Italy, September 10-14, 2012), Proceedings, Part I, 2012, 305–322.

[12] Buijs J. C. A. M., La Rosa M., Reijers H. A., van Dongen B. F., van der Aalst W. M. P., "Improving Business Process Models Using Observed Behavior", LNBIP, **162**, Springer-Verlag, Berlin, 2013, 44–59.

[13] Burattin A., Maggi F. M., Sperduti A, "Conformance checking based on multi-perspective declarative process models", *Expert Syst. Appl.*, **65** (2016), 194–211.

[14] Dijkman R. M., Dumas M., Garcia-Banuelos L., "Graph matching algorithms for business process model similarity search", Lecture Notes in Computer Science, **5701**, 2009, 48–63.

[15] van Dongen B. F., Carmona J., Chatain T., "A Unified Approach for Measuring Precision and Generalization Based on Anti-alignments", Lecture Notes in Computer Science, **9850**, Springer, 2016, 39–56.

[16] Fahland D., van der Aalst W. M. P., "Repairing Process Models to Reflect Reality", Lecture Notes in Computer Science, **7481**, Springer-Verlag, 2012, 229–245.

[17] Fahland D., van der Aalst W. M. P., "Model repair - Aligning process models to reality", *Inf. Syst.*, **47** (2015), 220–243.

[18] Fahland D., van der Aalst W. M. P., "Simplifying discovered process models in a controlled manner", *Inf. Syst.*, **38**:4 (2013), 585–605.

[19] Gambini M., La Rosa M., Migliorini S., Ter Hofstede A. H. M., "Automated Error Correction of Business Process Models", Lecture Notes in Computer Science, **6896**, Springer, 2011, 148–165.

[20] Günther C. W., van der Aalst W. M. P., "Fuzzy mining: Adaptive process simplification based on multi-perspective metrics", *BPM, Lecture Notes in Computer Science*, **4714** (2007), 328–343.

[21] Hompes B. F. A., *On Decomposed Process Discovery: How to Solve a Jigsaw Puzzle with Friends*, Master's Thesis, Technische Universiteit Eindhoven, 2014.

[22] Hompes B. F. A., Verbeek H. M. W., van der Aalst W. M. P., "Finding suitable activity clusters for decomposed process discovery", *SIMPDA 2014*, **1293** (2014), 16–30.

[23] Kalenkova A. A., Lomazova I. A., van der Aalst W. M. P., "Process Model Discovery: A Method Based on Transition System Decomposition", *Application and Theory of Petri Nets and Concurrency*, Proceedings of 35th International Conference, PETRI NETS 2014, Lecture Notes in Computer Science, **8489**, Springer, 2014, 71–90.

[24] Leemans S. J. J., Fahland D., van der Aalst W. M. P., "Discovering Block-Structured Process Models from Incomplete Event Logs", Lecture Notes in Computer Science, **8489**, Springer, 2014, 91–110.

[25] Leemans S. J. J., Fahland D., van der Aalst W. M. P., "Scalable Process Discovery with Guarantees", LNBIP, **214**, Springer, 2015, 85–101.

[26] Mans R., van der Aalst W. M. P., Verbeek H. M. W., "Supporting Process Mining Workflows with RapidProM", *Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014)*, CEUR-WS.org, 2014, 56–60.

[27] Мицюк А. А., Шугуров И. С., "Синтез моделей процессов по журналам событий с шумом", *Моделирование и анализ информационных систем*, **21**:4 (2014), 181–198; English transl.: Mitsyuk A. A., Shugurov I. S., "On Process Model Synthesis Based on Event Logs with Noise", *Automatic Control and Computer Sciences*, **50**:7 (2016), 460–470.

[28] Mitsyuk A. A., Lomazova I. A., Shugurov I. S., van der Aalst W. M. P., "Process Model Repair by Detecting Unfitting Fragments", *Supplementary Proceedings of AIST 2017, CEUR-WS.org*, 2017.

[29] Muñoz-Gama J., "Conformance Checking and Diagnosis in Process Mining — Comparing Observed and Modeled Processes", *LNBIP*, **270** (2016).

[30] Muñoz-Gama J., Carmona J., van der Aalst W. M. P., "Single-Entry Single-Exit Decomposed Conformance Checking", *Inf. Syst.*, **46** (2014), 102–122.

[31] Polyvyanyy A., van der Aalst W. M. P., ter Hofstede A. H. M., Wynn M. T., "Impact-driven process model repair", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, **25**:4 (2017), 28:1–28:60.

[32] de San Pedro J., Carmona J., Cortadella J., "Log-based simplification of process models", *BPM, Lecture Notes in Computer Science*, **9253** (2015), 457–474.

[33] Шершаков С. А., "Графический язык DPMine для автоматизации экспериментов в области автоматического построения и анализа процессов", *Моделирование и анализ информационных систем*, **21**:5 (2014), 102–115; English transl.: Shershakov S. A., "DPMine graphical language for automation of experiments in process mining", *Automatic Control and Computer Sciences*, **50**:7 (2016), 477–485.

[34] Shugurov I. S., Mitsyuk A. A., "Iskra: A Tool for Process Model Repair", *Proceedings of the Institute for System Programming of the Russian Academy of Sciences*, **27**:3 (2015), 237–254.

[35] Shugurov I. S., Mitsyuk A. A., "Applying MapReduce to Conformance Checking", *Proceedings of the Institute for System Programming of the Russian Academy of Sciences*, **28**:3 (2016), 103–122.

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

[37] 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 Track 2010*, **615**, CEUR-WS.org, 2010, 34–39.

[38] Verbeek H. M. W., "Decomposed Process Mining with Divide And Conquer", *BPM 2014 Demos*, **1295** (2014), 86–90.

[39] Verbeek H. M. W., van der Aalst W. M. P., Muñoz-Gama J., "Divide and Conquer: A Tool Framework for Supporting Decomposed Discovery in Process Mining", *The Computer Journal*, 08.05.2017. https://doi.org/10.1093/comjnl/bxx040.

[40] Weber I., Rogge-Solti A., Li C., Mendling J., "CCaaS: Online Conformance Checking as a Service", *BPM 2015 Demos*, **1418** (2015), 45–49.

[41] van der Werf J. M. E. M., van Dongen B. F., Hurkens C. A. J., Serebrenik A., "Process Discovery using Integer Linear Programming", *Fundam. Inform.*, **94** (2009), 387–412.

[42] Weijters A. J. M. M., Ribeiro J., Chawla N., "Flexible Heuristics Miner", *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, 2011, 310–317.

---

**Mitsyuk A. A., Lomazova I. A., van der Aalst W. M. P.**, "Using Event Logs for Local Correction of Process Models", *Modeling and Analysis of Information Systems*, **24**:4 (2017), 459–480.

**DOI:** 10.18255/1818-1015-2017-4-459-480

**Abstract.** During the life-cycle of an Information System (IS) its actual behaviour may not correspond to the original system model. However, to the IS support it is very important to have the latest model that reflects the current system behaviour. To correct the model, the information from the event log of the system may be used. In this paper, we consider the problem of process model adjustment (correction) using the information from an event log. The input data for this task are the initial process model (a Petri net) and the event log. The result of correction should be a new process model, better reflecting the real IS behavior than the initial model. The new model could be also built from scratch, for example, with the help of one of the known algorithms for automatic synthesis of the process model from an event log. However, this may lead to crucial changes in the structure of the original model, and it will be difficult to compare the new model with the initial one, hindering its understanding and

analysis. It is important to keep the initial structure of the model as much as possible. In this paper, we propose a method for process model correction based on the principle of "divide and conquer". The initial model is decomposed in several fragments. For each fragment its conformance to the event log is checked. Fragments which do not match the log are replaced by newly synthesized ones. The new model is then assembled from the fragments via transition fusion. The experiments demonstrate that our correction algorithm gives good results when it is used for correcting local discrepancies. The paper presents the description of the algorithm, the formal justification for its correctness, as well as the results of experimental testing by some artificial examples.

**Keywords:** process mining, process model repair, Petri nets, process model decomposition, divide and conquer

**On the authors:**
Alexey A. Mitsyuk, orcid.org/0000-0003-2352-3384, research fellow,
National Research University Higher School of Economics, Laboratory of Process-Aware Information Systems,
20 Myasnitskaya St., Moscow 101000, Russia, e-mail: amitsyuk@hse.ru

Irina A. Lomazova, orcid.org/0000-0002-9420-3751, doctor of sciences, professor
National Research University Higher School of Economics, Laboratory of Process-Aware Information Systems,
20 Myasnitskaya St., Moscow 101000, Russia, e-mail: ilomazova@hse.ru

Wil M.P. van der Aalst, orcid.org/0000-0002-0955-6940, PhD, dr.ir., professor
Eindhoven University of Technology (TU/e), Architecture of Information Systems,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands, e-mail: w.m.p.v.d.aalst@tue.nl

**Acknowledgments:**
This work is an output of a research project implemented as part of the Basic Research Program at the National Research University Higher School of Economics (HSE).