ОГЛАВЛЕНИЕ

АННОТАЦИЯ

Одним из важнейших этапов жизненного цикла программного обеспечения является этап тестирования. Применяющиеся на практике техники создания тестовых комплектов подразумевают использование полноценного набора требований к тестируемому программному продукту. Однако достаточно часто имеют место ситуации, когда актуальная и подробная документация, описывающая поведение системы, отсутствует. Это может быть обусловлено особенностями архитектуры рассматриваемой информационной системы, упущениями в организации процесса разработки и так далее.

В настоящей работе предлагается способ решения проблемы тестирования в условиях отсутствия набора спецификаций путем адаптации методов process mining для построения тестовой модели системы. Осуществляется обоснование принципов извлечения канонической модели из системного журнала событий, а также алгоритмов ее преобразования, включающих механизмы фильтрации от шума, обобщения и генерации тестовых кейсов.

Для реализации подхода было разработано приложение на языке программирования Java, позволяющее осуществлять построение тестовых моделей путем анализа логов событий в различных используемых на практике форматах. Работа программы была проверена на журналах событий, выгруженных с реальных систем.

ВВЕДЕНИЕ

В настоящее время значительную долю информационных систем (ИС) образуют приложения, созданные по принципам распределенной архитектуры. В частности, широкое распространение получили сервис-ориентированные распределенные приложения. Среди них – системы электронного банкинга, бронирования билетов на авиарейсы, различные клиент-серверные решения в области маркетинга. Ключевая особенность ИС такого класса заключается в том, что в качестве компонентов системы выступают слабо связанные между собой сервисы, взаимодействие между которыми, как правило, реализуется механизмом обмена сообщениями. Последний строится на основе унифицированных протоколов, таких как SMTP или HTTP.

По ряду объективных причин [1, с. 3-5], в последние годы микросервисная парадигма организации программных комплексов является одним из преобладающих на практике вариантов реализации сервис-ориентированной архитектуры. Таким образом спроектированные приложения в качестве компонентов содержат программные модули, выполняющие, как правило, достаточно тривиальные функции по сравнению с задачами, решаемыми системой в целом. Поддержка отдельных узлов комплекса представляет собой достаточно несложный процесс по причине простоты организации каждого компонента, что в итоге обусловливает высокую степень масштабируемости систем подобного класса.

Одним из нежелательных свойств распределенных приложений является разрозненность документации системы [2]. Зачастую описание целостного бизнес-процесса, реализуемого программным продуктом, состоит из набора спецификаций, имеющих отношение к отдельным компонентам, или же вообще отсутствует. В таком случае значительно затрудняется проведение тестирования системы, которое в условиях наличия документации

подразумевает написание и выполнение тестовых кейсов, составленных в соответствии с имеющимся набором функциональных требований к программному обеспечению (сценарное тестирование, [17]). Среди наиболее чувствительных к неполноте спецификаций видов тестирования выделяется регрессионное тестирование[1], представляющее собой один из важнейших методов динамической верификации ПО. Стоит отметить, что именно регрессионное тестирование крайне востребовано в области микросервисных приложений ввиду масштабируемости последних [26, с. 2].

Проблема отсутствия описания ожидаемого поведения тестируемой системы может быть решена путем применения методов process mining (интеллектуального анализа процессов) для построения модели одного или нескольких процессов, выполняющихся в рамках приложения. В основе process mining лежит идея использования журналов событий, сгенерированных в ходе функционирования системы, для обнаружения, мониторинга и оптимизации реальных бизнес-процессов [3, с. 8-9].

Предлагаемый в данной работе подход подразумевает применение извлеченной модели для генерации данных, служащих для составления тестовых наборов для процессов рассматриваемой системы.

Актуальность использования подобного метода определяется тем, что на текущий момент значительная доля ИС поддерживает автоматическое ведение системного журнала. Согласно [4], методы интеллектуального анализа процессов применимы к программным решениям в сферах WFM (workflow management), CRM (customer relationship management), ERP (enterprise resource planning), B2B (business to business).

Журналы (логи, трассы) событий, выгруженные с реальных систем, обладают рядом особенностей. Во-первых, на текущий момент нет соглашений, обязывающих разработчиков использовать некий

[1] Регрессионное тестирование (regression testing) – разновидность тестирования; проверка того, что изменения, внесенные в программное обеспечение, не добавили новых дефектов в уже протестированные участки исходного кода.

унифицированный формат в логах событий. Таким образом, на практике приходится сталкиваться с разнообразием форматов системных журналов. Во-вторых, данные, содержащиеся в логе, могут быть определенным образом зашумлены, что представляет серьезную проблему в случае задействования классических алгоритмов process mining для анализа имеющихся данных.

В ходе обзора литературы было установлено, что проблема применения методов process mining для получения тестовой модели системы является достаточно малоизученной. Так, в [6] описывается процесс генерации тестовых кейсов, однако освещенные выше особенности трасс, собранных с реальных систем, остаются за пределами рассмотрения.

В качестве цели настоящей работы выступает разработка метода автоматизированного построения тестовой модели процесса системы на основе анализа трассы событий.

Объектом исследования являются трассы событий, порождаемые распределенным приложением; предметом – возможность извлечения тестовой модели процесса, выполняемого рассматриваемой системой, из системного лога с помощью методов process mining.

С теоретической стороны, в текущей работе прежде всего обосновывается каноническая модель исследуемой системы, после чего описываются алгоритмы, использующиеся для модификации полученной модели и генерации тестовых случаев.

Адаптация методов интеллектуального анализа процессов для построения тестовой модели системы реализуется в виде программного приложения на языке программирования Java с использованием дополнительных утилит и библиотек для внутренней организации и визуализации данных.

РАЗДЕЛ I
ПОСТАНОВКА ЗАДАЧИ

Исходя из сформулированной цели, в данной работе выдвигался следующий ряд задач:

- выделение оптимальной канонической математической модели процесса в системе;
- реализация адаптеров преобразований журналов событий в общеупотребительных форматах в унифицированное представление данных;
- разработка функционала, обеспечивающего преобразование и визуализацию модели процессов системы, а также экспорт извлеченных тестовых случаев.

Далее необходимо привести определенные сведения, которые обеспечат ясность последующего изложения.

Под событием в системном логе L понимается запись о некотором зафиксированном взаимодействии между компонентами системы. Согласно определению, данному в [3, с. 100], событие характеризуется набором именованных атрибутов. Пусть $\hat{E}$ – пространство всевозможных событий, A_1 – множество наименований атрибутов. Введем функцию, отображающую событие на значение атрибута:

$$\xi_a(e) = proj_a(e),$$

где $e \in \hat{E}$ – некоторое событие, $a \in A_1$ – название атрибута из множества допустимых наименований A_1. Под обозначением $proj_a(e)$ понимается проекция события на указанный в качестве параметра атрибут. Например, событие может быть задано в виде набора пар:

$$e = \{(id, 128), (action, \text{установка соединения})\}.$$

В таком случае $\xi_{id}(e) = 128$. Стоит заметить, что для всех событий, представленных в логе L, множество A_1 выбирается в единственном

экземпляре, на чем строится последующий процесс анализа L. Для случая, когда событие e не обладает атрибутом a, используется следующая запись: $\xi_a(e) = \perp$.

Далее введем определение кейса процесса. Под кейсом (случаем) процесса в трассе L понимается набор записей об отдельном варианте исполнения некоторого фиксированного процесса. К примеру, упрощенный бизнес-процесс, описывающий принятие банком решения о кредитовании клиента, может быть исполнен в двух различных экземплярах: (оценка кредитных рисков – выдача ссуды) и (оценка кредитных рисков – сообщение об отказе). Формально [3, с. 104], кейс c – элемент множества всевозможных кейсов $\hat{C}$. Как и в случае событий, вводится в рассмотрение множество A_2 допустимых имен атрибутов, среди которых обязательным для каждого кейса является атрибут trace. Так, $\xi_{trace}(c) = \sigma$, где $\sigma \in \hat{E}^*$ – конечная последовательность событий, формирующих кейс c. На нее накладываются ограничения уникальности входящих в нее элементов, то есть

$$\forall i, j: 1 \leq i < j \leq |\sigma| \;\; \sigma(i) \neq \sigma(j),$$

где $\sigma(k)$ – элемент σ с номером k.

В результате, системный лог L представляет собой некоторое множество кейсов одного или нескольких процессов: $L \subseteq \hat{C}$. Каждое событие e ассоциировано с некоторым случаем процесса.

С учетом введенных определений первая из задач настоящей работы трактуется следующим образом. Требуется выделить такую модель исследуемой системы, которая:

- содержит достаточный для работы алгоритмов модификации и экспорта тестовых данных объем информации о процессах в системе;
- задействует минимальное число атрибутов событий и кейсов процессов с тем, чтобы к такой модели могло быть сведено как можно большее число трасс в различных исходных форматах.

При этом «оптимальность» канонической модели рассматривается с точки зрения удовлетворения соотношению «полнота – универсальность». То есть необходимо выбрать такие множества A_1 и A_2, чтобы извлеченные из лога L случаи процессов могли быть использованы для последующего построения тестовой модели системы, и максимальное число логов в различных форматах могло быть задействовано в качестве входных данных.

Задача реализации адаптеров преобразований журналов событий в каноническую модель подразумевает программную реализацию средств для перевода системных логов в форматах XES, MXML, log4j2 во внутреннее унифицированное представление данных. В данном контексте под внутренним представлением понимается каноническая модель трассы, разработанная на предыдущем этапе работы.

Наконец, задача реализации алгоритмов модификации, визуализации и экспорта абстрактных тестовых кейсов включает в себя разработку эффективных подходов к обработке извлеченной из трассы событий информации: очистке извлеченной информации от шума, выделению классов эквивалентности обнаруженных кейсов процессов, обобщению модели, построению репрезентации модели в виде графа и генерации результирующих данных из полученной тестовой модели системы.

РАЗДЕЛ II
ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ ЗАДАЧИ

2.1. Тестирование при отсутствии полного набора спецификаций

Основная проблема, решаемая в настоящей работе, – проблема проведения тестирования системы в условиях, когда полноценная и актуальная документация, описывающая конкретные требования к исследуемому программному обеспечению, недоступна. Подобная ситуация может возникать в связи с рядом причин.

Во-первых, как уже было упомянуто, набор спецификаций, имеющих отношение к модулям распределенной системы, может фактически оказаться разрозненным, что приводит к невозможности целостного анализа имеющихся требований. Такая практика является достаточно распространенной, поскольку отдельные компоненты подобных систем зачастую разрабатываются и модифицируются независимо друг от друга, откуда следуют возможные несоответствия в структуре и особенностях работы модулей программного обеспечения.

Во-вторых, согласно [25, с. 1], программные системы, характеризующиеся длительными этапами проектирования и реализации функционала, на момент внедрения сталкиваются с уже некоторым образом эволюционировавшими требованиями к разработанному продукту со стороны заказчика. К тому же, лишь в редких случаях набор спецификаций, сформулированный на начальной стадии разработки, формально документируется и поддерживается актуальным.

Проблема создания тестовых наборов при отсутствии полноценной документации на практике может решаться путем предварительного сбора сведений о системе [27, с. 6-7] в результате получения информации от разработчиков, аналитиков и пользователей рассматриваемого программного обеспечения и использования подобных данных для составления модельного

представления программного продукта. Помимо этого, может быть задействовано исследовательское тестирование (exploratory testing, [29]) для получения тестировщиком более полной информации относительно свойств ПО. Однако, описанный ad-hoc подход не является формализованным – приведенные принципы исследования поведения системы не гарантируют получения полного представления о процессах, выполняющихся в тестируемом приложении, и не подкрепляются теоретически.

Иным способом решения рассматриваемой задачи предстает разработка тестового комплекта на основе набора спецификаций, полученного в результате реверс-инжиниринга требований [25, 28]. Обратная разработка в данном контексте описывается как процесс, противоположный формированию требований (requirement engineering), то есть представляет собой деятельность по извлечению спецификаций из поведения системы, фиксируемого средствами мониторинга. Так, в [28, с. 2-3] предлагается организация реверс-инжиниринга требований путем получения кода программы, служащего в дальнейшем в качестве основы для выделения спецификаций системы. Схематически данный процесс приведен на рис. 1.

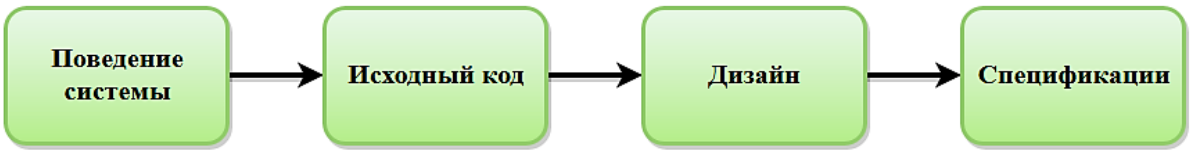

Рис. 1. Схематическое представление предлагаемого в [28] подхода к реверс-инжинирингу требований

Стоит заметить, что извлечение спецификаций системы на основе исходного кода программы – задача достаточно трудоемкая, что, к тому же, дополняется отсутствием гарантий того, что искомый набор требований в конечном счете будет полным. Это, в свою очередь, привносит дополнительные риски в процесс организации тестирования на основе полученных таким образом требований.

2.2. Применение process mining для тестирования

Более перспективной альтернативой уже рассмотренным подходам выступает построение с помощью методов интеллектуального анализа процессов модели поведения исследуемой системы, применимой для составления тестового набора [2, 6]. В основе данного метода лежит идея извлечения модельного представления тестируемого приложения из системного журнала, включающего в себя записи об автоматически зафиксированных в ходе работы системы событиях. При таком подходе потребность в формальных требованиях при составлении тестов нивелируется тем, что в модели содержатся варианты фактического исполнения процессов системы, которые служат основой при создании тестовых кейсов.

В [6] описывается применение process mining для отслеживания различных вариантов взаимодействия пользователя с интерфейсом программы. В данной статье приводится два базовых подхода к генерации тестовых случаев.

Первый из них заключается в порождении тест-кейсов путем использования зафиксированных в модели рабочих процессов пользователя (то есть типичных вариантов последовательностей выполняемых человеком действий). При этом выдвигается требование к степени детализации зарегистрированных в логе интеракций, представляющее собой условие учета различных параметров системы при фиксации события взаимодействия.

Второй подход подразумевает задействование полученной в ходе анализа модели при порождении вероятностных тестов, использующих случайным образом сгенерированные значения в качестве параметров. Подобная стратегия является менее эффективной, поскольку с практической точки зрения вероятностные тесты не способны обнаруживать сложные ошибки, являющиеся результатом сочетания не связанных между собой факторов [16, с. 81-82].

Однако стоит заметить, что в [6] не рассматриваются описанные ранее особенности журналов событий, полученных с реальных систем. Наиболее существенной из них является наличие зашумленных данных в анализируемой трассе, что в случае задействования неспециализированных алгоритмов process mining может привести к искаженной модели поведения системы и, следовательно, к некорректно сгенерированному на ее основе набору тестов. Подобная проблема решается в настоящей работе путем очистки извлеченной модели.

Что касается непосредственной процедуры, обеспечивающей использование модели системы для организации тестирования, дельта-анализ, приведенный в [5], с теоретической стороны представляет собой состоятельный подход, позволяющий осуществить проверку соответствия модели фактической и эталонной версий системы. Процесс тестирования в таком случае может быть выполнен по следующему алгоритму:

1. извлечение трассы событий из системы версии A;
2. построение канонической модели поведения системы;
3. применение алгоритмов фильтрации и обобщения модели;
4. составление тестовых кейсов на основании модели;
5. выполнение тестов на системе версии B;
6. повторение 1-3 для порожденной тестированием трассы событий;
7. анализ покрытия – сравнение моделей систем версий A и B.

Достоверность (степень соответствия реальному поведению) получаемой тестовой модели определяется рядом факторов, среди которых существенным выступает полнота исходного лога событий: не все имеющиеся в системе процессы могут быть зафиксированы в трассе. Это в общем случае создает ограничения для предлагаемого подхода. Впрочем, на практике приходится сталкиваться с достаточно объемными журналами событий, в которых был запечатлен длительный период работы системы, что в итоге позволяет извлекать вполне достоверные модели.

2.3. Выводы по разделу

В данном разделе были рассмотрены основные подходы, позволяющие в условиях отсутствия полноценной и актуальной документации системы провести тестирование программного обеспечения. Описанные стратегии восстановления требований к системе путем применения exploratory testing и реверс-инжиниринга не гарантируют получения полного набора спецификаций, предназначенных для создания на их основе тестового комплекта.

Подход, основанный на интеллектуальном анализе процессов, представляет собой гораздо более эффективный метод, главными преимуществами которого являются высокая степень автоматизации процесса тестирования и возможность применения к широкому классу программных систем. При таком подходе на трассу событий, являющуюся источником информации о системе, накладываются определенные ограничения, как-то: возможность восстановления последовательности событий, отнесения каждого события к определенному процессу или кейсу процесса системы. Тем не менее, значительная доля трасс информационных систем, поддерживающих автоматическое ведение журнала событий, удовлетворяет вышеупомянутым требованиям, что обусловливает практическую применимость рассмотренного метода.

РАЗДЕЛ III
ИССЛЕДОВАНИЕ И ПОСТРОЕНИЕ РЕШЕНИЯ ЗАДАЧИ

3.1. Выделение канонической модели

Формирование унифицированного представления изучаемого в ходе анализа трассы событий процесса служит для того, чтобы операторы, задаваемые в дальнейшем для преобразования имеющихся данных, имели в качестве операндов объекты, представляемые единообразно.

С одной стороны, как уже было указано в разделе «Постановка задачи», каноническая модель должна быть спроектирована таким образом, чтобы как можно большее количество логов событий в различных форматах могло быть к ней сведено. Под сведением подразумевается некоторая (возможно, состоящая из нескольких стадий) операция f, отображающая журнал L в модель M.

С математической точки зрения, к отображению f предъявляется требование инъективности. Пусть $\hat{L}_F$ представляет собой множество всех логов событий формата F, $\widehat{M}$ – множество всех моделей. Тогда отображение сведения f должно удовлетворять условию:

$$\forall m \in f\left(\hat{L}_F\right) \subseteq \widehat{M} \;\; \exists! \, l \in \hat{L}_F \colon f(l) = m. \qquad (3.1)$$

Фиксация формата F (т. е. способа записи лога событий) при выборе трассы l в (3.1) существенна, поскольку условие инъективности может быть нарушено, если в рассматриваемое множество $\hat{L}_F$ войдут, по крайней мере, две эквивалентных трассы. Логи событий l_1 и l_2 назовем эквивалентными, если они семантически тождественны, то есть содержат информацию о событиях, имеющих отношение к одному и тому же множеству кейсов C.

Для достижения универсальности модели требуется использование минимального объема информации из лога в некотором условном формате F. По утверждению Вил ван дер Аалста [3, с. 12], минимальными требованиями к событиям в журнале для применения методов process mining являются:

- возможность отнести данное событие к некоторому кейсу процесса;
- возможность извлечь из события информацию о произведенной в рамках события деятельности;
- упорядоченность представленных в логе событий в рамках одного кейса.

Приведенные требования выделяются в качестве отправной точки в процессе выделения оптимальной модели системы. Следует рассмотреть формализацию описанных условий.

Первые два требования к контенту журнала событий интерпретируются как гарантированное наличие у событий двух различных атрибутов: *activity* (деятельность; некоторое зафиксированное в рамках события действие) и *caseid* (идентификатор кейса процесса). То есть множество A_1 допустимых наименований атрибутов событий имеет в качестве подмножества набор $\{activity, caseid\}$.

Требование упорядоченности событий в рамках кейса накладывает следующее условие:

$$\forall e_1, e_2 \in \sigma_c : e_1 \neq e_2 \;\; e_1 \prec e_2 \oplus e_2 \prec e_1,$$

где $\sigma_c = \xi_{trace}(c)$ – последовательность событий в рамках кейса c, $\oplus$ – исключающее «или», $\prec$ – заданное на множестве событий отношение порядка. Естественной словесной формулировкой введенного отношения будет утверждение «предшествует». Стоит заметить, что свойствами отношения $\prec$ (обозначим дополнительно как R) являются:

- антирефлексивность: $(e, e) \notin R$;
- асимметричность: $(e_1, e_2) \in R \Rightarrow (e_2, e_1) \notin R$;
- транзитивность: $(e_1, e_2) \in R \wedge (e_2, e_3) \in R \Rightarrow (e_1, e_3) \in R$.

Наконец, следует понимать $\prec$ как отношение строгого линейного порядка, поскольку оно задается на всех событиях в рамках σ_c для каждого кейса $c \in L$.

Помимо перечисленных требований, к событиям в трассе в рамках данной работы было также добавлено дополнительное условие включения в набор атрибутов временной метки события. Это означает, что к множеству допустимых наименований атрибутов A_1 добавляется $\{timestamp\}$.

Решение подобного рода было основано на ряде соображений, сформировавшихся из собственных практических наблюдений и анализа работ экспертов в рамках обработки содержащейся в системных логах информации.

Так, в [7, с. 7] приводится набор основных характеристик, присущих файлам системных журналов:

– в течение всего времени работы программы в лог постепенно осуществляется запись строк (или групп строк), содержащих требующую фиксации информацию; при этом уже записанные данные никоим образом не модифицируются и не удаляются;
– каждая запись в журнале инициируется некоторым событием в системе, например, взаимодействием с пользователем, вызовом функции или процедурой ввода-вывода;
– записи в логе зачастую определенным образом параметризованы, т. е. несут некоторую дополнительную информацию;
– события, содержащиеся в журнале, часто в качестве одного из атрибутов имеют временную метку; это не является строгим правилом, однако журналы без таковых характеристик встречаются на практике достаточно редко.

В дополнение к описанному, в [8] указывается ряд элементов, как правило, встречающихся в записях журналов событий, порожденных в ходе функционирования веб-сервера. Среди них – имя пользователя, путь к посещаемому веб-ресурсу, временная метка, URL (унифицированный указатель ресурса), тип используемого HTTP-метода.

Как видно из представленного обзора применяющихся на практике атрибутов записей в системных журналах, уже упоминавшееся поле

временной метки события (*timestamp*) с большой долей вероятности будет находиться среди атрибутов событий журналов различных приложений. Помимо этого, наличие данного атрибута представляет собой возможность его использования в качестве дополнительной информации для алгоритмов очистки исследуемой трассы от аномалий, возникающих в связи с особенностями формирования системных логов в реальных условиях (подробнее – в подразделе 3.3).

Таким образом, на данном этапе построения решения задачи предлагается формирование такой модели события, которая с минимальным необходимым набором атрибутов будет пригодна для применения методов дальнейшей обработки содержащихся в параметрах события данных. Будем под каноническим событием (КС) понимать событие e, ассоциированное с множеством допустимых наименований A_1, которое имеет следующий вид: $A_1 = \{eventid, caseid, activity, timestamp\}$.

При этом $\xi_{eventid}(e) = n \in \mathbb{N} \cup \{0\}$ – порядковый номер события в последовательности σ_c, где кейс c однозначно определяется уникальным идентификатором $\xi_{caseid}(e)$. Порядковый номер устанавливается согласно предварительно заданному отношению порядка на всех $e \in \sigma_c$.

Перейдем, наконец, к освещению структуры канонической модели системы. Системный лог L в общем случае содержит информацию о выполнении некоторого множества P процессов приложения. Модель процесса $p \in P$ строится как совокупность кейсов, соответствующих этому процессу. Кейсы же, в свою очередь, характеризуются последовательностями КС, обозначаемыми как σ_c. Так описанная иерархическая структура агрегации данных о процессах исследуемой системы в конечном счете образует ее каноническую модель. Принципиальное устройство введенной модели системы проиллюстрировано на рис. 2.

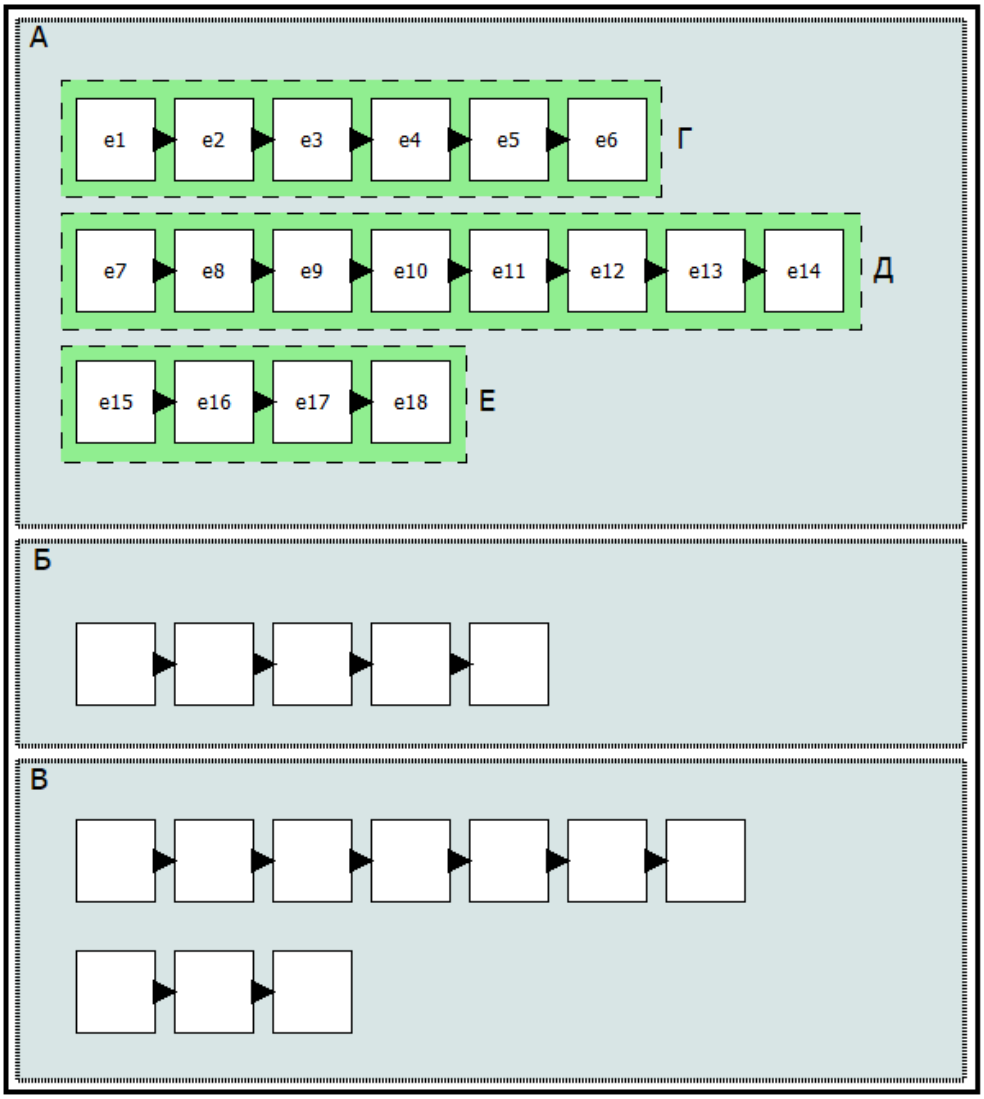

Рис. 2. Схема организации канонической модели

Журнал событий L схематически представляется в виде обрамляющего прямоугольника. В данном примере множество P состоит из процессов А, Б и В. Рассмотрим подробнее фрагмент лога, относящийся к процессу А. Как видно из схемы, зафиксировано три различных случая выполнения процесса – Г, Д и Е. В частности, $\sigma_{Д} = \{e_i\}_{i=7}^{14}$. Каждое из представленных в модели канонических событий однозначно идентифицируется некоторым дескриптором кейса и порядковым номером внутри последовательности событий, соответствующей выбранному кейсу.

3.2. Сведение логов событий к канонической модели

Следующим этапом построения решения является реализация адаптеров преобразований трасс событий в различных форматах в каноническую модель, описанную ранее. Требования, предъявляемые к рассматриваемым форматам, заключаются в следующем:

- распространенность: использование журналов в общеупотребительных форматах в качестве входных данных программы позволит применить предлагаемый подход к более широкому классу реальных систем;
- интерпретируемость: возможность использовать журнал для применения алгоритмов анализа и обработки данных.

В ходе исследования был выделен ряд форматов, удовлетворяющих представленным требованиям.

Во-первых, был выбран формат XES (eXtensible Event Stream, расширяемый поток событий), который был разработан в качестве стандарта для хранения и передачи журналов событий рабочей группой IEEE по направлению Process Mining [9]. Структура лога в формате XES задана иерархически и подобна схеме канонической модели для трассы событий. Диаграмма, показывающая принципы организации лога в формате XES, представлена на рис. 3.

С теоретической точки зрения, задача, которая решается при переводе трасс в каноническую модель, заключается в том, чтобы правильным образом осуществить отображение имеющихся в системном логе элементов на выделенные в качестве обязательных атрибуты канонического события. «Правильность» трактуется исходя из соображений интерпретируемости модели, полученной при некотором выбранном отображении сведения f. Подавляющее большинство трасс событий ориентировано на отображение потока управления процесса или множества процессов системы, поэтому сведение трасс в рассматриваемых форматах будет осуществляться с учетом указанного свойства.

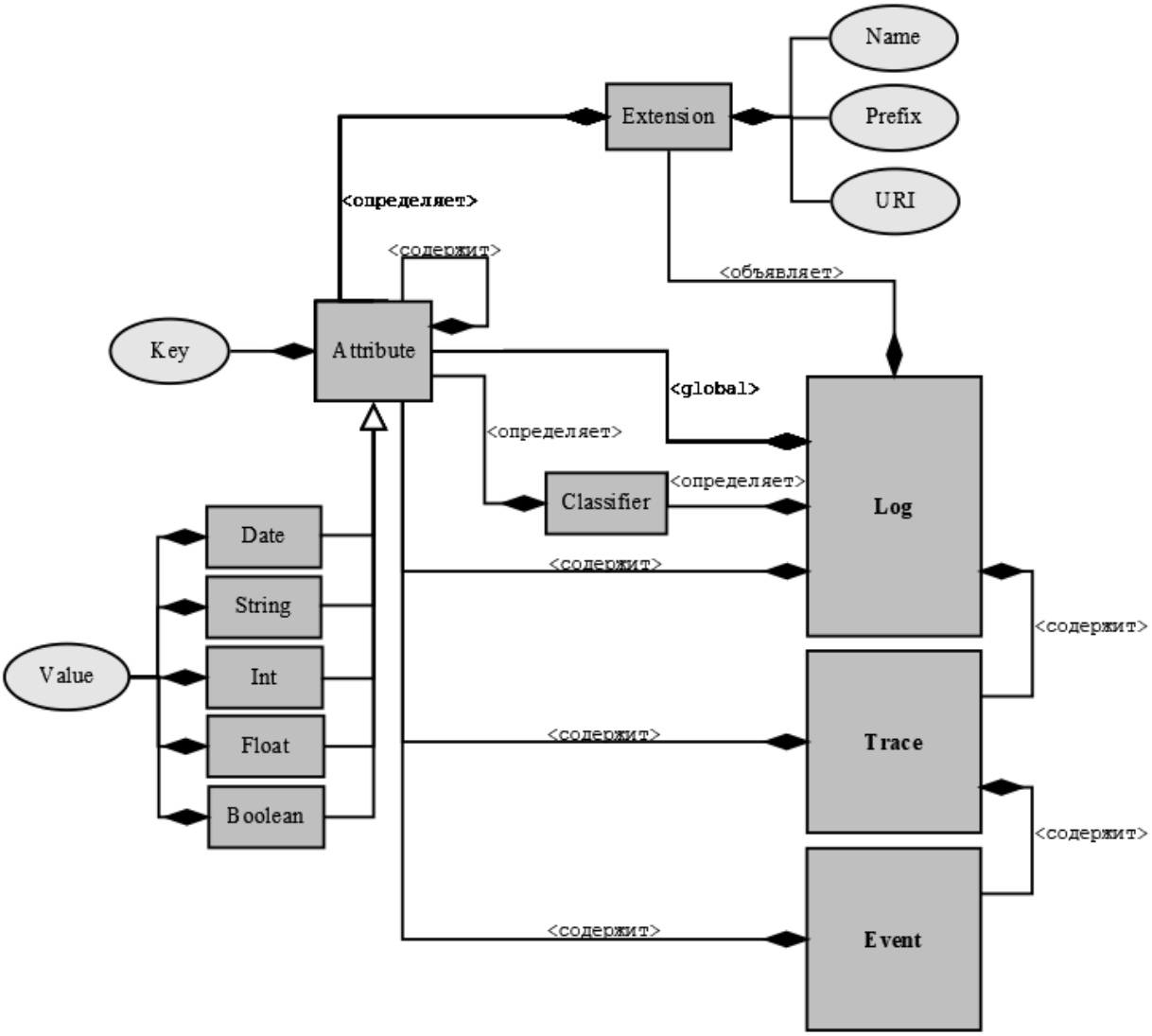

Рис. 3. Метамодель XES-формата

При непосредственном маппинге элементов файла журнала на атрибуты канонического события модели последнее будет целесообразно дополнить еще одним атрибутом *extra*, включающим в себя ту информацию из лога, которая не входит в набор необходимых атрибутов, однако потенциально полезна при генерации тестовых случаев. Таким образом, расширенным каноническим событием (РКС) будет являться некоторое событие e с множеством $A_1 = \{eventid, caseid, activity, timestamp, extra\}$.

Представим отображение для лога в формате XES (табл. 1), заметив прежде, что в данном формате не предусмотрена метка, предназначенная для идентификации процесса системы. То есть, неявно подразумевается, что лог описывает зафиксированное поведение некоторого единственного процесса. Обозначения в столбце «Элементы файла трассы» следующие: «*a*::*b*» означает, что *b* в документе является непосредственным дочерним узлом элемента *a*; «[?]» указывает местонахождение искомой информации.

Атрибуты РКС	**Элементы файла трассы**
eventid	(номер п/п <log>::<trace>::<event>)
caseid	<log>::<trace>::<string key="concept:name" value="[?]">
activity	<log>::<trace>::<event>::<string key="concept:name" value="[?]">
timestamp	<log>::<trace>::<event>::<date key="time:timestamp" value="[?]">
extra	<log>::<trace>::<event>::(остальные поля)

Табл. 1. Отображение элементов XES-лога на атрибуты расширенного канонического события

Во-вторых, был рассмотрен формат MXML (Mining eXtansible Markup Language), являющийся предшественником уже рассмотренного XES-стандарта и описанный в [10]. Данный формат был разработан для использования в качестве средства хранения и обмена логов событий для последующего применения интеллектуального анализа процессов. MXML является расширением формата XML, обладая при этом, по сравнению с XES, ограничениями на вид хранимой информации: заранее предопределен набор атрибутов, которыми могут обладать события в трассе. Пример фрагмента журнала событий в формате MXML приведен на рис. 4.

В отличие от формата XES, MXML позволяет содержать информацию относительно различных процессов приложения – блок данных, относящихся к определенному процессу, организован внутри тега <*Process*>.

```
<Process id="Client service 1">
    <ProcessInstance id="5817201">
        <AuditTrailEntry>
            <WorkflowModelElement>request service 1</WorkflowModelElement>
            <EventType>start</EventType>
            <Timestamp>2018-10-27T12:31:19.526+03:00</Timestamp>
            <Originator>user6168</Originator>
        </AuditTrailEntry>
        <AuditTrailEntry>
            <WorkflowModelElement>
                prepare receive buffer
            </WorkflowModelElement>
            <EventType>intermediate</EventType>
            <Timestamp>2018-10-27T13:23:14.281+03:00</Timestamp>
            <Originator>staff5</Originator>
        </AuditTrailEntry>
        <AuditTrailEntry>
            <WorkflowModelElement>perform task A</WorkflowModelElement>
            <EventType>intermediate</EventType>
            <Timestamp>2018-10-27T13:23:14.591+03:00</Timestamp>
            <Originator>staff5</Originator>
        </AuditTrailEntry>
        ...
```

Рис. 4. Фрагмент трассы событий в формате MXML

Отображение полей документа на атрибуты РКС не представляет особого труда, поскольку набор элементов MXML заранее предопределен. Маппинг представлен в табл. 2.

Атрибуты РКС	**Элементы файла трассы**
eventid	(номер п/п <WorkflowLog>::<Process>::<ProcessInstance>::<AuditTrailEntry>)
caseid	<WorkflowLog>::<Process>::<ProcessInstance id="[?]">
activity	<WorkflowLog>::<Process>::<ProcessInstance>::<AuditTrailEntry>::<WorkflowModelElement>
timestamp	<WorkflowLog>::<Process>::<ProcessInstance>::<AuditTrailEntry>::<Timestamp>
extra	<WorkflowLog>::<Process>::<ProcessInstance>::<AuditTrailEntry>::(остальные поля)

Табл. 2. Отображение элементов MXML-лога на атрибуты расширенного канонического события

Наконец, в рассмотрение включался формат, который используется в log4j2 [11] – библиотеке автоматического журналирования для приложений, написанных на языках программирования Java и Scala. Такой выбор обусловлен тем, что значительное число распределенных приложений в силу необходимости поддержки масштабируемости разработано с использованием API данных языков, из чего следует распространенность логов, порожденных с помощью log4j2. Формат сообщений при журналировании с использованием указанного фреймворка не является строго заданным. Одной из ключевых особенностей log4j2 является возможность достаточно специфической конфигурации вида записей в логе. Однако, в отличие от двух уже описанных форматов, данные в трассе образуют не древовидную конструкцию, а линейную – последовательность строк, соответствующих зафиксированным событиям.

Средства библиотеки позволяют с помощью задания шаблона сообщения настроить общий формат записи в логе. Для этого применяется набор спецификаторов, среди которых – спецификатор идентификатора процесса, временной метки, наименования класса, тела сообщения и т. д. Из множества спецификаторов было выделено подмножество, элементы которого были отображены на атрибуты расширенного канонического события (табл. 3).

Атрибуты РКС	**Спецификаторы шаблона log4j2**
eventid	%sequenceNumber
caseid	%processId
activity	(%class, %method)
timestamp	%date
extra	остальные спецификаторы

Табл. 3. Отображение спецификаторов шаблона log4j2 на атрибуты расширенного канонического события

3.3. Фильтрация данных

После сведения трассы L в модельное представление M требуется произвести очистку данных от возможных шумов. Под шумом понимается некоторое искажение в журнале событий, наличие которого обусловливает некорректное представление реального поведения системы. Возникновение шумов вызвано особенностями формирования журнала, а также способом выгрузки информации из этого журнала.

Системный лог представляет собой продукт функционирования приложения. В зависимости от организации процесса журналирования события в лог могут вноситься как программным образом, так и вручную (например, персонал клиники может указывать действия, производимые с пациентом). Компоненты системы могут обладать различными программными средствами логирования, которые, к тому же, могут иметь разные версии. Помимо этого, следует иметь в виду, что журнал хранится на физическом устройстве, что не может гарантировать абсолютной сохранности исходной информации. Наконец, лог, которым мы располагаем, может быть лишь некоторым фрагментом исходного общего лога (рис. 5), что также приводит к потенциальным шумам в полученной модели, которая наследует искажения трассы-прообраза.

Согласно [12, с. 4-5], под шумом могут пониматься следующие ситуации:

- появление лишнего перехода (события) в последовательности;
- пропуск перехода в последовательности.

К тому же, частичное отображение лога в модели порождает неполноценные кейсы процессов. К ним относятся префиксные и постфиксные кейсы.

Формально, представленные ситуации могут быть выражены следующим образом. Пусть $c \in M$ – некоторый кейс из канонической модели, c' – его искаженная версия. В таком случае несоответствие будет проявляться в различиях между σ_c и $\sigma_{c\prime}$. Разумеется, каждое событие, представленное в модели M, является уникальным, поэтому фактически любая

последовательность событий σ_c любого случая c процесса будет принципиально отличной от других последовательностей событий, входящих в модель.

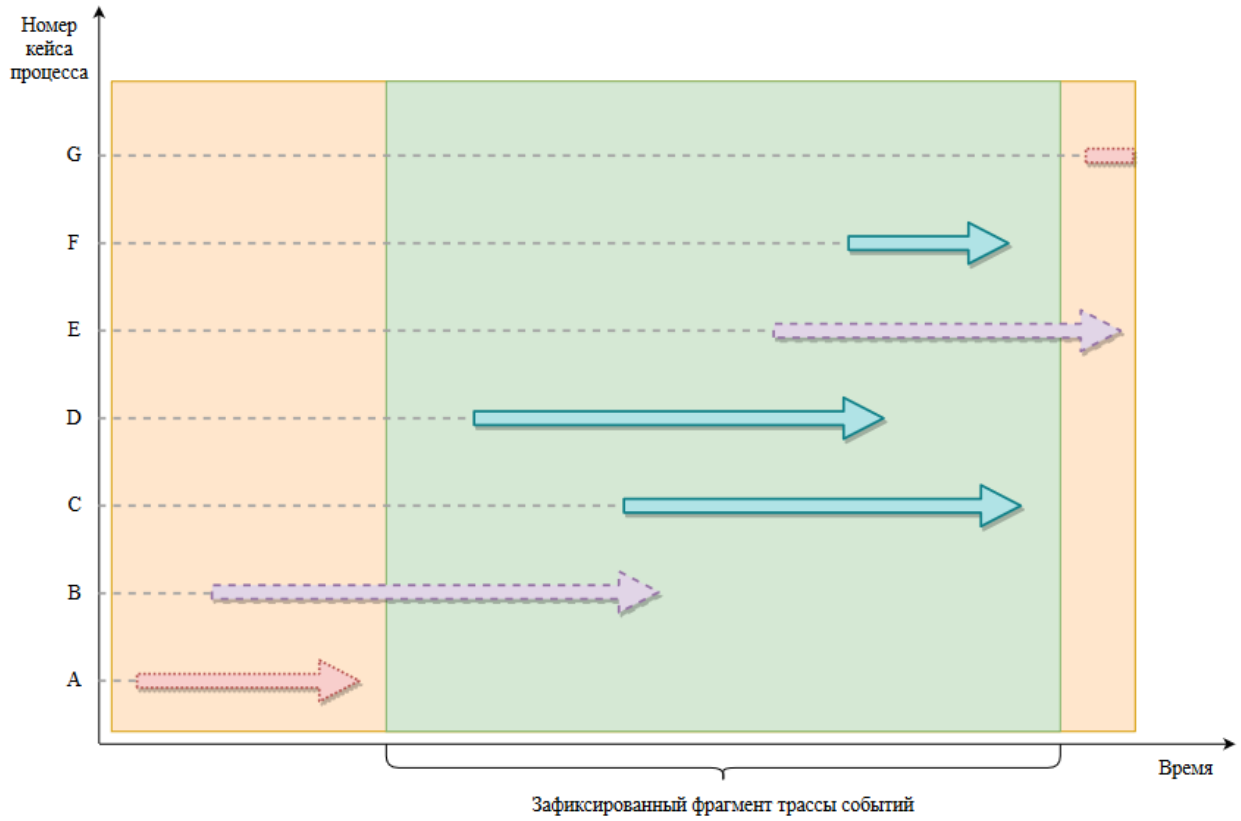

Рис. 5. Ситуация, при которой в доступный фрагмент трассы включаются неполноценные кейсы процессов или же зафиксированы не все возможные кейсы. Случаи A и G демонстрируют полностью не включенные во фрагмент кейсы. Случаи B и E соответствуют частично зафиксированным кейсам (B – постфиксный, E – префиксный). Случаи C, D, F корректно отображают действительность в извлеченном фрагменте лога.

Тем не менее, необходимо ввести возможность сравнения событий по некоторым атрибутам или их совокупностям для осуществления обработки имеющихся данных. Так, если производится анализ потока управления, то события e_1, e_2 в модели сравниваются по $\xi_{activity}(e)$. Далее в текущем разделе будет подразумеваться сравнение подобного вида, то есть

$$\xi_{activity}(e_1) = \xi_{activity}(e_2) \Leftrightarrow e_1 = e_2.$$

В таком случае событие e' в последовательности σ_c будет излишним по сравнению с σ_c, если имеют место следующие соотношения:

$$\sigma_c = (e_1, e_2, \dots, e_{i-1}, e_i, e_{i+1}, \dots, e_n);$$

$$\sigma_{c'} = (e_1, e_2, \dots, e_{i-1}, e_i, e', e_{i+1}, \dots, e_n).$$

Ситуация отсутствия события формализуется аналогично с очевидными изменениями.

Для отслеживания рассмотренных ситуаций и устранения шумов в модели предлагается использовать временные метки событий e_i и e_{i+1}, а также локальную относительную частоту (lrf – local relative frequency) последовательности $\sigma_{c'}$ относительно σ_c. Обозначим через $C[\sigma = \sigma_c]$ множество кейсов модели, таких что соответствующая им последовательность σ удовлетворяет следующим условиям:

$$|\sigma| = |\sigma_c|;\ \forall i = 1, 2, \dots, |\sigma|\ \ \sigma(i) = \sigma_c(i).$$

Тогда под $lrf(\sigma_{c'}, \sigma_c)$ будем понимать приведенную ниже величину:

$$lrf(\sigma_{c'}, \sigma_c) = \frac{|C[\sigma = \sigma_{c'}]|}{|C[\sigma = \sigma_{c'}]| + |C[\sigma = \sigma_c]|}.$$

Далее, для использования функции lrf следует задать частотный порог, при превышении которого последовательность $\sigma_{c'}$ не будет отнесена к шуму. Данное требование диктуется очевидным соображением: если выгрузка трассы событий содержит информацию о работе системы в течение длительного времени, то имеет смысл пометить как шум более редкие последовательности. Так, если значение lrf оказывается меньше порогового значения, $\sigma_{c'}$ понимается как шум и удаляется из модели.

Например, $\sigma_1 = (a, b, c, d, e)$ была зафиксирована 100 раз, а $\sigma_2 = (a, c, d, e)$ – всего 1 раз. Учитывая, что σ_2 может быть рассмотрена как искаженная относительно события b первая последовательность, σ_2 с высокой долей вероятности является шумом.

Приведенный принцип фильтрации на основе частотных показателей может быть дополнен проверкой с применением временных меток. Пусть

$$\sigma_c = \left(e_1^{(1)}, e_2^{(1)}, \dots, e_{i-1}^{(1)}, e_i^{(1)}, e_{i+1}^{(1)}, \dots, e_n^{(1)}\right);$$

$$\sigma_{c'} = \left(e_1^{(2)}, e_2^{(2)}, \dots, e_{i-1}^{(2)}, e_{i+1}^{(2)}, \dots, e_n^{(2)}\right);$$

$$\forall j = 1, 2, \dots, i-1, i+1, \dots, n \quad e_j^{(1)} = e_j^{(2)}.$$

Временные метки событий в модели могут быть представлены как целые числа, равные количеству миллисекунд с 00:00 1 января 1970 года до момента фиксации события. Если заранее выбрать некоторое допустимое отклонение δ и рассмотреть

$$\Delta t^{(1)} = \xi_{timestamp}\left(e_{i+1}^{(1)}\right) - \xi_{timestamp}\left(e_{i-1}^{(1)}\right),$$

$$\Delta t^{(2)} = \xi_{timestamp}\left(e_{i+1}^{(2)}\right) - \xi_{timestamp}\left(e_{i-1}^{(2)}\right),$$

то в случае $\left|\Delta t^{(1)} - \Delta t^{(2)}\right| < \delta$ при достаточно малом δ можно утверждать, что в последовательности $\sigma_{c'}$ было пропущено событие в i-ой позиции. Такой вывод обосновывается тем, что в обоих случаях время, затраченное на выполнение действий, ассоциированных с $e_{i-1}^{(1)}, e_i^{(1)}, e_{i+1}^{(1)}$, приблизительно одинаковое, однако последовательность событий различна. Это говорит о том, что в $\sigma_{c'}$ не было учтено $e_i^{(2)}$.

Кейс c' обозначим как префиксный, если

$$\exists c \in M: \forall i = 1,2, \dots, k;\ k < |\sigma_c| \quad \sigma_{c'}(i) = \sigma_c(i),$$

то есть начало последовательности σ_c до позиции k совпадает с $\sigma_{c'}(i)$. Для фильтрации подобных кейсов отдельно задается частотный порог, а затем вычисляется $lrf(\sigma_{c'}, \sigma_c)$: подобная процедура проделывается для всех пар кейсов в модели, в результате чего обнаруживаются потенциально неполноценные случаи. Постфиксные кейсы детектируются и отбрасываются аналогичным образом.

3.4. Обобщение модели

В качестве завершающей стадии обработки представленных в канонической модели данных выступает сокращение числа переходов в последовательностях σ_c. В [13 с. 6-7; 14 с. 116-117] рассматривается параметр обобщенности модели M, то есть возможности извлечь из модели информацию, непосредственно не представленную в логе событий. Процесс обобщения производится с целью избавления от нежелательного свойства модели быть излишне «подогнанной» под конкретный зафиксированный в журнале срез функционирования системы. В терминах машинного обучения это бы означало явление переобучения; в литературе по process mining данный феномен носит название overfitting.

В рамках данной работы предлагается применить алгоритм свертки повторяющихся фрагментов в циклы. Такой подход позволит значительно сократить число имеющихся в модели событий, заметно упростив структуру последней.

Алгоритм основывается на идее выделения шаблонов (паттернов) и поиска аналогичных им фрагментов в рассматриваемой последовательности событий. В качестве входных данных предоставляется список из событий, составляющих случай процесса. Помимо этого, используется упорядоченный набор ссылок на другие элементы последовательности, представляющий собой список множеств целых чисел – индексов событий в последовательности. Псевдокод алгоритма приведен ниже. Подразумевается, что элементы в коллекции нумеруются, начиная с нуля.

1. алг Свертка фрагментов
2. E := список событий σ_c
3. R := пустой список ссылок длины $|\sigma_c|$
4. i := 0
5. пока ($i < |E|$) выполнять
   5.1. $j := i + 1$

5.2. пока ($j < |E|$) выполнять

5.2.1. если ($E[i] == E[j]$) тогда

5.2.1.1. $p := E[i{:}j]$ //фиксация шаблона (часть списка от i до j)

5.2.1.2. $n :=$ *countPatternMatches*(E, p, i)

5.2.1.3. если ($n > 0$) тогда

5.2.1.3.1. $R[j\text{-}1] \cup= i$ //добавление ссылки на начало шаблона

5.2.1.3.2. *adjustReferences*(R, $|p|$, i, n)

5.2.1.3.3. $k := i + |p|*n$

5.2.1.3.4. *remove*(E, j, k)

5.2.1.3.5. *remove*(R, j, k)

5.2.1.3.6. $i := 0$

5.2.1.3.7. перейти на 5.

5.2.1.4. конец если

5.2.2. конец если

5.2.3. $j := j + 1$

5.3. конец пока

5.4. $i := i + 1$

6. конец пока

7. конец алг

В представленном описании алгоритма в целях удобства восприятия содержится вызов ряда процедур. Первая из задействованных – *countPatternMatches*(E, p, i) – осуществляет подсчет количества подряд идущих фрагментов, идентичных шаблону p, в списке E, начиная с индекса i. Это позволяет получить число элементов последовательности, требующих удаления. Далее, функция *adjustReferences*(R, $|p|$, i, n) производит отображение обнаруженных в $R[i+|p|]$, $R[i+|p|+1]$, … , $R[i+|p|*n - 1]$ ссылок на индексы относительно первого паттерна. Например, в ситуации, представленной на рис. 6, при повторном применении свертки (второй ярус

изображения) ссылка с элемента b на элемент d (было: индекс 4 ссылается на индекс 3) будет перенесена в диапазон индексов первого включения паттерна (b, c, d) (стало: индекс 1 ссылается на индекс 3).

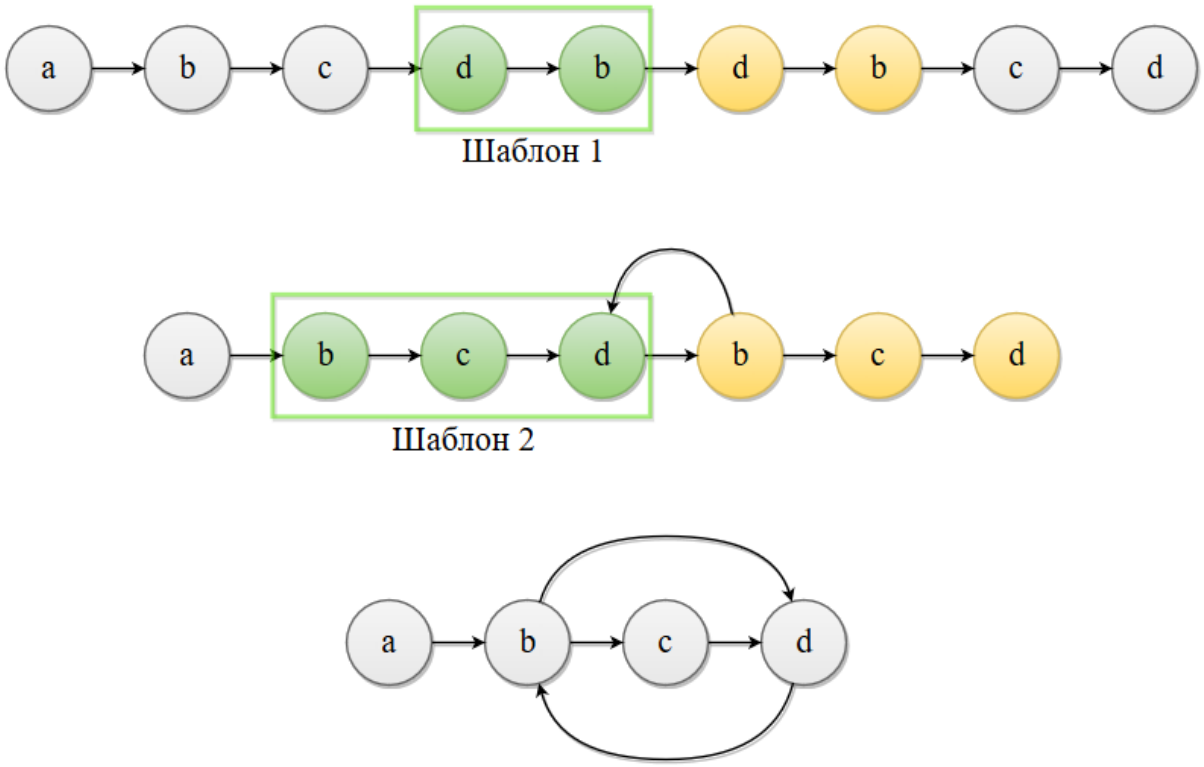

Рис. 6. Пример свертки повторяющихся фрагментов в циклы

Наконец, процедура *remove*(S, j, k) представляет собой подпрограмму удаления части последовательности S с позиции j включительно до индекса k, не включая k.

Приведенный алгоритм выполняется для каждого кейса $c \in M$. Таким образом выполняется обобщение имеющейся модели системы. Стоит заметить, что в результате применения свертки повторяющихся фрагментов со всяким случаем $c \in M$ ассоциируется последовательность множеств ссылок ref_c, описывающая дополнительные переходы между событиями $e \in c$.

3.5. Визуализация и генерация тестовых кейсов

Для непосредственного представления модели была выбрана репрезентация в виде графа $G = (V, E)$. В [15 с. 17-18] описана система помеченных переходов (labeled transition system, LTS) как ориентированный граф, где в качестве множества вершин V выступает множество состояний исследуемого процесса системы, а в качестве множества ребер E – набор дуг, соединяющих состояния. Под состоянием процесса в данном контексте следует понимать стадию, в которой находится процесс в фиксированный момент. Каждая $e \in E$ обладает меткой, которая задается как $\xi_{activity}(a)$, где a – событие, ассоциированное с переходом из состояния v_1 в v_2.

Заметим, что репрезентация состояния процесса может быть задана различным образом, в зависимости от предъявляемых требований к конечному представлению модели. В пределах текущей задачи целью является формирование тестовой модели, поэтому требуется наиболее точно отобразить данные, содержащиеся в канонической модели системы. Это обусловливает задание текущего состояния процесса как последовательности действий в рамках событий, зафиксированных до момента, соответствующего рассматриваемому состоянию.

Процесс построения LTS при выбранной репрезентации состояния является итеративным: $\forall c \in M$ основой для формирования отдельного пути в графе G служит σ_c. Последовательность σ_c образует следующий набор состояний:

$$S = \{(\sigma_c(1)), (\sigma_c(1), \sigma_c(2)), \dots, (\sigma_c(1), \sigma_c(2), \dots, \sigma_c(|\sigma_c|))\}.$$

Для всех $i = 1, 2, \dots, |S|$ состояние из S с индексом i соединяется с состоянием $(i-1)$ дугой, помеченной $\xi_{activity}(\sigma_c(i))$. В качестве изначального состояния в G задается некоторая стартовая вершина v_0. Заметим, что в данном случае граф будет представлять собой корневое дерево с корнем v_0.

На рис. 7 проиллюстрирована LTS, построенная на основе имеющихся в упрощенной модели M трех случаев процесса c_1, c_2 и c_3. События в

упомянутых кейсах отождествлены с атрибутами $activity$ событий. Как видно, каждому кейсу соответствует отдельный путь в графе; в качестве начального состояния выбрано v_0 – «пустое» состояние, поскольку согласно описанному заданию стадия выполнения процесса характеризуется последовательностью имевших место действий.

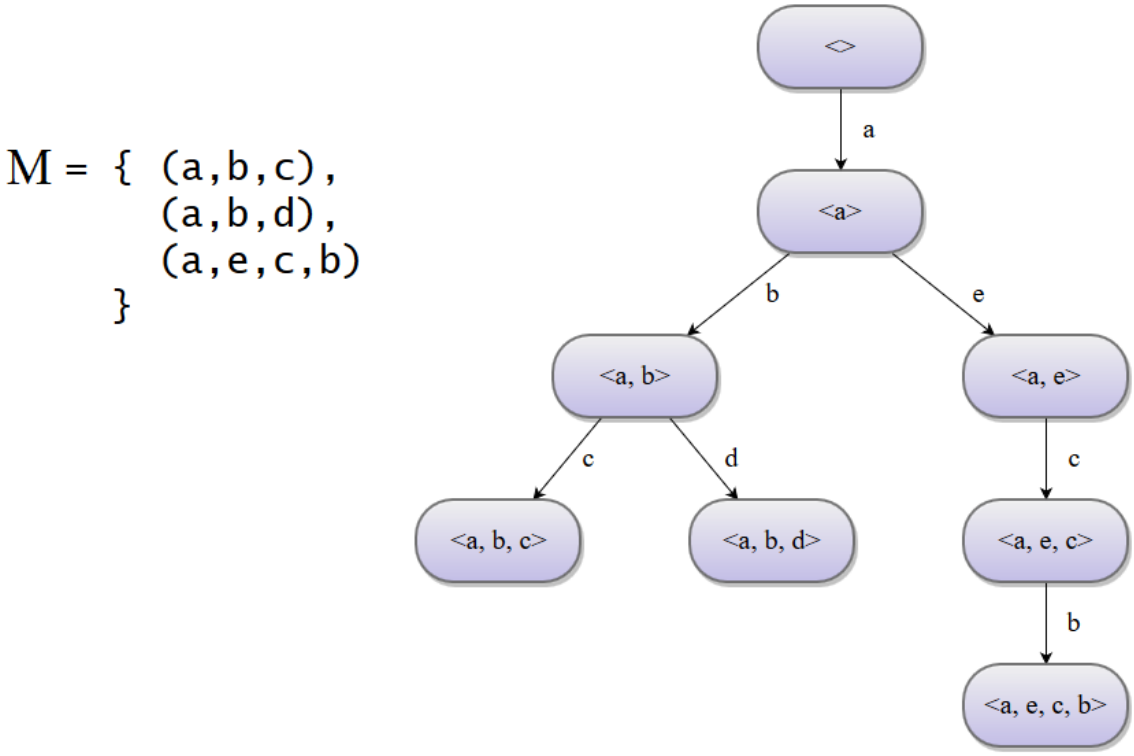

Рис. 7. Пример построения LTS на основе имеющихся в модели данных

Помимо LTS, для канонической модели в обобщенном виде предусматривается перевод в диаграмму действий (activity map, AM). Как и в предыдущем случае, AM – это ориентированный граф, однако в вершинах представлены $\xi_{activity}(e)\ \forall e \in M$, а в качестве дуг выбираются переходы между событиями, полученные в результате работы алгоритма свертки повторяющихся фрагментов (подраздел 3.4 текущего раздела).

Завершающим (с теоретической стороны) этапом построения решения в рамках настоящей работы выступает описание механизма автоматического перевода имеющейся модели системы в абстрактные тестовые случаи. В [16, с. 81-84] приведен набор основных техник для выбора тестовых ситуаций

и составления тестового набора, среди которых вероятностное тестирование, верификация с выделением классов эквивалентности, сценарное, автоматное тестирование и т. д. С учетом специфики представления модели, наиболее пригодным способом из перечисленных является ряд техник автоматного тестирования: данные для тестов генерируются на основе путей на графе переходов модели, описывающей поведение системы. То есть, в случае LTS подобная процедура осуществляется путем обхода имеющегося графового представления модели в глубину с промежуточной фиксацией в памяти последовательностей, соответствующих каждому пути от корня v_0 до конечной вершины-листа. Каждая такая последовательность содержит данные, предназначенные для составления отдельного тестового случая.

Аналогичные действия требуются и в случае АМ. Однако в силу того, что в общем случае данный граф не представляет собой дерево, необходимо использовать альтернативу обходу в глубину. В качестве решения предлагается ввести отношение эквивалентности R на множестве всех кейсов $\mathrm{c} \in M$, задаваемое как равенство значений $activity$ у начальных событий в последовательностях σ_c. Иными словами,

$$\forall \mathrm{c}_1, \mathrm{c}_2 : |\sigma_{c_1}| > 0, |\sigma_{c_2}| > 0 \quad \mathrm{c}_1 R \mathrm{c}_2 \Leftrightarrow \xi_{activity}\left(\sigma_{c_1}(1)\right) = \xi_{activity}\left(\sigma_{c_2}(1)\right).$$

Таким образом, множество кейсов разбивается на k классов эквивалентности. В результате, построение одной громоздкой диаграммы действий сводится к конструкции k более простых в отношении структуры аналогов уже для последовательностей событий в рамках кейсов одного класса эквивалентности M_i. Что более существенно, в каждой АМ(M_i) можно выделить начальную вершину, являющуюся общей для всех путей в данной диаграмме. Исходя из определения LTS и АМ, всякий объект визуального представления модели имеет своим прообразом некоторый элемент канонической модели. В случае отдельного пути в графе АМ(M_i) таким прообразом является случай $\mathrm{c} \in M_i$, ассоциированный с ref_c. Любой кейс может быть представлен как упорядоченный набор T_c пар вида

$$(activity, \{i_1, \dots, i_n\}),$$

где $activity = \xi_{activity}(\sigma_c(i))$, $\{i_1, \dots, i_n\}$ – множество ссылок на элементы в последовательности σ_c, определяемое из $ref_c(i)$. Подобным образом заданные T_c для каждого кейса представляют собой, как и в случае LTS, данные для составления отдельного абстрактного тестового кейса.

3.6. Выводы по разделу

В данном разделе было представлено подробное описание стадий исследования и построения решения рассматриваемых в рамках настоящей работы задач. Было дано теоретическое обоснование принимаемым решениям по выделению канонической модели поведения исследуемой системы, выбору форматов представления логов событий, а также использованию методов процессинга имеющейся модели, включающих в себя принципы очистки от шумовых данных, обобщения и визуализации канонической модели. Наконец, был описан механизм генерации тестовых кейсов из полученной в результате анализа модели системы.

РАЗДЕЛ IV
ОПИСАНИЕ ПРАКТИЧЕСКОЙ ЧАСТИ

4.1. Обоснование выбранного инструментария

Программное средство, реализующее описанные в разделе «Исследование и построение решения» стадии извлечения и последующей обработки модели системы, было разработано с помощью Java Development Kit 10.0.1 [18]. Выбор языка программирования Java был обусловлен рядом его преимуществ.

В первую очередь, это следование принципам объектно-ориентированного программирования, что позволяет эффективно организовать исходный код программы, распределяя данные и методы для работы с этими данными по заранее спроектированным классам. Далее, исполнение Java-программ производится с помощью JVM – виртуальной машины, предоставляющей возможность запускать программы на платформе, отличной от той, на которой производилась компиляция исходного кода. Так, Java поддерживает принцип WORA (write once, run anywhere – написано однажды, работает везде), то есть является кроссплатформенным языком. Наконец, данный язык программирования обладает средством автоматического управления памятью. В комплексе со сборщиком мусора, данный инструмент минимизирует риск утечек оперативной памяти, что, к тому же, повышает эффективность разработки путем уменьшения временных затрат на организацию ручного управления памятью.

В качестве среды разработки была использована Eclipse IDE 4.9.0, предоставляющая широкий выбор инструментария для создания и отладки программ на языке Java. Для работы с файлами трасс событий в форматах XES и MXML была применена библиотека JAXB [19], разработанная для распаковки содержимого XML-документов в программное представление, получению доступа к XML-контенту, а также для инкапсуляции Java-объектов

в структурные элементы XML. Помимо этого, для эффективной организации графового представления модели была использована библиотека JGraphT [20]. Средства библиотеки включают в себя возможность инициализации и преобразования графов различных типов, что дополняется функционалом для экспорта содержащихся в графах данных в текстовые форматы, такие как DIMACS, CSV, DOT, GML и GraphML. Для непосредственного перевода графа в визуальное представление в формате PDF был задействован пакет утилит Graphviz [21], среди возможностей которого – трансляция файлов в формате DOT в документы форматов PostScript и PDF.

4.2. Структура и принцип работы программного решения

Исходный код разработанного приложения (см. Приложение Б) организован в виде двух Java-пакетов: extractor и model (рис. 8).

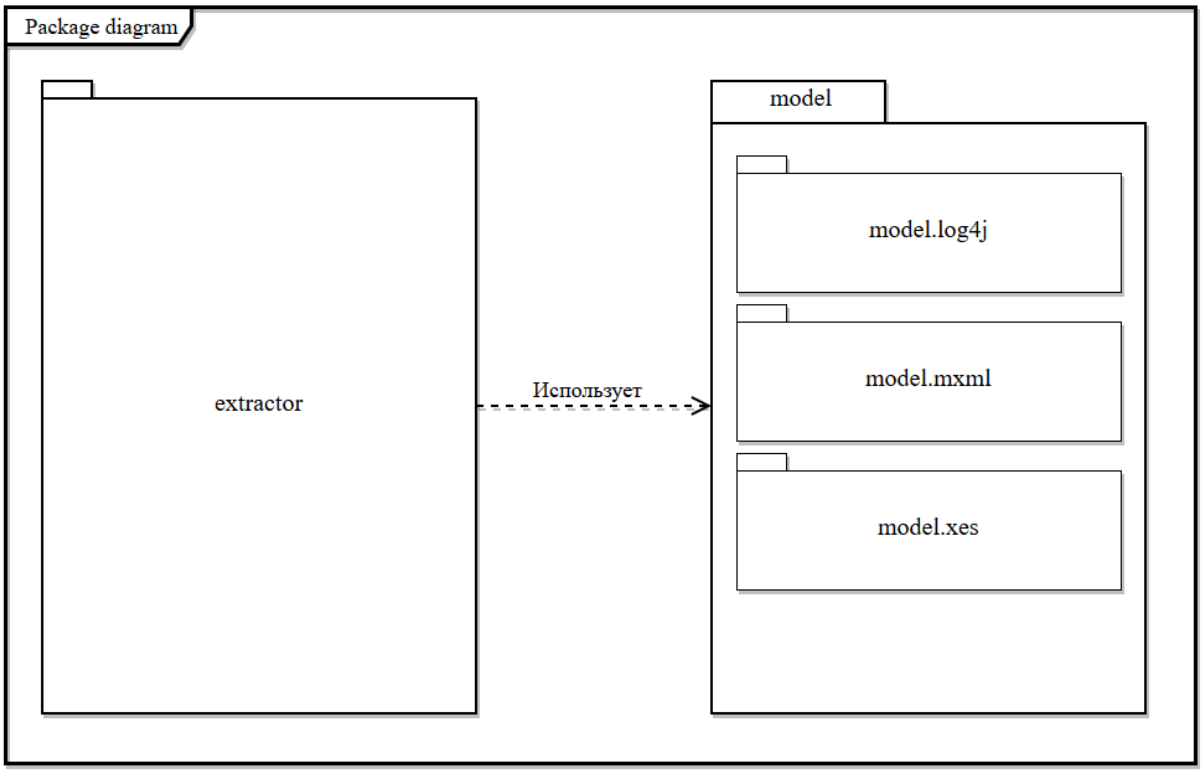

Рис. 8. Диаграмма пакетов приложения

Пакет model, помимо классов, описывающих отдельные составляющие модели системы, содержит также пакеты, соответствующие классам, предназначенным для работы с журналами в форматах log4j2, MXML и XES. Как можно видеть на диаграмме, пакет extractor использует интерфейсы, предоставляемые классами пакета model.

Каноническая модель описывается классом Canonical, где в качестве канонического события выступает класс Event, содержащий выделенные ранее атрибуты события в наборе своих полей. Отдельный кейс процесса представляется списком событий, однако в непосредственной технической реализации случай процесса моделируется экземпляром класса TaggedList. Класс Translator содержит механизмы перевода в каноническую модель данных, отображенных из трасс в изначальных форматах на объекты классов, включенных в пакеты model.log4j, model.mxml и model.xes. Класс ReferencedSequence используется при обобщении модели – данный класс применяется для представления последовательностей событий, дополненных списком, содержащим выделенные в ходе работы алгоритма свертки ссылки на другие элементы последовательности. Наконец, классы TransSystem и ActivityMap реализуют представление канонической модели в виде графовой структуры. Описанная организация пакета model представлена на диаграмме классов (рис. 9). Классы TransSystem и ActivityMap наследуют DefaultDirectedWeightedGraph и DefaultDirectedGraph (соответственно) из библиотеки JGraphT.

Далее, пакет extractor агрегирует классы, служащие для перевода модели из внутреннего представления в формат описания DOT, а также классы, обеспечивающие организацию взаимодействия с конечным пользователем (рис. 10). GraphManager реализует экспорт объектов классов TransSystem и ActivityMap во внешний текстовый формат. Класс InterfaceWindow задает графический интерфейс приложения, наследуя класс JFrame встроенной в JDK библиотеки разработки GUI Swing и определяя в качестве вложенного класса

LogStream. Наконец, Main представляет собой основной класс программы, исполняя роль точки входа в приложение.

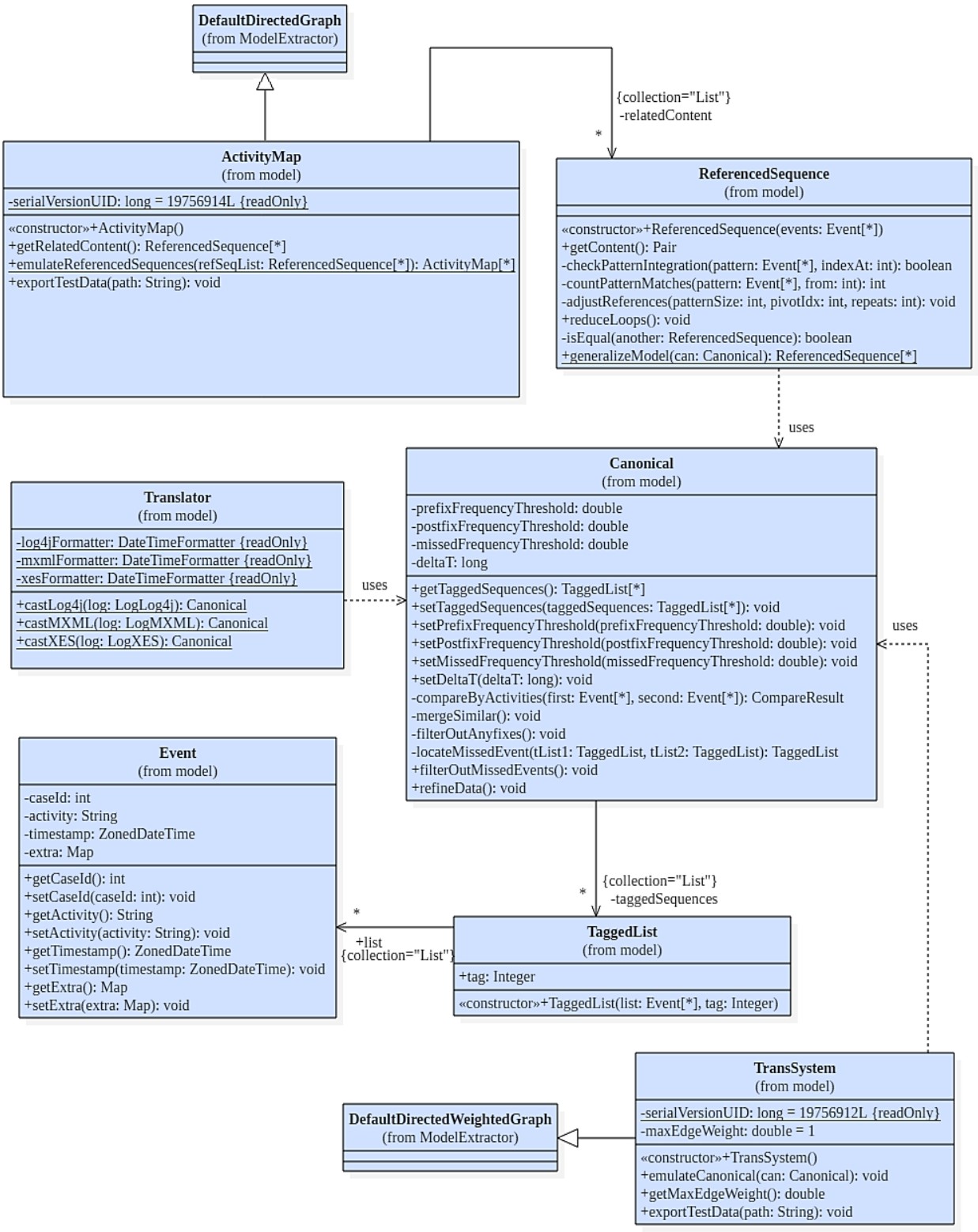

Рис. 9. UML-диаграмма классов пакета model

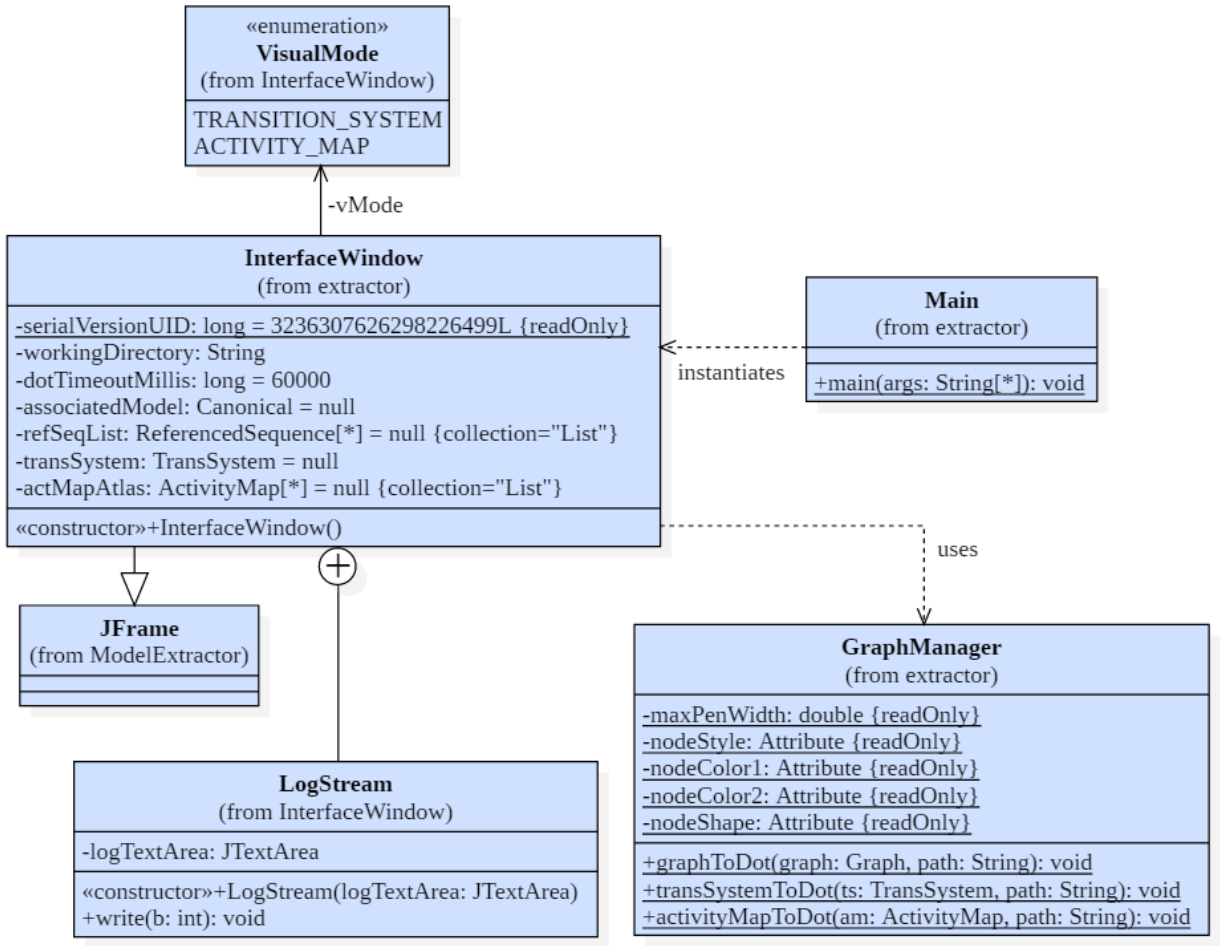

Рис. 10. UML-диаграмма классов пакета extractor

Перейдем к рассмотрению схемы работы реализованной программы. Разобьем представленный функционал на логические блоки, каждый из которых обособлен согласно специфике производимой обработки данных. Так, в разработанном приложении выделяются блоки: диспетчеризации, преобразования (адаптеры перевода в промежуточное представление), трансляции в каноническую модель, фильтрации, обобщения, визуализации и генерации тестовых кейсов. Порядок выполнения упомянутых элементов представлен на рис. 11. В ходе работы программного средства в первую очередь определяется формат входных данных – формат исходной трассы событий. На данной стадии применяется блок диспетчеризации, осуществляющий выбор соответствующего адаптера преобразования.

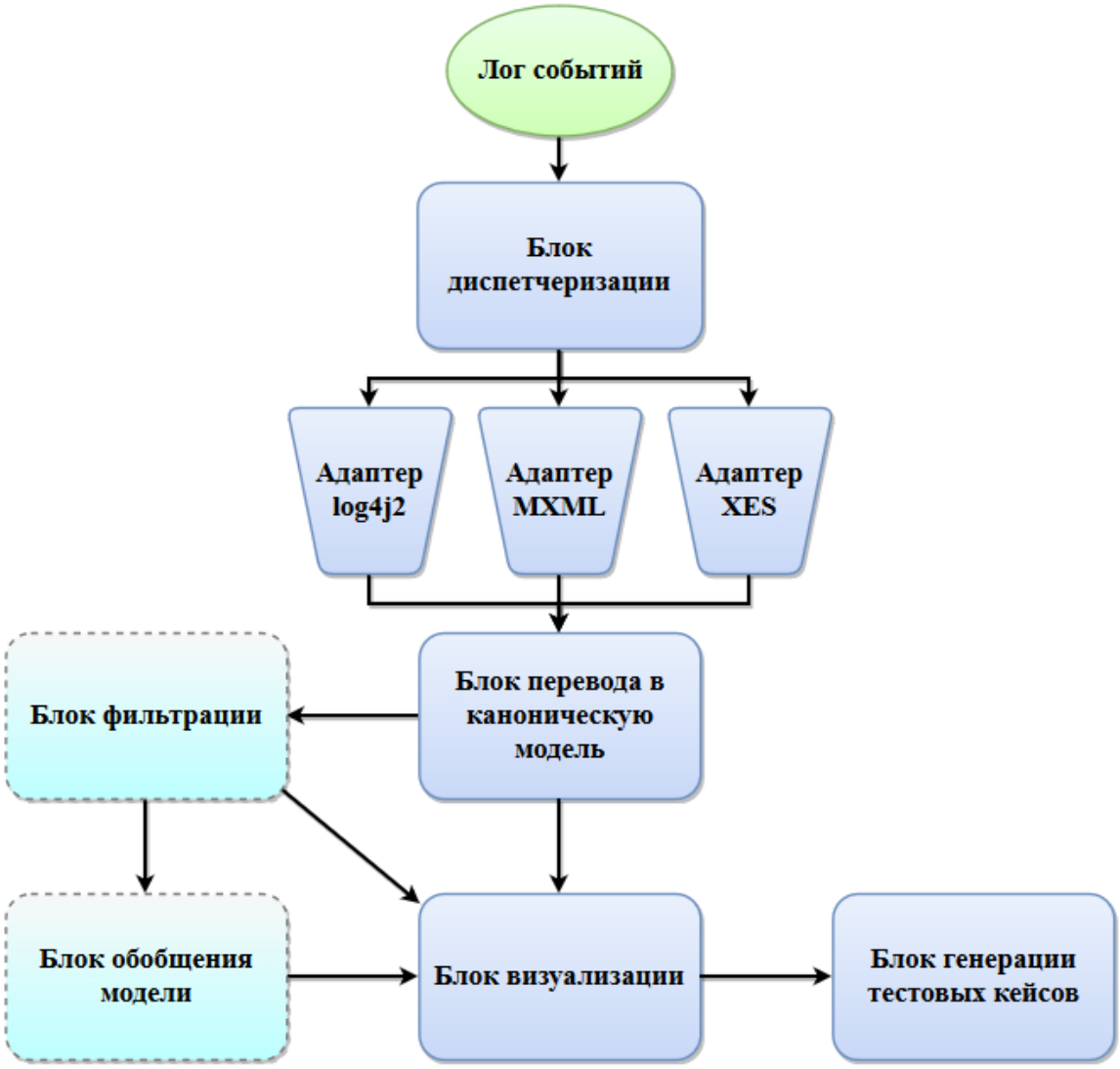

Рис. 11. Схема работы программного решения

Каждый из адаптеров содержит средства трансляции журнала событий в промежуточное представление, предназначенное для конкретного формата. После получения таковой репрезентации трассы блок перевода в каноническую модель осуществляет приведение данных к унифицированному формату. Дальнейший ход работы программы может варьироваться в зависимости от выбранной пользователем конфигурации выполнения обработки трассы. В связи с этим, блоки фильтрации и обобщения модели на схеме (рис. 11) выделены пунктирной линией: данные блоки не обязательно выполняются в ходе функционирования приложения. Тем не менее, любой из

них может быть применен к модели при условии сохранения представленного в схеме порядка следования. Блок визуализации предназначен для перевода канонической модели в одно из графовых представлений, выбор которого осуществляется в зависимости от того, было ли выполнено обобщение модели: если было применено, то используется диаграмма действий; в противном случае производится перевод в LTS. Конечным в последовательности выступает блок генерации тестовых случаев, выполняющий экспорт данных из модели в XML-файлы, содержащие информацию о зафиксированных сценариях поведения системы. Пример фрагмента одного из таких файлов приведен на рис. 12.

```
<Testcase>
  <Action>
    incoming claim
  </Action>
  <Dataset>
    call centre: Brisbane
    org:resource: customer
    lifecycle:transition: complete
  </Dataset>
  <Action>
    B check if sufficient information is available
  </Action>
  <Dataset>
    org:resource: Call Centre Agent Brisbane
    location: Brisbane
    lifecycle:transition: start
  </Dataset>
  <Action>
  ...
```

Рис. 12. Фрагмент сгенерированного абстрактного тестового кейса

4.3. Исходные данные

В качестве исходных данных были рассмотрены логи событий, собранные с реальных систем, то есть потенциально содержащие зашумленные данные. Экземпляры подобных журналов содержатся в открытом репозитории 4TU.Centre for Research Data [22]. Данная коллекция включает трассы событий, полученные из инструментированных средствами

фиксации информации о взаимодействиях между модулями систем. Упомянутый ресурс содержит трассы систем из сфер муниципального управления, медицинского обслуживания и т. д.

4.4. Результаты анализа трасс событий

В качестве первой для анализа была выбрана трасса, содержащая события вызова методов при однократном исполнении набора юнит-тестов для исследовательской машины экипажа NASA [23]. Обозначим данный лог литерой A. Исходный журнал включает в себя 73638 событий, зафиксированных с 13.02.2017 15:50:51,610 по 13.02.2017 15:50:55,840. Для отображения зависимости времени обработки данных от числа событий в трассе при идентичной структуре события было создано несколько усеченных версий лога A. Подобная процедура осуществлялась путем последовательного отбрасывания последних N строк в файле журнала с поправками на сохранение целостности записей. Число N выбиралось произвольным образом при выполнении условия $N \approx \frac{1}{6}|F|$, где $|F|$ – число строк в полноценном файле трассы.

Параметры программы при обработке трассы A были следующие: частотный порог для префиксных кейсов – 25 %, частотный порог для постфиксных кейсов – 15 %, порог для кейсов с пропущенным событием – 15 %, допустимая временная девиация δ – 600 *мс*. Результаты выполнения программы представлены в табл. 4. Измерения времени работы усреднялись по трем сеансам извлечения модели.

Кол-во событий	**Кейсов изначально**	**Кейсов после удаления идентичных**	**Кейсов после фильтрации**	**Кейсов после обобщения**	**Время работы (*мс*)**
36015	1242	1236	1236	1222	2339
48383	1682	1673	1673	1659	3494
60869	2088	2045	2044	1979	4713
73638	2566	2513	2512	2431	6947

Табл. 4. Результаты обработки трассы A

В таблице иллюстрируется порядок последовательного очищения модели от случаев процесса, не предназначенных для включения в конечное представление тестовой модели. Время работы, фигурирующее в правом столбце, включает в себя суммарную продолжительность выполнения следующих стадий работы приложения: открытие файла и построение канонической модели, очистка от шума (префиксные, постфиксные кейсы и случаи с пропуском события), обобщение модели путем применения алгоритма свертки, формирование графового представления, экспорт в формат DOT и, наконец, генерация набора тестовых кейсов. Фрагмент диаграммы действий, полученной в ходе анализа, представлен на рис. 13.

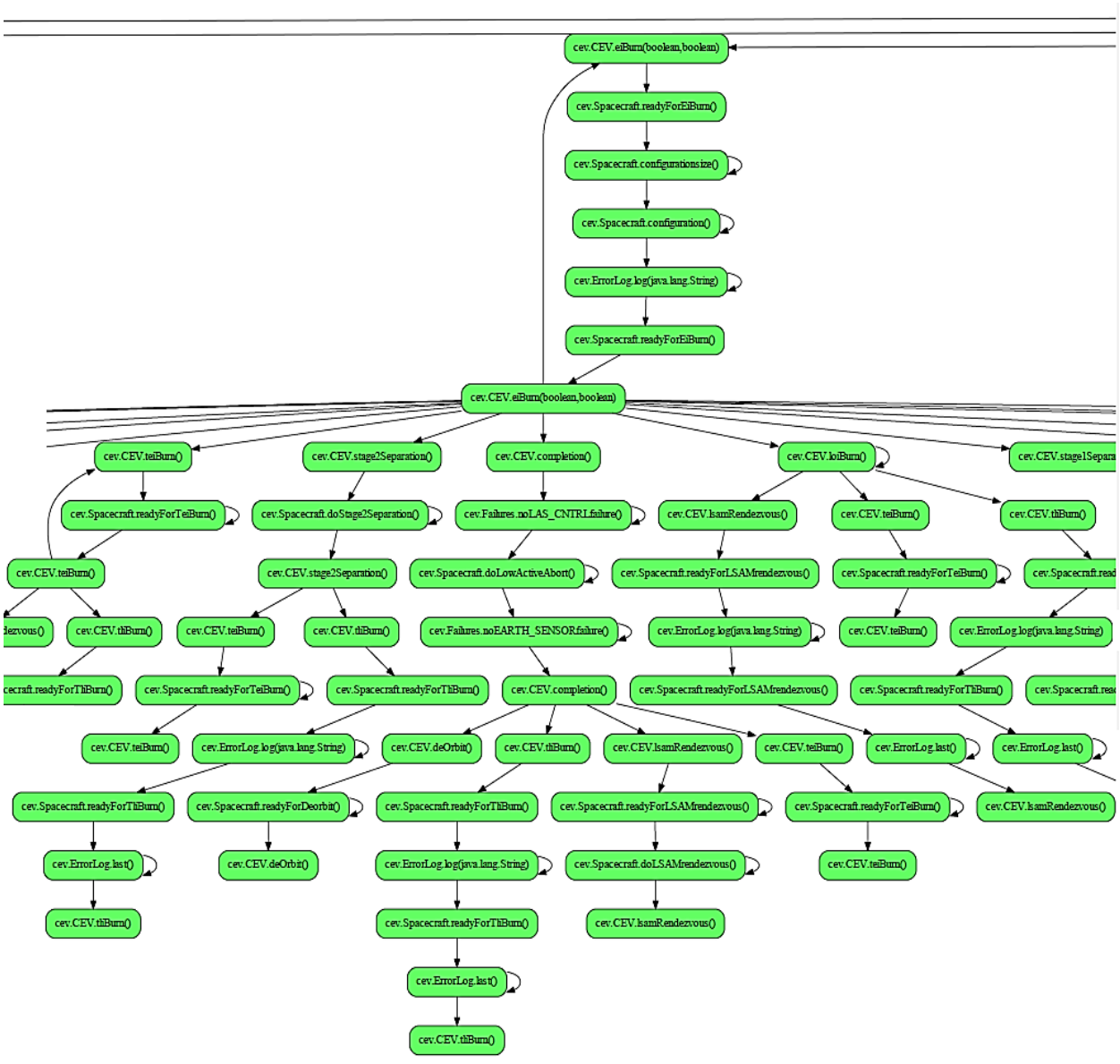

Рис. 13. Фрагмент activity map для трассы A

Следующей рассмотренной трассой является лог, представленный в [24] и полученный с системы обработки заявок на кредитование. Данный лог обозначим как лог B. Трасса содержит 262200 событий, зафиксированных в промежуток с 1.10.2011 00:38:44,546 по 14.03.2012 16:04:54,681.

При выполнении обработки журнала были выбраны следующие параметры фильтрации: частотный порог для префиксных кейсов – 15 %, частотный порог для постфиксных кейсов – 10 %, порог для кейсов с пропущенным событием – 15 %, допустимая временная девиация δ – 512 *мс*. Результаты, полученные при анализе, продемонстрированы в табл. 5.

Кол-во событий	**Кейсов изначально**	**Кейсов после удаления идентичных**	**Кейсов после фильтрации**	**Кейсов после обобщения**	**Время работы (*мс*)**
131527	6609	2326	2249	1155	4965
175066	8642	2979	2874	1479	7975
218600	10661	3640	3513	1774	11424
262200	13087	4366	4206	2036	16700

Табл. 5. Результаты обработки трассы B

Фрагмент сгенерированной диаграммы действий для журнала B представлен на рис. 14.

Как видно из сравнения таблиц 4 и 5, в случае лога B число кейсов, распознанных как зашумленные, в процентном содержании гораздо выше, чем в случае лога A. Это в первую очередь объясняется различиями в системах-источниках рассмотренных журналов событий. Стадии фильтрации и обобщения значительно сократили общее число кейсов в канонической модели, что привело к упрощению результирующего представления и, следовательно, упростило полученный в итоге набор тестовых кейсов.

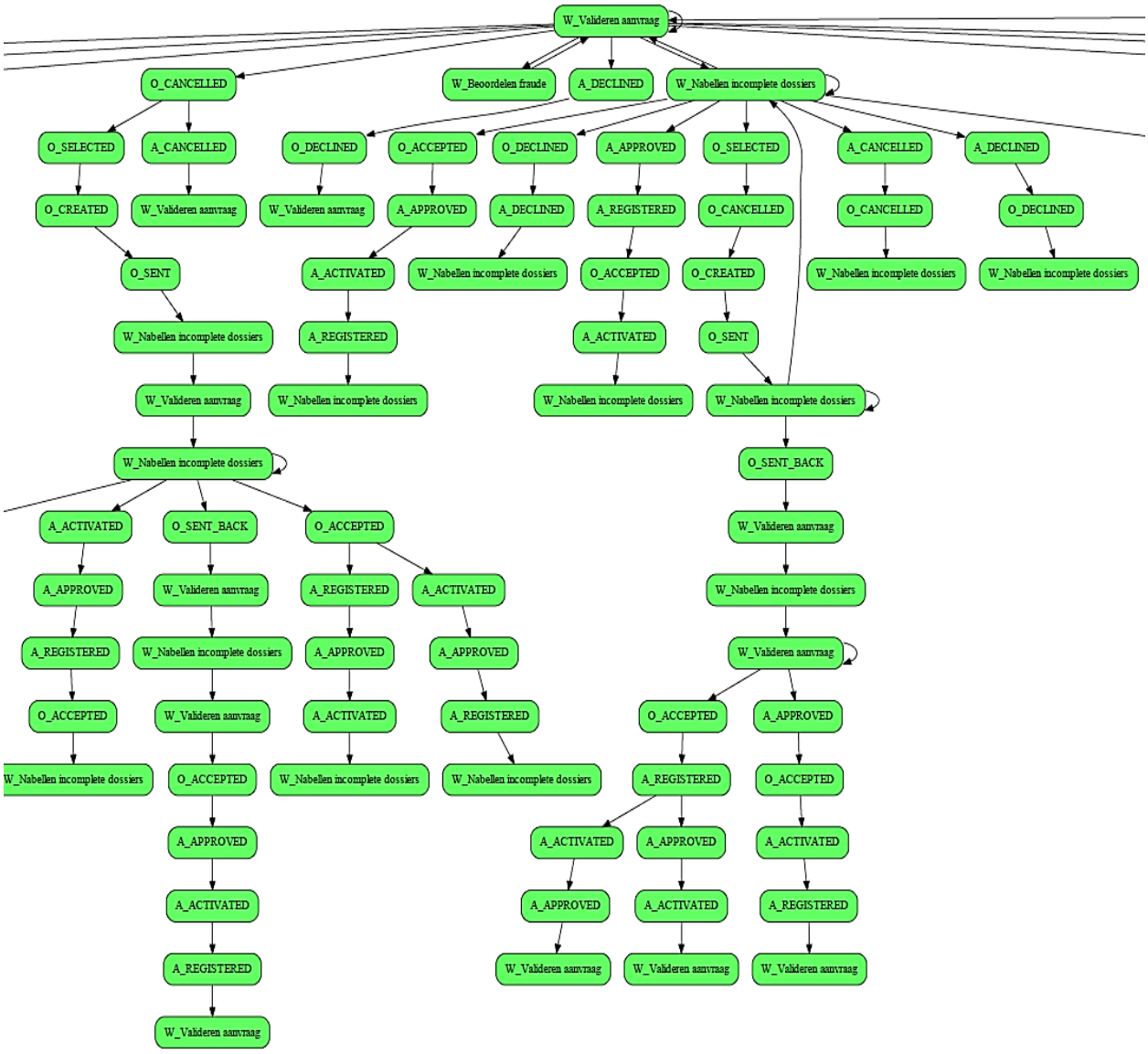

Рис. 14. Фрагмент activity map трассы B

Что касается процессорного времени, затраченного на каждую из стадий обработки данных, наиболее трудоемкую составляющую алгоритма программы имеет участок, связанный с фильтрацией трассы, поскольку включает в себя процедуры попарного просмотра всех имеющихся в модели кейсов. Трудоемкость стадий выполнения программы проиллюстрирована на рис. 15, где в процентном соотношении показаны временные затраты, связанные с отдельным этапом работы приложения. Диаграмма построена на основе зафиксированных в ходе обработки полной версии трассы A усредненных значений затраченного времени.

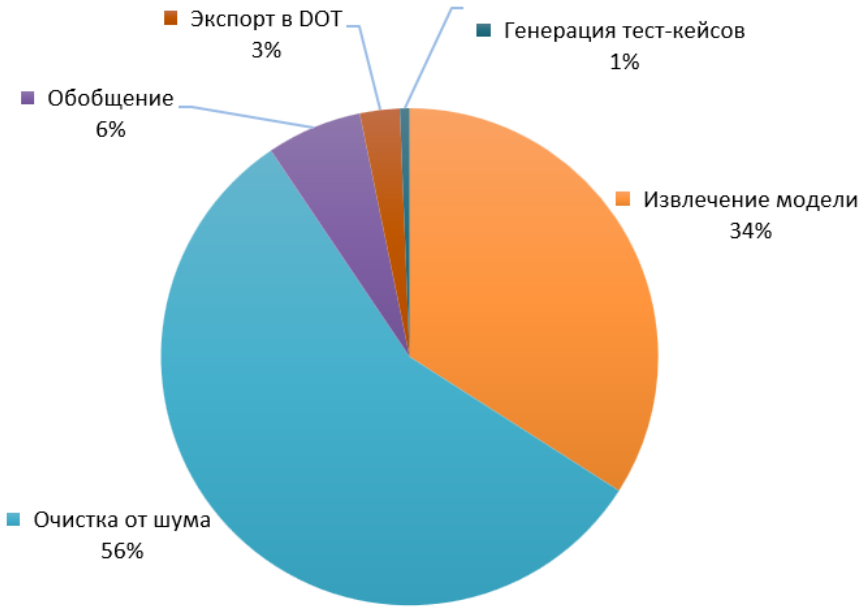

Рис. 15. Временные затраты на каждую стадию обработки данных (включены все стадии)

Стоит заметить, что в случае запуска программы без задействования этапа обобщения модели (см. рис. 16) соотношение долей затраченных временных ресурсов на идентичной трассе будет отличаться от приведенного в рис. 15.

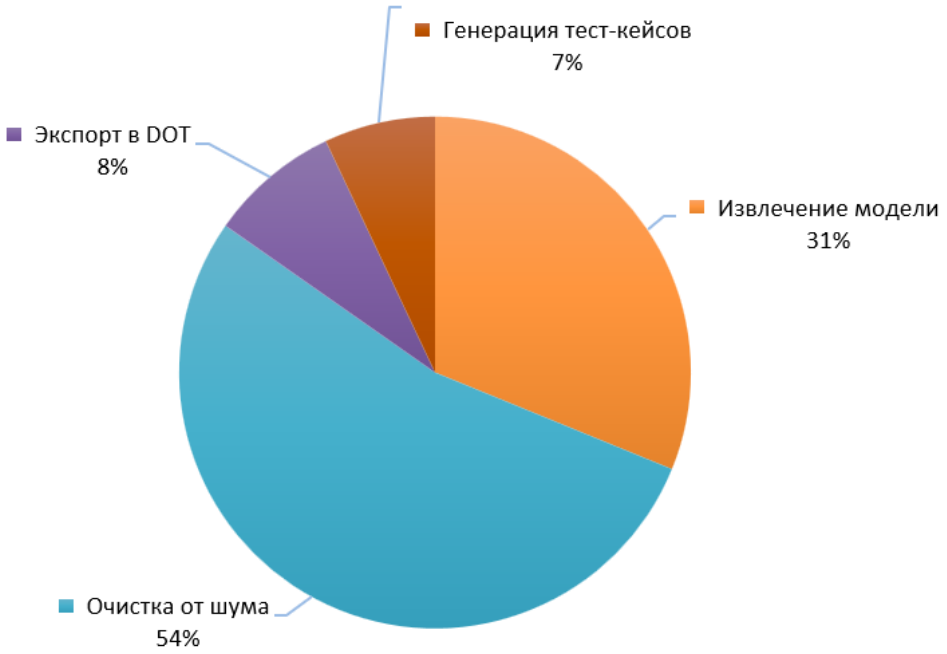

Рис. 16. Временные затраты на каждую стадию обработки данных (без стадии обобщения)

Расхождения проявляются в показателях для стадий экспорта модели в DOT и генерации тестовых кейсов, что обусловливается, во-первых, большим числом кейсов в модели при отсутствии обобщения, и, во-вторых, более подробной структурой записей в случае использования LTS в качестве объекта для порождения набора тестовых кейсов. Так, сгенерированный с задействованием обобщения модели (то есть на основе AM) тестовый случай содержит информацию исключительно о порядке выполнения действий в системе, в то время как созданный на основе LTS помимо упомянутого включает также дополнительные данные о состоянии системы (если таковые имелись в исходном журнале событий). В итоге, различия наблюдаются не только во времени работы отдельных участков программы, но и в размерах полученных файлов с тест-кейсами: 3730 *KB* (с обобщением) и 30838 *KB* (без обобщения) при размере исходного лога A в 73127 *KB*. Примеры фрагментов сгенерированных файлов с тестовыми случаями приведены в Приложении А.

4.5. Выводы по разделу

В данном разделе было дано обоснование выбранному для разработки программного средства инструментарию, подробно освещены аспекты реализации приложения, а также приведены результаты построения тестовых моделей на основе анализа журналов событий, полученных с реальных систем. Методы фильтрации и обобщения модели, предложенные в данной работе, показали свою состоятельность при использовании для очищения извлеченной из лога модели от несущественных кейсов при генерации набора тестовых случаев. Помимо вышеупомянутого, была рассмотрена и проанализирована зависимость времени выполнения отдельных этапов обработки данных, содержащихся в канонической модели, от выбранного способа преобразования модели системы (при использовании обобщения и без использования).

ЗАКЛЮЧЕНИЕ

В данной работе была рассмотрена проблема организации процесса тестирования информационных систем при отсутствии полноценного набора спецификаций. Библиографический анализ показал, что большинство использующихся практик построения тестового комплекта для отдельного процесса системы основывается на использовании набора требований к поведению тестируемого программного обеспечения, что оказывается не применимым в случае отсутствия таковых спецификаций.

Для решения проблемы разработки тестового набора был предложен подход, заключающийся в использовании методов интеллектуального анализа процессов для построения тестовой модели системы. При этом подразумевается, что исследуемая система инструментирована механизмом регистрации событий, связанных с осуществлением взаимодействий между модулями этой системы.

Так, в качестве цели работы выдвигалась разработка метода автоматизированного построения тестовой модели процесса системы на основе анализа трассы событий. Для достижения поставленной цели был решен ряд задач: выделение и обоснование канонической модели системы, создание адаптеров преобразований трасс в общеупотребительных форматах к канонической модели, реализация механизмов фильтрации, обобщения и визуализации полученного представления, а также разработка способа генерации тестовых кейсов для рассматриваемой системы.

Программная реализация метода извлечения тестовой модели была выполнена на языке программирования Java (JDK 10.0.1). Разработанное приложение позволяет работать с логами значительных объемов (порядка 10^5 событий) в форматах log4j2, XES и MXML, применять к полученной модели алгоритмы обобщения и фильтрации с различными параметрами, а также

генерировать данные, предназначенные для использования при составлении тестовых случаев.

В итоге, поставленная цель была достигнута: метод и соответствующая программная реализация извлечения тестовой модели были получены; разработанное приложение было протестировано на трассах, выгруженных с реальных систем.

Рассмотренный подход в перспективе может быть усовершенствован путем использования альтернативных алгоритмов преобразования извлеченной модели системы для достижения более подробного результирующего представления или же с целью получения более абстрактной модели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Balalaie A., Heydarnoori A., Jamshidi P. Microservices Architecture Enables DevOps: an Experience Report on Migration to a Cloud-Native Architecture // IEEE Software. 2016. Vol. 33(3). P. 42-52.
2. Федотов В. Н. Автоматизация регрессионного тестирования при помощи анализа трасс событий // Труды ИСП РАН. 2013. № 24.
3. van der Aalst W. M. P. Process mining: discovery, conformance and enhancement of business processes. Berlin. Springer, 2011. 352 p.
4. Sonawane S. B., Patki R. P. Process Mining by using Event Logs // International Journal of Computer Applications. 2015. Vol. 116(19). P. 31-35.
5. van der Aalst W. M. P. Business alignment: using process mining as a tool for Delta analysis and conformance testing // Requirements Engineering. 2005. Vol. 10(3). P. 198-211.
6. Janes A., Test Case Generation and Prioritization: A Process-Mining Approach // 2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). Tokyo. 2017. P. 38-39.
7. Valdman J. Log file analysis, Tech. Rep. DCSE/TR-2001-04, 2001.
8. Grace J., Maheswari V., Dhinaharan N. Analysis of Web Logs And Web User In Web Mining // International Journal of Network Security & Its Applications. 2011. Vol. 3(1). P. 99-110.
9. IEEE Computational Intelligence Society: IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams, IEEE Std 1849-2016. 2016.
10. van Dongen B. F. A Meta Model for Process Mining Data // In Proceedings of the CAiSE WORKSHOPS. 2005. P. 309-320.
11. Библиотека логирования log4j2 [Электронный ресурс]. – URL: https://logging.apache.org/log4j/2.x/manual/index.html.

12. 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 (SYRCoSE2014). 2014. P. 88-95.
13. van der Aalst W. M. P., Rubin V., H. M. W. Verbeek, van Dongen B. F., Kindler E., Günther C. W. Process mining: A two-step approach to balance between underfitting and overfitting // Software and Systems Modeling. 2008. Vol. 9(1). P. 87-111.
14. Buijs J. C. A. M. Flexible evolutionary algorithms for mining structured process models. Eindhoven: Technische Universiteit Eindhoven. 2014.
15. van der Aalst W. M. P., Rubin V., van Dongen B. F., Kindler E., Günther C. W. Process mining: A two-step approach using transition systems and regions // Technical Report BPM Center Report BPM-06-30, Department of Technology Management, Eindhoven University of Technology. 2006.
16. Кулямин В. В. Методы верификации программного обеспечения. Конкурс обзорно-аналитических статей по направлению «Информационно-телекоммуникационные системы», 2008.
17. Ryser J., Glinz M. A Scenario-Based Approach to Validating and Testing Software Systems Using Statecharts // Proc. of 12-th International Conference on Software and Systems Engineering and Their Applications (ICSSEA '99), 1999.
18. Платформа разработки приложений Java Standard Edition [Электронный ресурс]. – URL: https://www.oracle.com/technetwork/java/javase/overview/index.html
19. Фреймворк для работы с XML-файлами JAXB [Электронный ресурс]. – URL: https://javaee.github.io/jaxb-v2.
20. Библиотека для работы с графами JGraphT [Электронный ресурс]. – URL: https://jgrapht.org.
21. Программное обеспечение для визуализации графов Graphviz [Электронный ресурс]. – URL: https://www.graphviz.org.

22. Репозиторий логов событий реальных систем [Электронный ресурс]. – URL: https://data.4tu.nl/repository/collection:event_logs_real.
23. Leemans M. NASA Crew Exploration Vehicle (CEV) Software Event Log. Eindhoven University of Technology [Электронный ресурс]. – URL: https://doi.org/10.4121/uuid:60383406-ffcd-441f-aa5e-4ec763426b76.
24. van Dongen B. F. BPI Challenge 2012. 4TU.Centre for Research Data [Электронный ресурс]. – URL: https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f.
25. Zhang Q., Karcher A. System Reverse Engineering to Requirements and Tests // ICONS 2012: The Seventh International Conference on Systems. 2012. P. 35-38.
26. Rajal J. S., Sharma S. A Review on Various Techniques for Regression Testing and Test Case Prioritization // International Journal of Computer Applications. 2015. Vol. 116(16). P. 8-13.
27. Firesmith D. Common Requirements Problems, Their Negative Consequences, and the Industry Best Practices to Help Solve Them // Journal of Object Technology. 2007. Vol. 6(1). P. 17-33.
28. Hassan S., Qamar U., Hassan T. Software Reverse Engineering to Requirement Engineering for Evolution of Legacy System // IT Convergence and Security (ICITCS), 5th International Conference on IEEE. 2015.
29. Itkonen J., Rautiainen K. Exploratory testing: A multiple case study // International Symposium on Empirical Software Engineering (ISESE). 2005. P. 84-93.

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ ИЗВЛЕЧЕННЫХ ТЕСТОВЫХ КЕЙСОВ

```
</Traversal>
<Traversal>
  <Action>cev.CEV()  </Action>
  <References>  </References>
  <Action>cev.ErrorLog()  </Action>
  <References>1  </References>
  <Action>cev.Failures(cev.ErrorLog)  </Action>
  <References>2  </References>
  <Action>cev.Spacecraft(cev.Failures,cev.ErrorLog)  </Action>
  <References>3  </References>
  <Action>cev.CEV()  </Action>
  <References>  </References>
  <Action>cev.CEV.completion()  </Action>
  <References>  </References>
  <Action>cev.Failures.noLAS_CNTRLfailure()  </Action>
  <References>6  </References>
  <Action>cev.Spacecraft.doLowActiveAbort()  </Action>
  <References>7  </References>
  <Action>cev.Failures.noEARTH_SENSORfailure()  </Action>
  <References>8  </References>
  <Action>cev.CEV.completion()  </Action>
  <References>  </References>
  <Action>cev.CEV.edsSeparation()  </Action>
  <References>  </References>
  <Action>cev.Spacecraft.doEDSseparation()  </Action>
  <References>11  </References>
  <Action>cev.CEV.edsSeparation()  </Action>
  <References>  </References>
  <Action>cev.CEV.deOrbit()  </Action>
  <References>  </References>
  <Action>cev.Spacecraft.readyForDeorbit()  </Action>
  <References>14  </References>
  <Action>cev.CEV.deOrbit()  </Action>
  <References>  </References>
</Traversal>
<Traversal>
  <Action>cev.CEV()  </Action>
  <References>  </References>
  <Action>cev.ErrorLog()  </Action>
  <References>1  </References>
  <Action>cev.Failures(cev.ErrorLog)  </Action>
  <References>2  </References>
  <Action>cev.Spacecraft(cev.Failures,cev.ErrorLog)  </Action>
  <References>3  </References>
  <Action>cev.CEV()  </Action>
  <References>  </References>
  <Action>cev.CEV.completion()  </Action>
  <References>  </References>
  <Action>cev.Failures.noLAS_CNTRLfailure()  </Action>
  <References>6  </References>
  <Action>cev.Spacecraft.doLowActiveAbort()  </Action>
  <References>7  </References>
  <Action>cev.Failures.noEARTH_SENSORfailure()  </Action>
  <References>8  </References>
  <Action>cev.CEV.completion()  </Action>
  <References>5  </References>
  <Action>cev.CEV.eiBurn(boolean,boolean)  </Action>
  <References>  </References>
  <Action>cev.Spacecraft.readyForEiBurn()  </Action>
  <References>  </References>
  <Action>cev.Spacecraft.configurationsize()  </Action>
  <References>12  </References>
  <Action>cev.Spacecraft.readyForEiBurn()  </Action>
  <References>  </References>
  <Action>cev.Spacecraft.doEiBurn(boolean,boolean)  </Action>
  <References>14  </References>
  <Action>cev.CEV.eiBurn(boolean,boolean)  </Action>
  <References>  </References>
  <Action>cev.CEV.tliBurn()  </Action>
  <References>  </References>
  <Action>cev.Spacecraft.readyForTliBurn()  </Action>
  <References>  </References>
  <Action>cev.ErrorLog.log(java.lang.String)  </Action>
  <References>18  </References>
  <Action>cev.Spacecraft.readyForTliBurn()  </Action>
  <References>  </References>
  <Action>cev.ErrorLog.last()  </Action>
  <References>20  </References>
  <Action>cev.CEV.tliBurn()  </Action>
  <References>  </References>
</Traversal>
<Traversal>
  <Action>cev.CEV()  </Action>
  <References>  </References>
  <Action>cev.ErrorLog()  </Action>
  ...
```

Рис. А.1. Расширенный фрагмент извлеченного из activity map абстрактного тестового кейса для трассы [23]

```
<Testcase>
  <Action>
    cev.CEV()
  </Action>
  <Dataset>
    apprun:threadid: 1
    org:resource: 1
    apploc:joinpoint: cev.CEV()
    apprun:nanotime: 363666988705274
    apploc:filename: CEV.java
    apploc:app: symdym
    apploc:linenr: 20
    apploc:node: 0.1-SNAPSHOT
    lifecycle:transition: start
    apploc:etype: call_new
    apploc:tier: symdym-examples
    apploc:regionstr:
  </Dataset>
  <Action>
    cev.ErrorLog()
  </Action>
  <Dataset>
    apprun:threadid: 1
    org:resource: 1
    apploc:joinpoint: cev.ErrorLog()
    apprun:nanotime: 363666991619724
    apploc:filename: ErrorLog.java
    apploc:app: symdym
    apploc:linenr: 29
    apploc:node: 0.1-SNAPSHOT
    lifecycle:transition: start
    apploc:etype: call_new
    apploc:tier: symdym-examples
    apploc:regionstr:
  </Dataset>
  <Action>
    cev.ErrorLog()
  </Action>
  <Dataset>
    apprun:threadid: 1
    org:resource: 1
    apploc:joinpoint: cev.ErrorLog()
    apprun:nanotime: 363666991759994
    apploc:filename: ErrorLog.java
    apploc:app: symdym
    apploc:linenr: 29
    apploc:node: 0.1-SNAPSHOT
    lifecycle:transition: complete
    apploc:etype: return_new
    apploc:tier: symdym-examples
    apploc:regionstr:
  </Dataset>
  <Action>
    cev.Failures(cev.ErrorLog)
  </Action>
  <Dataset>
    apprun:threadid: 1
    org:resource: 1
    apploc:joinpoint: cev.Failures(cev.ErrorLog)
    apprun:nanotime: 363666995326615
    apploc:filename: Failures.java
    apploc:app: symdym
    apploc:linenr: 31
    apploc:node: 0.1-SNAPSHOT
    lifecycle:transition: start
    apploc:etype: call_new
    apploc:tier: symdym-examples
    apploc:regionstr:
  </Dataset>
  <Action>
    cev.Failures(cev.ErrorLog)
  </Action>
  <Dataset>
    apprun:threadid: 1
    org:resource: 1
    apploc:joinpoint: cev.Failures(cev.ErrorLog)
    apprun:nanotime: 363667031957176
    apploc:filename: Failures.java
    apploc:app: symdym
    apploc:linenr: 31
    apploc:node: 0.1-SNAPSHOT
    lifecycle:transition: complete
    apploc:etype: return_new
    apploc:tier: symdym-examples
    ...
```

Рис. А.2. Расширенный фрагмент извлеченного из LTS абстрактного тестового кейса для трассы [23]

ПРИЛОЖЕНИЕ Б
ЛИСТИНГ ПРОГРАММНЫХ МОДУЛЕЙ

1. Файл ActivityMap.java пакета model.

```
package model;

import java.io.*;
import java.util.*;
import org.jgrapht.graph.DefaultDirectedGraph;
import org.jgrapht.graph.DefaultEdge;

public class ActivityMap extends
    DefaultDirectedGraph<ActivityMap.AMVertex, DefaultEdge> {

    private static final long serialVersionUID = 19756914L;
    private List<ReferencedSequence> relatedContent;

    public static class AMVertex {
        private String activity;

        public AMVertex() {}

        public AMVertex(String activity) {
            this.activity = activity;
        }

        public String getActivity() {
            return activity;
        }

        public void setActivity(String activity) {
            this.activity = activity;
        }
    }

    public ActivityMap() {
        super(DefaultEdge.class);
    }

    public List<ReferencedSequence> getRelatedContent() {
        return this.relatedContent;
    }

    public static List<ActivityMap> emulateReferencedSequences(
            List<ReferencedSequence> refSeqList) {

        List<ActivityMap> atlas = new ArrayList<>();
        Map<String, Set<Integer>> mapCatalog = new HashMap<>();
        for (int i = 0; i < refSeqList.size(); ++i) {

            ReferencedSequence rs = refSeqList.get(i);
            String firstActivity = rs.getContent().x.get(0).getActivity();
            Set<Integer> associatedIndices = mapCatalog.get(firstActivity);
            if (associatedIndices == null) {
                associatedIndices = new HashSet<>();
```

```
    }

    associatedIndices.add(i);
    mapCatalog.put(firstActivity, associatedIndices);
}

System.out.println("discovered " +
        mapCatalog.size() + " equivalence case class(es)");
for (Map.Entry<String, Set<Integer>> entry : mapCatalog.entrySet()) {

    ActivityMap actMap = new ActivityMap();
    AMVertex root = new AMVertex(entry.getKey());
    actMap.relatedContent = new ArrayList<>();
    actMap.addVertex(root);

    // iterating over cases having the same first activity
    for (Integer idx : entry.getValue()) {

        ReferencedSequence rs = refSeqList.get(idx);
        actMap.relatedContent.add(rs);

        List<Event> eventList = rs.getContent().x;
        List<HashSet<Integer>> refList = rs.getContent().y;
        List<AMVertex> visitedVertices = new ArrayList<>();
        List<HashSet<AMVertex>> pendingReferals = new ArrayList<>();

        for (int i = 0; i < refList.size(); ++i) {
            pendingReferals.add(new HashSet<>());
        }

        AMVertex curVertex = root;
        visitedVertices.add(curVertex);

        // processing root element references
        for (Integer refIdx : refList.get(0)) {
            if (refIdx == 0) {
                DefaultEdge edge = new DefaultEdge();
                actMap.addEdge(root, root, edge);
            }

            else {
                if (refIdx < refList.size()) {
                    pendingReferals.get(refIdx).add(curVertex);
                }
            }
        }

        for (int i = 1; i < eventList.size(); ++i) {
            String activity = eventList.get(i).getActivity();
            AMVertex toVertex = null;
            for (DefaultEdge outgoingE :
                actMap.outgoingEdgesOf(curVertex)) {

                AMVertex outgoingV = actMap.getEdgeTarget(outgoingE);
                if (outgoingV.getActivity().equals(activity)) {
                    toVertex = outgoingV;
                    break;
                }
            }
            // not found
            if (toVertex == null) {
```

```
                    toVertex = new AMVertex(activity);
                    actMap.addVertex(toVertex);
                }

                DefaultEdge edge = new DefaultEdge();
                actMap.addEdge(curVertex, toVertex, edge);
                curVertex = toVertex;
                visitedVertices.add(curVertex);

                for (Integer refIdx : refList.get(i)) {
                    if (refIdx < visitedVertices.size()) {
                        DefaultEdge refEdge = new DefaultEdge();
                        actMap.addEdge(curVertex,
                                visitedVertices.get(refIdx), refEdge);
                    }
                    else {
                        if (refIdx < refList.size()) {
                            pendingReferals.get(refIdx).add(curVertex);
                        }
                    }
                }

                Set<AMVertex> pending = pendingReferals.get(i);
                for (AMVertex referal : pending) {
                    DefaultEdge refEdge = new DefaultEdge();
                    actMap.addEdge(referal, curVertex, refEdge);
                }
                pending.clear();
            }
        }
        atlas.add(actMap);
    }
    return atlas;
}

public void exportTestData(String path) {

    File file = new File(path);
    FileWriter fw = null;
    BufferedWriter bw = null;

    try {

        fw = new FileWriter(file);
        bw = new BufferedWriter(fw);
        bw.write("<?xml version=\"1.0\" encoding=\"UTF-8\" ?>");
        bw.newLine();

        for (ReferencedSequence rSeq : this.relatedContent) {

            bw.write("<Traversal>");
            bw.newLine();
            Pair<List<Event>, List<HashSet<Integer>>> content =
                    rSeq.getContent();
            for (int i = 0; i < content.x.size(); ++i) {

                bw.write("  <Action>");
                bw.write(content.x.get(i).getActivity());
                bw.write("  </Action>");
                bw.newLine();
                bw.write("  <References>");
```

```
                    for (Integer refIdx : content.y.get(i)) {
                        bw.write(refIdx + " ");
                    }

                    bw.write("  </References>");
                    bw.newLine();
                }

                bw.write("</Traversal>");
                bw.newLine();
            }
        }

        catch (IOException e) {
            e.printStackTrace();
        }

        finally {
            try {
                bw.close();
                fw.close();
            }

            catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

2. Файл Canonical.java пакета model.

```
package model;

import java.util.ArrayList;
import java.util.List;
import java.time.temporal.ChronoUnit;


/**
 * This class represents canonical process model extracted from some event log.
 * Term "canonical" is considered as "unified" means that all further model
 * processing will be using only this unified representation.
 */
public class Canonical {

    private enum CompareResult {
        SAME,
        DIFFERENT,
        PREFIX,
        POSTFIX
    }

    // Content describing particular process data. TaggedList.list is a
    // sequence of events with similar caseId; corresponding TaggedList.tag
    // holds number of that sequence occurrences in the model.
    // Initially all the tags must be 1, later they may be merged with
    // appropriate counter increment.
    private List<TaggedList> taggedSequences = new ArrayList<>();
```

```
private double prefixFrequencyThreshold;
private double postfixFrequencyThreshold;
private double missedFrequencyThreshold;
private long deltaT;

public List<TaggedList> getTaggedSequences() {
    return this.taggedSequences;
}

public void setTaggedSequences(List<TaggedList> taggedSequences) {
    this.taggedSequences = taggedSequences;
}

public void setPrefixFrequencyThreshold(double prefixFrequencyThreshold) {
    this.prefixFrequencyThreshold = prefixFrequencyThreshold;
}

public void setPostfixFrequencyThreshold(double postfixFrequencyThreshold) {
    this.postfixFrequencyThreshold = postfixFrequencyThreshold;
}

public void setMissedFrequencyThreshold(double missedFrequencyThreshold) {
    this.missedFrequencyThreshold = missedFrequencyThreshold;
}

public void setDeltaT(long deltaT) {
    this.deltaT = deltaT;
}

private CompareResult compareByActivities(List<Event> first,
        List<Event> second) {

    if (first.size() == second.size()) {

        for (int i = 0; i < first.size(); ++i) {
            if (!first.get(i).getActivity().
                equals(second.get(i).getActivity())) {

                return CompareResult.DIFFERENT;
            }
        }

        return CompareResult.SAME;
    }

    else {

        List<Event> master = (first.size() > second.size())? first: second;
        List<Event> slave = (first == master)? second: first;
        boolean prefixChecking = true, postfixChecking = true;
        int sSize = slave.size();

        for (int i = 0; i < sSize; ++i) {
            if (prefixChecking) {
                if (!master.get(i).getActivity().
                    equals(slave.get(i).getActivity())) {

                    prefixChecking = false;
                }
            }
```

```
            if (postfixChecking) {
                if (!master.get(master.size()-sSize+i).getActivity().
                    equals(slave.get(i).getActivity())) {

                    postfixChecking = false;
                }
            }
        }

        if (prefixChecking) { return CompareResult.PREFIX; }
        if (postfixChecking) { return CompareResult.POSTFIX; }
        return CompareResult.DIFFERENT;
    }
}

// merges sequences with the same Event.activity lists
private void mergeSimilar() {

    int baseIdx = 0;
    while (baseIdx < taggedSequences.size()) {

        TaggedList baseElem = taggedSequences.get(baseIdx);
        List<Integer> idxsToRemove = new ArrayList<>();
        for (int i = baseIdx+1; i < taggedSequences.size(); ++i) {

            CompareResult res = compareByActivities(baseElem.list,
                    taggedSequences.get(i).list);

            if (res == CompareResult.SAME) {
                baseElem.tag++;
                idxsToRemove.add(i);
            }
        }

        List<TaggedList> toRemove = new ArrayList<>();
        for (Integer each : idxsToRemove) {
            toRemove.add(taggedSequences.get(each));
        }

        taggedSequences.removeAll(toRemove);
        baseIdx++;
    }
}

// filtering prefix & postfix cases which are less frequent than threshold
private void filterOutAnyfixes() {

    int foundPrefs = 0, foundPosts = 0,
        filteredPrefs = 0, filteredPosts = 0;
    List<TaggedList> toRemove = new ArrayList<>();

    for (int i = 0; i < taggedSequences.size(); ++i) {
        for (int j = i+1; j < taggedSequences.size(); ++j) {

            TaggedList first  = taggedSequences.get(i);
            TaggedList second = taggedSequences.get(j);
            CompareResult res = compareByActivities(
                    first.list, second.list);

            // optimizing memory usage by passing non-anyfix cases
            if (res == CompareResult.DIFFERENT || res == CompareResult.SAME)
```

```
                continue;

            TaggedList candidate = (first.list.size() > second.list.size())?
                    second: first;
            double patternOccurrencies = candidate.tag;
            // LRF = local relative frequency
            double lrf = patternOccurrencies / (first.tag+second.tag);

            if (res == CompareResult.PREFIX) {

                foundPrefs++;
                if (lrf < prefixFrequencyThreshold) {

                    filteredPrefs++;
                    toRemove.add(candidate);
                }
            }

            if (res == CompareResult.POSTFIX) {

                foundPosts++;
                if (lrf < postfixFrequencyThreshold) {

                    filteredPosts++;
                    toRemove.add(candidate);
                }
            }
        }
    }

    taggedSequences.removeAll(toRemove);
    System.out.println("successfully filtered out:");
    System.out.println("--cases: " + toRemove.size() + " with");
    System.out.println("--prefix occurrencies: " + filteredPrefs +
            "/" + foundPrefs);
    System.out.println("--postfix occurrencies: " + filteredPosts +
            "/" + foundPosts);
}

// returns null if considered as not containing missed events
private TaggedList locateMissedEvent(TaggedList tList1, TaggedList tList2) {

    int n1 = tList1.list.size();
    int n2 = tList2.list.size();

    // checking sequences that are different by one event
    if (Math.abs(n1 - n2) != 1) {
        return null;
    }

    TaggedList candidate = (n1 > n2)? tList2: tList1;
    TaggedList masterList = (candidate == tList1)? tList2: tList1;
    double lrf = candidate.tag / (masterList.tag + candidate.tag);
    if (lrf > this.missedFrequencyThreshold) {
        return null;
    }

    List<Event> reduced = candidate.list;
    List<Event> sterling = masterList.list;
    boolean untilMode = true;
```

```
    for (int i = 0; i < candidate.list.size(); ++i) {

        // until-missed mode
        if (untilMode) {
            if (!reduced.get(i).getActivity().
                equals(sterling.get(i).getActivity())) {

                // skipping such sequence pair because postfix cases
                // were filtered on the previous refinement stage
                if (i == 0) {
                    return null;
                }

                List<Event> reducedFragment = reduced.subList(i-1, i+1);
                List<Event> sterlingFragment = sterling.subList(i-1, i+2);

                // further event equality checking
                if (!reducedFragment.get(1).getActivity().
                    equals(sterlingFragment.get(2).getActivity())) {

                    return null;
                }

                // timestamp checking
                if (Math.abs(reducedFragment.get(0).getTimestamp().
                        until(reducedFragment.get(1).getTimestamp(),
                                ChronoUnit.MILLIS) -
                        sterlingFragment.get(0).getTimestamp().
                        until(sterlingFragment.get(2).getTimestamp(),
                                ChronoUnit.MILLIS)) > this.deltaT) {

                    return null;
                }

                untilMode = false;
            }
        }

        // after-missed mode
        else {
            if (!reduced.get(i).getActivity().
                equals(sterling.get(i+1).getActivity())) {

                return null;
            }
        }
    }

    return candidate;
}

public void filterOutMissedEvents() {

    int filteredCases = 0;
    List<TaggedList> toRemove = new ArrayList<>();

    for (int i = 0; i < this.taggedSequences.size(); ++i) {
        for (int j = i+1; j < this.taggedSequences.size(); ++j) {

            TaggedList tList1 = this.taggedSequences.get(i);
            TaggedList tList2 = this.taggedSequences.get(j);
```

```
                TaggedList candidate = locateMissedEvent(tList1, tList2);

                if (candidate != null) {
                    filteredCases++;
                    toRemove.add(candidate);
                }
            }
        }

        this.taggedSequences.removeAll(toRemove);
        System.out.println("successfully filtered out:");
        System.out.println("--cases w/missed event: " + filteredCases);
    }

    // aggregation method for summarizing all internal processing
    public void refineData() {

        System.out.println("refining canonical model...");
        System.out.println("initial cases number: " +
                this.taggedSequences.size());
        this.mergeSimilar();
        System.out.println("merged to " +
                this.taggedSequences.size() + " case(s)");
        this.filterOutAnyfixes();
        this.filterOutMissedEvents();
    }
}
```

3. Файл Event.java пакета model.

```
package model;

import java.time.ZonedDateTime;
import java.util.Map;

public class Event {

    private int caseId;
    private String activity;
    private ZonedDateTime timestamp;
    private Map<String, String> extra;

    public int getCaseId() {
        return this.caseId;
    }

    public void setCaseId(int caseId) {
        this.caseId = caseId;
    }

    public String getActivity() {
        return this.activity;
    }

    public void setActivity(String activity) {
        this.activity = activity;
    }
```

```
    public ZonedDateTime getTimestamp() {
        return this.timestamp;
    }

    public void setTimestamp(ZonedDateTime timestamp) {
        this.timestamp = timestamp;
    }

    public Map<String, String> getExtra() {
        return this.extra;
    }

    public void setExtra(Map<String, String> extra) {
        this.extra = extra;
    }
}
```

4. Файл Pair.java пакета model.

```
package model;

// container class
public class Pair<U, V> {
    public U x;
    public V y;
}
```

5. Файл ReferencedSequence.java пакета model.

```
package model;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;

/**
 * Describes list of Event class objects where each of them can have a set of
 * references to other elements of the sequence.
 */
public class ReferencedSequence {

    private Pair<List<Event>, List<HashSet<Integer>>> content;

    // copies references to `events` into the internal structure
    public ReferencedSequence(List<Event> events) {

        List<Event> eventSequence = new ArrayList<>();
        List<HashSet<Integer>> refSequence = new ArrayList<>();
        for (Event e : events) {
            eventSequence.add(e);
            refSequence.add(new HashSet<>());
        }

        Pair<List<Event>, List<HashSet<Integer>>> pair = new Pair<>();
        pair.x = eventSequence;
```

```
    pair.y = refSequence;
    this.content = pair;
}

public Pair<List<Event>, List<HashSet<Integer>>> getContent() {
    return this.content;
}

private boolean checkPatternIntegration(List<Event> pattern, int indexAt) {

    List<Event> list = this.content.x;
    if (indexAt + pattern.size() > list.size()) { return false; }
    for (int i = 0; i < pattern.size(); ++i) {
        if (!list.get(indexAt + i).getActivity().
            equals(pattern.get(i).getActivity())) {

            return false;
        }
    }

    return true;
}

private int countPatternMatches(List<Event> pattern, int from) {

    List<Event> list = this.content.x;
    int repeats = 0, i = from;
    while (i < list.size()) {
        if (checkPatternIntegration(pattern, i)) {
            repeats++;
            i += pattern.size();
        }

        else { break; }
    }

    return repeats;
}

// collects references from patterns next to
// the one starting on pivotIdx and mapping them on start pattern indices
private void adjustReferences(int patternSize, int pivotIdx, int repeats) {

    List<HashSet<Integer>> refList = this.content.y;
    for (int n = 1; n <= repeats; ++n) {

        // inspecting n-th similar pattern, e.g.
        //
        //           _pivotIdx=4
        // a b c g | d e f | d e f | d e f |
        //                    n=1     n=2

        // i -- pattern-relative offset
        for (int i = 0; i < patternSize; ++i) {

            HashSet<Integer> alignedRefs = new HashSet<>();
            for (Integer ref : refList.get(pivotIdx+patternSize*n+i)) {
                if (ref >= pivotIdx) {
                    alignedRefs.add(pivotIdx + (ref-pivotIdx)%patternSize);
                }
```

```
                else {
                    alignedRefs.add(ref);
                }
            }

            refList.get(pivotIdx+i).addAll(alignedRefs);
        }
    }
}

public void reduceLoops() {

    List<Event> eventList = this.content.x;
    List<HashSet<Integer>> refList = this.content.y;

    int pivotIdx = 0;
    while (pivotIdx < eventList.size()) {

        int i = pivotIdx+1;
        while (i < eventList.size()) {

            // found possible start of similar to
            // events.slice(pivotIdx, i-1) pattern
            if (eventList.get(pivotIdx).getActivity().
                equals(eventList.get(i).getActivity())) {

                final List<Event> pattern = eventList.subList(pivotIdx, i);
                int repeats = countPatternMatches(pattern, i);

                if (repeats > 0) {

                    // adding just detected back reference
                    refList.get(i-1).add(pivotIdx);
                    // collecting & refreshing refs from patterns
                    // that are going to reduce
                    adjustReferences(pattern.size(), pivotIdx, repeats);

                    int until = i+pattern.size()*repeats;
                    eventList.subList(i, until).clear();
                    refList.subList(i, until).clear();

                    // after the inner `while` break, next operation will
                    // be pivotIdx++; so we will start from beginning
                    pivotIdx = -1;
                    break;
                }
            }

            i++;
        }

        pivotIdx++;
    }
}

private boolean isEqual(ReferencedSequence another) {

    Pair<List<Event>, List<HashSet<Integer>>> content1 = this.content;
    Pair<List<Event>, List<HashSet<Integer>>> content2 = another.content;
    if (content1.x.size() != content2.x.size()) {
        return false;
```

```
        }

        for (int i = 0; i < content1.x.size(); ++i) {

            if (!content1.x.get(i).getActivity().
                equals(content2.x.get(i).getActivity()) ||
                !content1.y.get(i).
                equals(content2.y.get(i))) {

                return false;
            }
        }

        return true;
    }

    public static List<ReferencedSequence> generalizeModel(Canonical can) {

        List<ReferencedSequence> rSeqList = new ArrayList<>();
        for (TaggedList tList : can.getTaggedSequences()) {

            ReferencedSequence rSeq = new ReferencedSequence(tList.list);
            rSeq.reduceLoops();

            // removing similar sequences
            boolean isUnique = true;
            for (ReferencedSequence each : rSeqList) {
                if (rSeq.isEqual(each)) {
                    isUnique = false;
                    break;
                }
            }

            if (isUnique) { rSeqList.add(rSeq); }
        }

        return rSeqList;
    }
}
```

6. Файл TaggedList.java пакета model.

```
package model;

import java.util.List;

// container class
public class TaggedList {

    public List<Event> list;
    public Integer tag;

    public TaggedList(List<Event> list, Integer tag) {

        this.list = list;
        this.tag = tag;
    }
}
```

7. Файл Translator.java пакета model.

```
package model;

import java.time.ZonedDateTime;
import java.time.ZoneId;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.*;
import model.log4j.*;
import model.mxml.*;
import model.xes.*;

public class Translator {

    private static final DateTimeFormatter log4jFormatter =
            DateTimeFormatter.ofPattern("dd.MM.yyyy'T'HH:mm:ss").
            withZone(ZoneId.of("UTC"));
    private static final DateTimeFormatter mxmlFormatter =
            DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH:mm:ss.SSSXXX");
    private static final DateTimeFormatter xesFormatter =
            DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH:mm:ss.SSSXXX");
    private static String customPattern = null;

    public static void setCustomPattern(String pattern) {
        customPattern = pattern;
    }

    public static Canonical castLog4j(LogLog4j log) {

        List<EventLog4j> events = log.getEventList();
        Canonical output = new Canonical();
        if (events.isEmpty()) {
            return output;
        }

        DateTimeFormatter formatter = (customPattern == null)? log4jFormatter:
            DateTimeFormatter.ofPattern(customPattern);
        Event earliest = new Event(),
                latest = new Event();
        ZonedDateTime found = ZonedDateTime.parse(events.get(0).timestamp,
                formatter);
        earliest.setTimestamp(found);
        latest.setTimestamp(found);

        List<TaggedList> taggedSequences = new ArrayList<>();
        Map<Integer, Integer> caseNumbers = new HashMap<>();

        for (EventLog4j each : events) {

            int caseId = Integer.parseInt(each.processIdentifier);
            Integer correspondingIdx = caseNumbers.get(caseId);
            TaggedList tList = null;

            // haven't met such caseId
            if (correspondingIdx == null) {

                List<Event> eventList = new ArrayList<>();
```

```
            tList = new TaggedList(eventList, 1);
            caseNumbers.put(caseId, taggedSequences.size());
            taggedSequences.add(tList);
        }

        else {
            tList = taggedSequences.get(correspondingIdx);
        }

        Event event = new Event();
        event.setCaseId(caseId);
        event.setActivity(each.className + ":" + each.methodName);
        event.setTimestamp(ZonedDateTime.parse(each.timestamp,
                formatter));

        Map<String, String> extra = new HashMap<>();
        extra.put("Data", each.extraInfo);
        event.setExtra(extra);
        tList.list.add(event);

        if (earliest.getTimestamp().
                until(event.getTimestamp(), ChronoUnit.MILLIS) < 0) {
            earliest = event;
        }

        if (event.getTimestamp().
                until(latest.getTimestamp(), ChronoUnit.MILLIS) < 0) {
            latest = event;
        }
    }

    ZonedDateTime from = earliest.getTimestamp(),
            to = latest.getTimestamp();
    System.out.println("parsed log with " + events.size() + " events");
    System.out.println("captured time fragment from " +
            from.format(formatter) + " to " +
            to.format(formatter) + " (" +
            from.until(to, ChronoUnit.MILLIS) + " ms)");

    output.setTaggedSequences(taggedSequences);
    return output;
}

public static Canonical castMXML(LogMXML log) {

    Canonical output = new Canonical();
    if (log.getProcessList().isEmpty()) {
        return output;
    }

    DateTimeFormatter formatter = (customPattern == null)? mxmlFormatter:
        DateTimeFormatter.ofPattern(customPattern);
    int eventNumber = 0,
            caseNum = 0;
    List<TaggedList> taggedSequences = new ArrayList<>();
    ProcessMXML process = log.getProcessList().get(0);
    ZonedDateTime from = ZonedDateTime.of(2039, 1, 1,
            0, 0, 0, 0, ZoneId.systemDefault());
    ZonedDateTime to = ZonedDateTime.of(1970, 1, 1,
            0, 0, 0, 0, ZoneId.systemDefault());
```

```
    for (CaseMXML cMXML : process.getCaseList()) {

        List<Event> eventList = new ArrayList<>();
        for (EventMXML eMXML : cMXML.getEventList()) {

            Event event = new Event();
            Map<String, String> extra = new HashMap<>();
            if (eMXML.getData() != null) {

                List<DataMXML.Attr> attrs = eMXML.getData().getAttrList();
                for (DataMXML.Attr attr : attrs) {
                    extra.put(attr.getName(), attr.getValue());
                }
            }

            event.setCaseId(caseNum);
            event.setActivity(eMXML.getWorkflowModelElement());
            event.setTimestamp(ZonedDateTime.parse(eMXML.getTimestamp(),
                    formatter));
            event.setExtra(extra);
            eventList.add(event);
            eventNumber++;

            if (from.until(event.getTimestamp(), ChronoUnit.MILLIS) < 0) {
                from = event.getTimestamp();
            }

            if (event.getTimestamp().until(to, ChronoUnit.MILLIS) < 0) {
                to = event.getTimestamp();
            }
        }

        taggedSequences.add(new TaggedList(eventList, 1));
        caseNum++;
    }

    System.out.println("parsed log with " + eventNumber + " events");
    System.out.println("captured time fragment from " +
            from.format(formatter) + " to " +
            to.format(formatter) + " (" +
            from.until(to, ChronoUnit.MILLIS) + " ms)");

    output.setTaggedSequences(taggedSequences);
    return output;
}

public static Canonical castXES(LogXES log) {

    Canonical output = new Canonical();
    if (log.getCaseList().isEmpty()) {
        return output;
    }

    DateTimeFormatter formatter = (customPattern == null)? xesFormatter:
        DateTimeFormatter.ofPattern(customPattern);
    int eventNumber = 0,
            caseNum = 0;
    List<TaggedList> taggedSequences = new ArrayList<>();
    ZonedDateTime from = ZonedDateTime.of(2039, 1, 1,
            0, 0, 0, 0, ZoneId.systemDefault());
    ZonedDateTime to = ZonedDateTime.of(1970, 1, 1,
```

```
                0, 0, 0, 0, ZoneId.systemDefault());

        for (CaseXES cXES : log.getCaseList()) {

            List<Event> eventList = new ArrayList<>();
            for (EventXES eXES : cXES.getEventList()) {

                Event event = new Event();
                String activity = null;
                List<EventXES.StringXES> stringList = eXES.getStringList();
                Map<String, String> extra = new HashMap<>();

                for (EventXES.StringXES str : stringList) {
                    if (str.getKey().equals("concept:name")) {
                        activity = str.getValue();
                    }

                    else {
                        extra.put(str.getKey(), str.getValue());
                    }
                }

                event.setCaseId(caseNum);
                event.setActivity(activity);
                event.setTimestamp(
                        ZonedDateTime.parse(eXES.getDate().getValue(),
                        formatter));
                event.setExtra(extra);
                eventList.add(event);
                eventNumber++;

                if (from.until(event.getTimestamp(), ChronoUnit.MILLIS) < 0) {
                    from = event.getTimestamp();
                }

                if (event.getTimestamp().until(to, ChronoUnit.MILLIS) < 0) {
                    to = event.getTimestamp();
                }
            }

            taggedSequences.add(new TaggedList(eventList, 1));
            caseNum++;
        }

        System.out.println("parsed log with " + eventNumber + " events");
        System.out.println("captured time fragment from " +
                from.format(formatter) + " to " +
                to.format(formatter) + " (" +
                from.until(to, ChronoUnit.MILLIS) + " ms)");

        output.setTaggedSequences(taggedSequences);
        return output;
    }
}
```

8. Файл TransSystem.java пакета model.

```
package model;
```

```
import java.util.*;
import org.jgrapht.graph.DefaultDirectedWeightedGraph;
import org.jgrapht.graph.DefaultWeightedEdge;
import org.jgrapht.traverse.DepthFirstIterator;
import org.jgrapht.traverse.GraphIterator;
import java.io.*;

public class TransSystem extends
    DefaultDirectedWeightedGraph<TransSystem.TSVertex, TransSystem.TSEdge> {

    private static final long serialVersionUID = 19756912L;
    private double maxEdgeWeight = 1;

    public static class TSVertex {

        private static int globalId = 0;
        private int id;
        private List<Event> state = new ArrayList<>();

        public TSVertex() {
            this.id = globalId++;
        }

        public TSVertex(List<Event> state) {

            this.id = globalId++;
            for (Event each : state) {
                this.state.add(each);
            }
        }

        public List<Event> getState() {
            return this.state;
        }

        public List<Event> getStateClone() {

            List<Event> clone = new ArrayList<>();
            for (Event each : this.state) {
                clone.add(each);
            }
            return clone;
        }

        public void setState(List<Event> state) {
            for (Event each : state) {
                this.state.add(each);
            }
        }

        @Override
        public String toString() {
            return "state " + this.id;
        }
    }

    public static class TSEdge extends DefaultWeightedEdge {

        private static final long serialVersionUID = 81275L;
        private String label;
```

```
    private double weight;

    public TSEdge(String label, double weight) {

        this.label = label;
        this.weight = weight;
    }

    public String getLabel() {
        return this.label;
    }

    @Override
    public double getWeight() {
        return this.weight;
    }

    public void setWeight(double weight) {
        this.weight = weight;
    }

    @Override
    public String toString() {
        return "(" + this.getSource() + " : " +
                this.getTarget() + " : " + label + ")";
    }
}

public TransSystem() {
    super(TSEdge.class);
}

// initializes this TransSystem object with data containing in Canonical
public void emulateCanonical(Canonical can) {

    TSVertex root = new TSVertex();
    this.addVertex(root);
    // posVertex describes currently inspecting vertex
    TSVertex posVertex;
    double maxEdgeW = 1;

    for (TaggedList tList : can.getTaggedSequences()) {

        posVertex = root;
        for (Event e : tList.list) {

            TSEdge matchedEdge = null;
            for (TSEdge edge : outgoingEdgesOf(posVertex)) {
                if (edge.getLabel().equals(e.getActivity())) {
                    matchedEdge = edge;
                    break;
                }
            }

            // we have found edge with same marking as an event activity,
            // so increasing this edge weight & continuing traversal
            if (matchedEdge != null) {

                double adjustedWeight = matchedEdge.getWeight() + tList.tag;
                matchedEdge.setWeight(adjustedWeight);
                if (adjustedWeight > maxEdgeW) {
```

```
                    maxEdgeW = adjustedWeight;
                }

                posVertex = getEdgeTarget(matchedEdge);
            }

            // we need to create new vertex and connect it to posVertex
            // via newly created edge with a label e.activity
            else {

                        List<Event> newState = posVertex.getStateClone();
                        newState.add(e);
                        TSVertex newVertex = new TSVertex(newState);
                this.addVertex(newVertex);
                TSEdge newEdge = new TSEdge(e.getActivity(), tList.tag);
                this.addEdge(posVertex, newVertex, newEdge);
                posVertex = newVertex;
            }
        }
    }

    this.maxEdgeWeight = maxEdgeW;
}

public double getMaxEdgeWeight() {
    return this.maxEdgeWeight;
}

public void exportTestData(String path) {

    File file = new File(path);
    FileWriter fw = null;
    BufferedWriter bw = null;

    try {

        fw = new FileWriter(file);
        bw = new BufferedWriter(fw);
        GraphIterator<TSVertex, TSEdge> iter =
                new DepthFirstIterator<TSVertex, TSEdge>(this);
        bw.write("<?xml version=\"1.0\" encoding=\"UTF-8\" ?>");
        bw.newLine();

        while (iter.hasNext()) {

            TSVertex visited = iter.next();
            // this vertex is a leaf -> generate test case
            if (this.outgoingEdgesOf(visited).isEmpty()) {

                bw.write("<Testcase>");
                bw.newLine();
                for (Event each : visited.getState()) {

                    bw.write("  <Action>");
                    bw.newLine();
                    bw.write("    " + each.getActivity());
                    bw.newLine();
                    bw.write("  </Action>");
                    bw.newLine();

                    bw.write("  <Dataset>");
```

```
                        bw.newLine();
                        for (Map.Entry<String, String> entry :
                            each.getExtra().entrySet()) {

                            bw.write("    " + entry.getKey() +
                                    ": " + entry.getValue());
                            bw.newLine();
                        }

                        bw.write("  </Dataset>");
                        bw.newLine();
                    }

                    bw.write("</Testcase>");
                    bw.newLine();
                    bw.newLine();
                }
            }
        }

        catch (IOException e) {
            e.printStackTrace();
        }

        finally {
            try {
                bw.close();
                fw.close();
            }

            catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

9. Файл EventLog4j.java пакета model.log4j.

```
package model.log4j;

public class EventLog4j {

    public String timestamp;
    public String processIdentifier;
    public String className;
    public String methodName;
    public String extraInfo;

    // provides parsing log entries
    public static EventLog4j parseLogEntry(String entry, String pattern) {

        EventLog4j event = new EventLog4j();
        String[] entryTokens = entry.split(" ");
        String[] patternTokens = pattern.split(" ");
        String extraField = "";

        if (entryTokens.length != patternTokens.length) {
```

```
            System.err.println("log4j event is not parsed:" +
                    " pattern matching problem");
            return event;
        }

        for (int i = 0; i < entryTokens.length; ++i) {

            String pToken = patternTokens[i];
            int specifierIdx = pToken.indexOf("%");
            String specifierName = pToken.substring(specifierIdx+1);

            if (specifierName.equals("d")) {
                event.timestamp = entryTokens[i];
            }

            else if (specifierName.equals("pid")) {
                event.processIdentifier = entryTokens[i];
            }

            else if (specifierName.equals("C")) {
                event.className = entryTokens[i];
            }

            else if (specifierName.equals("M")) {
                event.methodName = entryTokens[i];
            }

            else {
                extraField = extraField + ";" + entryTokens[i];
            }
        }

        event.extraInfo = extraField;
        return event;
    }
}
```

10. Файл LogLog4j.java пакета model.log4j.

```
package model.log4j;

import java.util.List;
import java.util.ArrayList;
import java.io.*;

public class LogLog4j {

    private static String pattern = "%d %t %C %M %pid %m";
    private List<EventLog4j> eventList = new ArrayList<>();

    public static void setPattern(String newPattern) {
        pattern = newPattern;
    }

    public List<EventLog4j> getEventList() {
        return this.eventList;
    }
```

```
    public void setEventList(List<EventLog4j> eventList) {
        this.eventList = eventList;
    }

    public static LogLog4j extractLogLog4j(String path) {

        LogLog4j log = new LogLog4j();
        List<EventLog4j> events = log.getEventList();
        File file = new File(path);
        FileReader fr = null;
        BufferedReader br = null;

        try {

            fr = new FileReader(file);
            br = new BufferedReader(fr);
            String line = br.readLine();

            while (line != null) {

                EventLog4j event = EventLog4j.parseLogEntry(line, pattern);
                events.add(event);
                line = br.readLine();
            }
        }

        catch (IOException e) {
            e.printStackTrace();
        }

        finally {
            try {
                br.close();
                fr.close();
            }

            catch (IOException e) {
                e.printStackTrace();
            }
        }

        return log;
    }
}
```

11. Файл CaseMXML.java пакета model.mxml.

```
package model.mxml;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.*;

@XmlType(propOrder = { "eventList" })
public class CaseMXML {

    private List<EventMXML> eventList = new ArrayList<>();
    @XmlAttribute
```

```
    private int id;

    @XmlElement(name = "AuditTrailEntry")
    public List<EventMXML> getEventList() {
        return this.eventList;
    }

    public int getId() {
        return this.id;
    }
}
```

12. Файл DataMXML.java пакета model.mxml.

```
package model.mxml;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.*;

@XmlType(propOrder = { "attrList" })
public class DataMXML {

    @XmlAccessorType(XmlAccessType.FIELD)
    @XmlType(name = "Attribute", propOrder = { "value" })
    public static class Attr {

        @XmlValue
        private String value;
        @XmlAttribute(name = "name")
        private String name;

        public String getValue() {
            return this.value;
        }

        public String getName() {
            return this.name;
        }
    }

    private List<Attr> attrList = new ArrayList<>();

    @XmlElement(name = "Attribute")
    public List<Attr> getAttrList() {
        return this.attrList;
    }
}
```

13. Файл EventMXML.java пакета model.mxml.

```
package model.mxml;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;
```

```
@XmlType(propOrder = { "data", "workflowModelElement", "eventType",
        "timestamp", "originator" })
public class EventMXML {

    @XmlElement(name = "Data")
    private DataMXML data;
    @XmlElement(name = "WorkflowModelElement")
    private String workflowModelElement;
    @XmlElement(name = "EventType")
    private String eventType;
    @XmlElement(name = "Timestamp")
    private String timestamp;
    @XmlElement(name = "Originator")
    private String originator;

    public DataMXML getData() {
        return this.data;
    }

    public String getWorkflowModelElement() {
        return this.workflowModelElement;
    }

    public String getEventType() {
        return this.eventType;
    }

    public String getTimestamp() {
        return this.timestamp;
    }

    public String getOriginator() {
        return this.originator;
    }
}
```

14. Файл LogMXML.java пакета model.mxml.

```
package model.mxml;

import java.io.File;
import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import javax.xml.bind.annotation.*;

@XmlRootElement(name = "WorkflowLog")
@XmlType(propOrder = { "processList" })
public class LogMXML {

    private List<ProcessMXML> processList = new ArrayList<>();

    @XmlElement(name = "Process")
    public List<ProcessMXML> getProcessList() {
        return this.processList;
```

```
    }

    public static LogMXML extractLogMXML(String path) {
        try {

            JAXBContext context = JAXBContext.newInstance(LogMXML.class);
            Unmarshaller unmar = context.createUnmarshaller();
            LogMXML log = (LogMXML) unmar.unmarshal(new File(path));
            return log;
        }

        catch (JAXBException e) {
            e.printStackTrace();
        }

        return null;
    }
}
```

15. Файл ProcessMXML.java пакета model.mxml.

```
package model.mxml;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.*;


@XmlType(propOrder = {"caseList"})
public class ProcessMXML {

       private List<CaseMXML> caseList = new ArrayList<>();
       @XmlAttribute
       private String description;

       @XmlElement(name = "ProcessInstance")
       public List<CaseMXML> getCaseList() {
              return this.caseList;
       }

       public String getDescription() {
              return this.description;
       }
}
```

16. Файл CaseXES.java пакета model.xes.

```
package model.xes;

import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.annotation.*;

@XmlType(propOrder = { "eventList" })
public class CaseXES {
```

```
    private List<EventXES> eventList = new ArrayList<>();

    @XmlElement(name = "event")
    public List<EventXES> getEventList() {
        return this.eventList;
    }
}
```

17. Файл EventXES.java пакета model.xes.

```
package model.xes;

import java.util.List;
import java.util.ArrayList;
import javax.xml.bind.annotation.*;

@XmlType(propOrder = { "stringList", "date" })
public class EventXES {

    @XmlAccessorType(XmlAccessType.FIELD)
    @XmlType(name = "date", propOrder = { "key", "value" })
    public static class DateXES {

        @XmlAttribute(name = "key")
        private String key;

        @XmlAttribute(name = "value")
        private String value;

        public String getKey() {
            return this.key;
        }

        public String getValue() {
            return this.value;
        }
    }

    @XmlAccessorType(XmlAccessType.FIELD)
    @XmlType(name = "string", propOrder = { "key", "value" })
    public static class StringXES {

        @XmlAttribute(name = "key")
        private String key;

        @XmlAttribute(name = "value")
        private String value;

        public String getKey() {
            return this.key;
        }

        public String getValue() {
            return this.value;
        }
    }
```

```
    private List<StringXES> stringList = new ArrayList<>();
    @XmlElement(name = "date")
    private DateXES date;

    @XmlElement(name = "string")
    public List<StringXES> getStringList() {
        return this.stringList;
    }

    public DateXES getDate() {
        return this.date;
    }
}
```

18. Файл LogXES.java пакета model.xes.

```
package model.xes;

import java.io.File;
import java.util.ArrayList;
import java.util.List;
import javax.xml.bind.*;
import javax.xml.bind.annotation.*;

@XmlRootElement(name = "log", namespace="http://www.xes-standard.org/")
@XmlType(propOrder = { "caseList" })
public class LogXES {

    private List<CaseXES> caseList = new ArrayList<>();

    @XmlElement(name = "trace")
    public List<CaseXES> getCaseList() {
        return this.caseList;
    }

    public static LogXES extractLogXES(String path) {
        try {

            JAXBContext context = JAXBContext.newInstance(LogXES.class);
            Unmarshaller unmar = context.createUnmarshaller();
            LogXES log = (LogXES) unmar.unmarshal(new File(path));
            return log;
        }

        catch (JAXBException e) {
            e.printStackTrace();
        }

        return null;
    }
}
```

19. Файл GraphManager.java пакета extractor.

```
package extractor;
```

```
import java.io.*;
import java.util.HashMap;
import java.util.Map;
import org.jgrapht.Graph;
import org.jgrapht.graph.DefaultEdge;
import org.jgrapht.io.*;
import model.ActivityMap;
import model.TransSystem;


public class GraphManager {

    private static final double maxPenWidth = 7.0;
    private static final Attribute nodeStyle =
            DefaultAttribute.createAttribute("rounded, filled");
    private static final Attribute nodeColor1 =
            DefaultAttribute.createAttribute("#6699ff");
    private static final Attribute nodeColor2 =
            DefaultAttribute.createAttribute("#66ff66");
    private static final Attribute nodeShape =
            DefaultAttribute.createAttribute("box");


    public static void graphToDot(Graph<String, DefaultEdge> graph,
            String path) {

        DOTExporter<String, DefaultEdge> exporter =
            new DOTExporter<>(
                new IntegerComponentNameProvider<String>(),
                new StringComponentNameProvider<String>(),
                new StringComponentNameProvider<DefaultEdge>());

        try (FileWriter fw = new FileWriter(new File(path))) {

            exporter.exportGraph(graph, fw);
            fw.flush();
        }

        catch (IOException e) {
            e.printStackTrace();
        }
    }

      public static void transSystemToDot(TransSystem ts, String path) {

        DOTExporter<TransSystem.TSVertex, TransSystem.TSEdge> exporter =
            new DOTExporter<>(
                new IntegerComponentNameProvider<TransSystem.TSVertex>(),
                new StringComponentNameProvider<TransSystem.TSVertex>(),
                new ComponentNameProvider<TransSystem.TSEdge>() {
                    @Override
                    public String getName(TransSystem.TSEdge edge) {
                        return "  " + edge.getLabel() + " (" +
                                (int) edge.getWeight() + ")  ";
                    }
                },
                new ComponentAttributeProvider<TransSystem.TSVertex>() {
                    @Override
                    public Map<String, Attribute> getComponentAttributes(
                            TransSystem.TSVertex vertex) {
```

```
                    Map<String, Attribute> attributes = new HashMap<>();
                    attributes.put("style", nodeStyle);
                    attributes.put("fillcolor", nodeColor1);
                    attributes.put("shape", nodeShape);
                    return attributes;
                }
            }, new ComponentAttributeProvider<TransSystem.TSEdge>() {
                @Override
                public Map<String, Attribute> getComponentAttributes(
                        TransSystem.TSEdge edge) {

                    Map<String, Attribute> attributes = new HashMap<>();
                    double edgeWidth = 1 + (maxPenWidth - 1) *
                            edge.getWeight() / ts.getMaxEdgeWeight();
                    attributes.put("penwidth", DefaultAttribute.
                            createAttribute(String.format("%.2f",
                                    edgeWidth)));
                    return attributes;
                }
            });

    File file = new File(path);
    FileWriter fw = null;
    try {

        fw = new FileWriter(file);
        exporter.exportGraph(ts, fw);
        fw.flush();
    }

    catch (IOException e) {
        e.printStackTrace();
    }

    finally {
        try {
            fw.close();
        }

        catch (IOException e) {
            e.printStackTrace();
        }
    }
}

   public static void activityMapToDot(ActivityMap am, String path) {

    DOTExporter<ActivityMap.AMVertex, DefaultEdge> exporter =
            new DOTExporter<>(
                new IntegerComponentNameProvider<ActivityMap.AMVertex>(),
                new ComponentNameProvider<ActivityMap.AMVertex>() {
                    @Override
                    public String getName(ActivityMap.AMVertex vertex) {
                        return " " + vertex.getActivity() + " ";
                    }
                },
                new ComponentNameProvider<DefaultEdge>() {
                    @Override
                    public String getName(DefaultEdge edge) {
                        return "";
```

```
                        }
                    },
                    new ComponentAttributeProvider<ActivityMap.AMVertex>() {
                        @Override
                        public Map<String, Attribute> getComponentAttributes(
                                ActivityMap.AMVertex vertex) {

                            Map<String, Attribute> attributes = new HashMap<>();
                            attributes.put("style", nodeStyle);
                            attributes.put("fillcolor", nodeColor2);
                            attributes.put("shape", nodeShape);
                            return attributes;
                        }
                    },
                    new ComponentAttributeProvider<DefaultEdge>() {
                        @Override
                        public Map<String, Attribute> getComponentAttributes(
                                DefaultEdge edge) {

                            Map<String, Attribute> attributes = new HashMap<>();
                            return attributes;
                        }
                    });

        File file = new File(path);
        FileWriter fw = null;
        try {

            fw = new FileWriter(file);
            exporter.exportGraph(am, fw);
            fw.flush();
        }

        catch (IOException e) {
            e.printStackTrace();
        }

        finally {
            try {
                fw.close();
            }

            catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

20. Файл InterfaceWindow.java пакета extractor.

```
package extractor;

import java.awt.GridLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;
```

```
import javax.swing.filechooser.FileNameExtensionFilter;
import java.io.*;
import java.util.concurrent.TimeUnit;
import java.util.List;
import model.*;
import model.log4j.*;
import model.mxml.*;
import model.xes.*;

public class InterfaceWindow extends JFrame {

    private static final long serialVersionUID = 3236307626298226499L;
    private String workingDirectory = System.getProperty("user.dir");
    private VisualMode vMode = VisualMode.TRANSITION_SYSTEM;
    private long dotTimeoutMillis = 60000;

    private Canonical associatedModel = null;
    private List<ReferencedSequence> refSeqList = null;
    private TransSystem transSystem = null;
    private List<ActivityMap> actMapAtlas = null;

    private static enum VisualMode {
        TRANSITION_SYSTEM,
        ACTIVITY_MAP
    }

    private class LogStream extends OutputStream {
        private JTextArea logTextArea;

        public LogStream(JTextArea logTextArea) {
            this.logTextArea = logTextArea;
        }

        @Override
        public void write(int b) throws IOException {

            logTextArea.append(String.valueOf((char)b));
            logTextArea.setCaretPosition(logTextArea.getDocument().getLength());
        }
    }

    public InterfaceWindow() {

        super("Test model extraction");
        final InterfaceWindow thisInterface = this;

        final JTabbedPane tabbedPane = new JTabbedPane();
        final JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y_AXIS));

        final JPanel controlPanel = new JPanel(new GridLayout(5, 2, 10, 8));
        final JLabel fileChooseLabel = new JLabel("Specify event log file ");
        controlPanel.add(fileChooseLabel);
        final JButton buttonOpenFile = new JButton("Open file...");

        controlPanel.add(buttonOpenFile);
        final JLabel refineLabel = new JLabel("Apply model refinement?");
        controlPanel.add(refineLabel);
        final JButton refineButton = new JButton("Refine");

        controlPanel.add(refineButton);
```

```
final JLabel generalizeLabel = new JLabel("Generalize model?");
controlPanel.add(generalizeLabel);
final JButton generalizeButton = new JButton("Generalize");
generalizeButton.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent actionEvent) {

        long startTime = System.nanoTime();
        thisInterface.vMode = VisualMode.ACTIVITY_MAP;
        thisInterface.refSeqList =
                ReferencedSequence.generalizeModel(associatedModel);

        System.out.println("--former cases number: " +
                associatedModel.getTaggedSequences().size());
        System.out.println("--generalized cases number: " +
                refSeqList.size());
        System.out.println("successfully generalized model; " +
                "time taken: " + (System.nanoTime()-startTime)/1000000 +
                "  ms");
    }
});

controlPanel.add(generalizeButton);

final JLabel exportPdfLabel = new JLabel("Export to PDF?");
controlPanel.add(exportPdfLabel);
final JButton exportPdfButton = new JButton("Export");
exportPdfButton.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent actionEvent) {

        System.out.println("translating to PDF...");
        String outputBasePath = workingDirectory + File.separator;
        int artifactNumber = 0;
        long startTime = System.nanoTime();

        if (thisInterface.vMode == VisualMode.TRANSITION_SYSTEM) {

            artifactNumber = 1;
            transSystem = new TransSystem();
            transSystem.emulateCanonical(associatedModel);
            GraphManager.transSystemToDot(transSystem,
                    outputBasePath + "result0.dot");
        }

        else {

            List<ActivityMap> actMaps = ActivityMap.
                emulateReferencedSequences(thisInterface.refSeqList);
            artifactNumber = actMaps.size();
            thisInterface.actMapAtlas = actMaps;

            for (int i = 0; i < artifactNumber; ++i) {
                GraphManager.activityMapToDot(actMaps.get(i),
                        outputBasePath + "result" + i + ".dot");
            }
        }

        System.out.println("exported model to DOT; " +
```

```
                "time taken: " + (System.nanoTime()-startTime)/1000000 +
                "  ms");

        try {
            for (int i = 0; i < artifactNumber; ++i) {

                System.out.println("PDFying " + i + "th artifact...");
                Process dotProgram = new ProcessBuilder(
                        "dot.exe", "-Tpdf", "-O",
                        outputBasePath + "result" + i + ".dot").start();
                boolean waitStatus = dotProgram.waitFor(
                        dotTimeoutMillis, TimeUnit.MILLISECONDS);

                if (waitStatus) {
                    System.out.println(i + "th artifact is translated");
                }

                else {
                    System.out.println("dot to pdf translation " +
                            "timeout exceeded; aborting");
                    dotProgram.destroy();
                }
            }

            System.out.println("translation is done");
        }

        catch (IOException e) {
            e.printStackTrace();
        }

        catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
});

controlPanel.add(exportPdfButton);

final JLabel genTestsLabel = new JLabel("Generate test cases?");
controlPanel.add(genTestsLabel);
final JButton genTestsButton = new JButton("Generate");
genTestsButton.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent actionEvent) {

        System.out.println("preparing for test case generating...");
        long startTime = System.nanoTime();

        if (thisInterface.vMode == VisualMode.TRANSITION_SYSTEM) {

            thisInterface.transSystem.exportTestData(workingDirectory +
                    File.separator + "testcases0.xml");
            System.out.println("testcases0.xml file is ready");
        }

        else {
            for (int i = 0; i < thisInterface.actMapAtlas.size(); ++i) {
                actMapAtlas.get(i).exportTestData(workingDirectory +
                    File.separator + "testcases" + i + ".xml");
```

```
                System.out.println("testcases" + i +
                        ".xml file is ready");
            }
        }

        System.out.println("time taken: " +
                (System.nanoTime()-startTime)/1000000 + "  ms");
    }
});

controlPanel.add(genTestsButton);
final JTextArea logTextArea =
        new JTextArea("===========< PROGRAM LOG >===========", 40, 80);
logTextArea.setEditable(false);
PrintStream printStream =
        new PrintStream(new LogStream(logTextArea));

// redirecting standard streams to log area stream
System.setErr(printStream);
System.setOut(printStream);
final JScrollPane logPane = new JScrollPane(logTextArea);
System.out.println();

mainPanel.add(controlPanel);
mainPanel.add(Box.createRigidArea(new Dimension(0, 20)));
mainPanel.add(logPane);
final JPanel settingsGeneralPanel =
        new JPanel(new GridLayout(6, 1, 0, 10));

final JLabel workingDirLabel = new JLabel("Working directory");
settingsGeneralPanel.add(workingDirLabel);
final JPanel workingDirPanel = new JPanel(new GridLayout(1, 2, 10, 0));
final JTextField workingDirField =
        new JTextField(this.workingDirectory);
workingDirField.setEditable(false);
workingDirPanel.add(workingDirField);
final JButton changeDirButton = new JButton("Change...");
changeDirButton.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent actionEvent) {

        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);

        int eCode = fileChooser.showOpenDialog(settingsGeneralPanel);
        if (eCode != JFileChooser.APPROVE_OPTION) {

            System.out.println("no directory's been chosen; aborting");
            return;
        }

        thisInterface.workingDirectory =
                fileChooser.getSelectedFile().getAbsolutePath();
        System.out.println("setting " +
                thisInterface.workingDirectory + " as a working dir");
        workingDirField.setText(workingDirectory);
    }
});
workingDirPanel.add(changeDirButton);
settingsGeneralPanel.add(workingDirPanel);
```

```
final JLabel log4jMappingLabel = new JLabel("Mapping for log4j logs");
settingsGeneralPanel.add(log4jMappingLabel);
final JTextField log4jMappingField =
        new JTextField("%d %t %C %M %pid %m");
settingsGeneralPanel.add(log4jMappingField);

final JLabel timestampFormatLabel =
        new JLabel("Timestamp pattern (leave empty for default)");
settingsGeneralPanel.add(timestampFormatLabel);
final JTextField timestampFormatField =
        new JTextField();
settingsGeneralPanel.add(timestampFormatField);
final JPanel settingsFiltrationPanel =
        new JPanel(new GridLayout(4, 2, 10, 10));
final JLabel prefixFreqThresholdLabel =
        new JLabel("Prefix frequency threshold (%) ");
settingsFiltrationPanel.add(prefixFreqThresholdLabel);
final JTextField prefixFreqThresholdField = new JTextField("15");
settingsFiltrationPanel.add(prefixFreqThresholdField);

final JLabel postfixFreqThresholdLabel =
        new JLabel("Postfix frequency threshold (%) ");
settingsFiltrationPanel.add(postfixFreqThresholdLabel);
final JTextField postfixFreqThresholdField = new JTextField("10");
settingsFiltrationPanel.add(postfixFreqThresholdField);

final JLabel missedEventFreqThresholdLabel =
        new JLabel("Missed event frequency threshold (%) ");
settingsFiltrationPanel.add(missedEventFreqThresholdLabel);
final JTextField missedEventFreqThresholdField = new JTextField("15");
settingsFiltrationPanel.add(missedEventFreqThresholdField);

final JLabel timeDeviationLabel =
        new JLabel("Acceptable missed event time deviation (ms)");
settingsFiltrationPanel.add(timeDeviationLabel);
final JTextField timeDeviationField = new JTextField("512");
settingsFiltrationPanel.add(timeDeviationField);

tabbedPane.addTab("Main", mainPanel);
tabbedPane.addTab("Settings (general)", settingsGeneralPanel);
tabbedPane.addTab("Settings (filtration)", settingsFiltrationPanel);
this.add(tabbedPane);

buttonOpenFile.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent actionEvent) {

        JFileChooser fileChooser = new JFileChooser();
        FileNameExtensionFilter filter =
            new FileNameExtensionFilter("*.xes, *.mxml, *.log",
                    "xes", "mxml", "log");
        fileChooser.setFileFilter(filter);

        int eCode = fileChooser.showOpenDialog(controlPanel);
        if (eCode != JFileChooser.APPROVE_OPTION) {

            System.out.println("no file has been chosen; aborting");
            return;
        }
```

```
        long startTime = System.nanoTime();
        String path = fileChooser.getSelectedFile().getAbsolutePath();
        System.out.println("opening file " + path + "...");
        String extension = path.substring(
                path.lastIndexOf(".")+1, path.length());
        System.out.println("detected ." + extension + " extension");
        switch (extension) {

            case "log":

                LogLog4j log4j = LogLog4j.extractLogLog4j(path);
                associatedModel = Translator.castLog4j(log4j);
                break;

            case "mxml":

                LogMXML logMXML = LogMXML.extractLogMXML(path);
                associatedModel = Translator.castMXML(logMXML);
                break;

            case "xes":

                LogXES logXES = LogXES.extractLogXES(path);
                associatedModel = Translator.castXES(logXES);
                break;

            default:
                System.out.println("error: unknown file format");
                return;
        }

        System.out.println("successfully extracted initial model; " +
            "time taken: " + (System.nanoTime()-startTime)/1000000
            + " ms");

        // renewing model data
        thisInterface.vMode = VisualMode.TRANSITION_SYSTEM;
        thisInterface.refSeqList = null;
        thisInterface.transSystem = null;
        thisInterface.actMapAtlas = null;
    }
});

refineButton.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent actionEvent) {

        associatedModel.setDeltaT(Long.parseLong(
            timeDeviationField.getText()));
        associatedModel.setPrefixFrequencyThreshold(
            Double.parseDouble(prefixFreqThresholdField.getText())/100);
        associatedModel.setPostfixFrequencyThreshold(
            Double.parseDouble(
                postfixFreqThresholdField.getText())/100);
        associatedModel.setMissedFrequencyThreshold(
            Double.parseDouble(
                missedEventFreqThresholdField.getText())/100);

        long startTime = System.nanoTime();
```

```
                associatedModel.refineData();
                System.out.println("successfully refined model; " +
                        "time taken: " + (System.nanoTime()-startTime)/1000000
                        + " ms");
            }
        });

        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setPreferredSize(new Dimension(500, 600));
        this.pack();
        this.setLocationRelativeTo(null);
        this.setVisible(true);
    }
}
```

21. Файл Main.java пакета extractor (точка входа в приложение).

```
package extractor;

public class Main {

    public static void main(String[] args) {

        InterfaceWindow iWindow = new InterfaceWindow();
        System.out.println("initialized gui: " + iWindow);
      }
}
```