Модел. и анализ информ. систем. Т. **21**, № **5** (2014) 102–115

УДК 519.682.6+004.43

Графический язык DPMine для автоматизации экспериментов в области автоматического построения и анализа процессов

Шершаков С. А.[1]

Национальный исследовательский университет «Высшая школа экономики»
101000 Россия, г. Москва, ул. Мясницкая, 20

e-mail: sshershakov@hse.ru

получена 1 октября 2014

Ключевые слова: извлечение и анализ процессов, язык моделирования, автоматизация, эксперименты, нечеткая модель

Извлечение процессов (process mining) — это новое направление в области моделирования и анализа процессов, в котором важную роль играет использование информации из журналов (логов) событий, хранящих историю поведения системы. Методы и подходы, используемые при извлечении процессов, часто опираются на различные эвристики, и эксперименты с большими логами событий важны для обоснования и сравнения разрабатываемых методов и алгоритмов. Такие эксперименты весьма трудоемки, поэтому их автоматизация является актуальной задачей в области извлечения процессов. В данной работе представлен язык DPMine, разработанный специально для описания и проведения экспериментов по извлечению и анализу моделей процессов. Дается описание основных концепций языка, а также принципов и механизмов его расширения. Рассматриваются вопросы интеграции языка в инструмент моделирования VTMine в виде динамически загружаемых компонентов. Приводится пример эксперимента по построению нечеткой модели процесса по логу данных, хранящемуся в виде нормализованной базы данных.

Введение

Данная статья посвящена моделированию экспериментов в области процессно-ориентированных информационных систем, стремительное развитие которых связано с изменяющимися подходами к управлению бизнес- и технологическими процессами.

Процессы в информационных системах становятся все более сложными, количество связанной с ними информации неуклонно растет. Это приводит к бурному развитию исследований, связанных с обработкой «больших данных» (Big Data) [1]. В последнее время появились новые специальности в области исследования данных

[1]Работа выполнена в рамках Программы фундаментальных исследований НИУ ВШЭ в 2014 году.

и процессов: аналитик данных (data analyst), исследователь в области процессов (process researcher), инженер по процессам (process engineer) [2] и др.

Новое направление в исследованиях процессов получило в англоязычной литературе название *Process Mining* — «извлечение (автоматическое построение) и анализ процессов», «майнинг процессов» [3]. В рамках этого направления в качестве отправной точки для изучения процессов предлагается использовать *журналы (логи) событий* (event logs), которые автоматически генерируются большинством процессно-ориентированных информационных систем. *Лог событий* — это след, который оставляет информационная система в ходе своей работы. Он хранит информацию о состояниях системы в разные периоды ее работы. Как правило, такой лог представляет собой набор последовательностей *событий* (event). Отдельная последовательность событий называется *трассой* (trace), трасса представляет историю исполнения процесса в одном отдельно взятом *случае* (case). Каждое событие в трассе является отметкой о выполнении так называемой *активности* (activity) — отдельной задачи или этапа рассматриваемого процесса.

Извлечение процессов тесно связано с *добычей данных* (data mining), *машинным обучением* (machine learning), моделированием и анализом моделей процессов. Основные задачи и цели этого направления исследований изложены в Манифесте (Process Mining Manifesto) [4] и могут быть кратко представлены тремя ключевыми проблемами: 1) извлечение модели из лога данных (process discovery), 2) проверка соответствия некоторой модели реальным данным (conformance checking) и 3) улучшение и исправление модели с учетом изменяющихся данных (model enhancement).

Несмотря на то, что извлечение процессов является относительно новым направлением, оно уже активно применяется при моделировании и анализе бизнес-процессов [5] в менеджменте, при разработке программного обеспечения [6, 7], в управлении технологическими процессами, в медицине.

В основе теории извлечения процессов лежат формальные модели и алгоритмы, однако при исследовании конкретных предметных областей обычно используется большое количество эмпирических предположений и допущений. Этим определяется предметно-ориентированный, прикладной характер таких исследований, требующий проведения значительного числа *экспериментов* для оценки применимости тех или иных методов. Эксперименты могут проводиться не только с использованием логов реальных систем, но и с логами, синтезированными по моделям процессов, представленных, например, в виде сетей Петри [8, 9].

Для описания моделей процессов используются различные средства — языковые нотации (workflow notations) и программные инструменты (workflow engines/systems). В настоящий момент известно значительное число нотаций для описания процессов, потоков задач (в основе большинства из них лежат сети Петри) и систем, реализующих эти нотации в инструменты для конечного пользователя. В качестве примера можно привести нотацию для описания бизнес-процессов BPMN [10], язык исполнения бизнес-процессов BPEL [11], язык моделирования процессов YAWL [12].

Для описания *эксперимента* в области извлечения процессов, как и для описания других процессов, необходимо использовать определенную нотацию для его записи. Такой записью может быть, например, *программа* или *модель эксперимента*. Запись такой модели может осуществляться при помощи одного из вышеперечисленных языков, однако такой подход влечет за собой ряд недостатков. Они связаны,

в первую очередь, с тем, что часть из имеющихся языков моделирования ориентирована на специализированную предметную область, — например, на управление бизнес-процессами или оркестрацию веб-сервисов. Другие, более универсальные, не всегда позволяют описать модель эксперимента лаконичным и удобным для пользователя образом.

Для решения задач автоматизации экспериментов в области извлечения процессов был предложен подход на основе графического расширяемого языка моделирования DPMine [13]. Требования, продиктовавшие разработку нового языка и сопутствующих инструментов его реализации, включали наличие простой, прозрачной, гибкой и, главное, расширяемой семантики. Язык DPMine разрабатывался изначально под эту конкретную задачу и получил свое воплощение в виде набора плагинов [14] для программной платформы ProM [15] — одного из самых функционально широких инструментов для Process Mining.

Несмотря на свое основное назначение, область применения языка DPMine не ограничивается экспериментами по извлечению и анализу процессов. За счет расширяемой блоковой системы DPMine может быть использован для автоматизации разных программно реализуемых процессов. В отличие от похожего, на первый взгляд, по назначению языка YAWL [12], в языке DPMine не декларируется обобщенный набор блоков/задач для поддержки управляющих потоком конструкций (например таких, как XOR/AND split/join) как неотъемлемая составляющая языка. Вместо этого в языке DPMine поддерживается гибкая система расширения типов блоков, привносящих в язык любую необходимую семантику, в том числе стандартные управляющие конструкции.

В настоящей работе рассматриваются ключевые особенности языка DPMine, реализованного в виде библиотеки DPMine/C на языке C++ [16], интегрированной в универсальную расширяемую систему моделирования VTMine [17], которая в свою очередь является для него *целевой платформой*. Обсуждаются ключевые решения для реализации расширяемой структуры обоих инструментов, работающих в тесном взаимодействии друг с другом. Рассматривается пример эксперимента на языке DPMine для решения задачи автоматического построения модели процесса в виде *нечеткой диаграммы* (fuzzy map) исходя из лога событий, представленного в виде нормализованной базы данных.

Остальная часть работы организована следующим образом. Раздел 1 посвящен основным концепциям, компонентам и принципам реализации языка DPMine. В разделе 2 обсуждаются подходы к интеграции библиотеки DPMine/C в инструмент для моделирования VTMine. Пример создания модели эксперимента в области Process Mining рассматривается в разделе 3. Наконец, в разделе 4 подводится итог проделанной работе и формулируются задачи для дальнейшего исследования.

1. Основные компоненты языка

DPMine является графическим языком описания процессов, экспериментов, состоящих из отдельных компонентов — блоков, система типов которых является расширяемой, что позволяет использовать его для описания процессов в разных предметных областях [13, 14]. В настоящем разделе рассматриваются основные концептуальные элементы языка DPMine.

1.1. Модели, схемы, блоки, порты и коннекторы

Основным рабочим документом языка является *модель*. Модель описывает некоторый процесс в виде *потоков задач* (task workflow), конечным результатом которого является исполнение входящих в модель задач — всех или некоторых.

Модель рассматривается на разных уровнях абстракции (рис. 1а). *Объектная модель* (object model) находится на низком уровне и представляет структуру взаимодействующих объектов в оперативной памяти. *Графическая модель* (graphical model) имеет визуальное представление в виде редактируемой графической схемы. *Хранимая модель* (persistent model) используется для сериализации модели, например в виде файла.

Модель включает *главную схему* и атрибуты, такие как название, версия и т. д. Модель исполняется *базовым инструментом*, роль которого играет приложение, клиентом которого является библиотека DPMine/C. Задачи, описываемые моделью, определяются назначением базового инструмента. Примером таких задач является создание, преобразование, визуализация и сериализация формальных моделей.

Семантика языка реализуется с помощью концепции *блоков*, *портов*, *коннекторов* и *схем*. Любое расширение языка осуществляется посредством только этих четырех компонентов.

Блок является основным элементом языка и рассматривается как элементарная операция. Блоки, в зависимости от своего типа, реализуют задачи базового инструмента (например путем вызова определенного метода или обращения к плагину), используются для иерархического представления сложных схем в виде единого блока, реализуют конструкции управления потоком исполнения, используются как оператор подстановки для передачи некоторой схемы в другую схему в виде параметра и др. По выполняемым задачам блоки объединяются в иерархию типов.

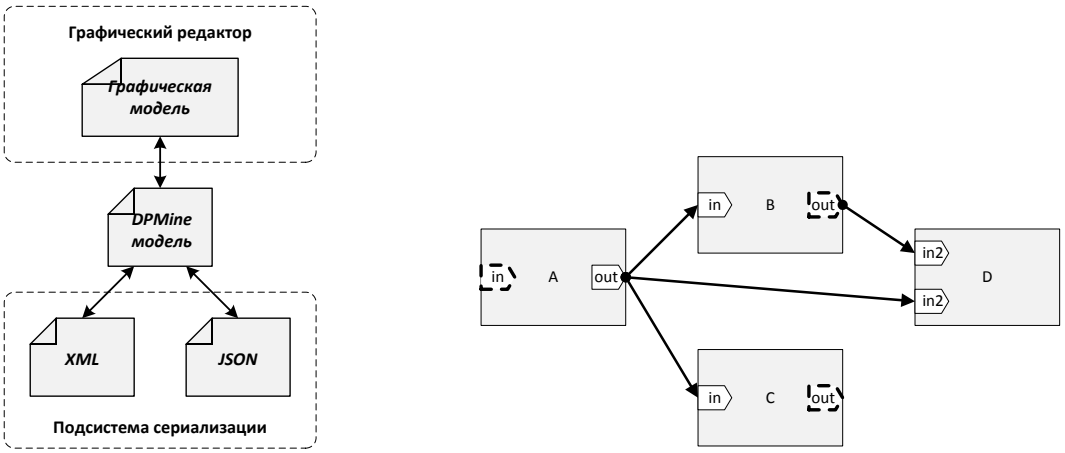
Порт — объект связи, принадлежащий блоку, обладающий характеристиками направления (входные, выходные и прокси-порты) и типа данных. Порты используются для транспортировки объектов (ресурсов) заданного типа в блок и из него. В зависимости от типа блока подразделяются на пользовательские (custom) и встроенные (built-in).

Коннектор — направленный объект связи, соединяющий два блока через их порты: выходной порт одного блока (своим началом) с входным портом другого (своим концом). Из одного выходного порта может выходить несколько коннекторов, в один входной порт входит всегда только один коннектор (рис. 1б).

Схема — множество взаимодействующих блоков, связанных между собой коннекторами. Схема является основным способом реализации абстрагирования, изолирования и иерархии подпроцессов. *Интерфейс схемы* — произвольное подмножество Ifp портов множества всех портов, образованного блоками, входящими в состав этой схемы. На рис. 1б интерфейсом схемы являются порты `Ifp = {A.in, B.out, C.out}` (выделены пунктирной обводкой).

На уровне объектной модели схема упаковывается в контейнер, представленный блоком специального типа *схема*. Он наследует характеристики базового типа — *телосодержащего* блока (*bodyness block*), инкапсулирующего в себе другие блоки (тело), для которых он является родительским блоком, и являющегося базовым помимо схемы для таких типов, как циклы и др.

Телосодержащий блок может рассматриваться на двух уровнях. На *внутреннем*



(а) Уровни представления модели (б) Пример схемы

Рис. 1. Модель и схема

уровне — это изолированная схема, *интерфейс* которой подключается коннекторами к прокси-портам содержащего ее блока, через которые осуществляется обмен данными с внешними блоками. На *внешнем уровне* он является регулярным блоком, содержащим входные и выходные порты (являющиеся одновременно и прокси-портами внутреннего уровня) и обменивающимся данными с находящимися на том же иерархическом уровне (сестринскими) блоками. Такой подход позволяет рассматривать схему в виде единственного блока, абстрагируясь от его содержимого.

1.2. Исполнение модели, схемы, блока

Модель DPMine является *исполняемой*. *Исполнение модели* (model execution) осуществляется путем исполнения *главной схемы*, которое определяется составом входящих в нее блоков и структурой их соединения. В результате исполнения модели, блоки, входящие в нее, создают новые и преобразуют существующие ресурсы, выполняют подготовительные операции для их визуализации и сохранения и др.

Исполнение модели осуществляется специальным агентом — *исполнителем*, связывающим систему моделирования DPMine с *базовым инструментом*.

Исполнение главной схемы модели осуществляется путем исполнения ее блока-контейнера. *Исполнение блока* — операция, выполняемая программным обеспечением, ассоциированным с *типом*, определяющим каждый блок языка DPMine. Примером такой операции является алгоритм, получающий в качестве параметров ресурсы с входных портов блока, создающий на их основе новые ресурсы и помещающий их на выходные порты этого блока (например преобразование сети Петри в остовный граф этой сети).

Условием исполнения блока является удовлетворение всех его внешних зависимостей. Для блока B его зависимости считаются удовлетворенными, если: 1) блок B не имеет входных портов; 2) блок имеет входные порты и для каждого такого порта выполняются следующие требования: 2а) для порта не установлен флаг «обязательно подключен», тогда порт может быть не подключен коннектором к другому (выходному) порту другого блока; 2б) порт подключен коннектором к другому (вы-

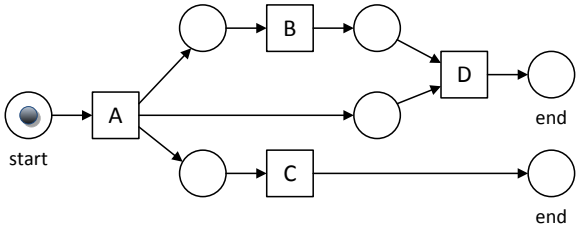


Рис. 2. Модель сети Петри для исполнения схемы на рис. 1б

ходному) порту другого блока, и статус такого блока — «исполнен», что требует наличия на его выходных портах данных, имплементирующих заявленные типом блока интерфейсы выходных ресурсов.

Блок с удовлетворенными внешними зависимостями имеет статус *исполнимый*. Если блок является *исполнимым*, тогда агентом-исполнителем вызывается процедура исполнения этого блока, а сам блок получает статус *исполняемый*. Блок, попытка исполнения которого завершена успешно, получает статус *исполненный*.

Алгоритм исполнения телосодержащего блока является итеративным. На каждой итерации агент-исполнитель осуществляет попытку исполнить подмножество *исполнимых* блоков множества всех блоков, входящих в состав схемы телосодержащего блока. В процессе работы алгоритм определяет и использует несколько специальных *флагов состояния* для полной схемы. Так, флаг *incomplete* показывает, что после очередной итерации все еще остаются *неисполненные* блоки. Флаг *hasExecution* указывает, что за очередную итерацию по крайней мере один блок был исполнен. Наконец, флаг *hasPending* отображает состояние некоторых блоков, все еще исполняющихся к моменту окончания очередной итерации. Такие блоки называются *ожидаемыми*.

В начале работы алгоритма все три флага сбрасываются. Алгоритм осуществляет попытку исполнения каждого блока, входящего в схему. Если *обозреваемый блок* является *исполненным*, алгоритм пропускает его и переходит к следующему блоку. Если *обозреваемый блок* является *ожидаемым*, выставляется флаг *hasPending*, и алгоритм переходит к следующему блоку. Если *обозреваемый блок* должен быть выполнен в эту итерацию, осуществляется проверка, удовлетворены ли его входные зависимости, и в случае успеха он передается для диспетчеризации исполнения агенту-исполнителю. Если по окончании очередной итерации остается хотя бы один блок со статусом, отличным от *исполненного* (то есть блок либо не был начат исполняться, либо находится в состоянии ожидаемого исполнения), выставляется флаг *incomplete*. В последнем случае, если за прошедшую итерацию хотя бы один блок был поставлен на исполнение, алгоритм осуществляет еще одну итерацию.

Процесс исполнения схемы может быть представлен моделью в форме сети Петри. Так, для схемы (рис. 1б), не содержащей блоков выбора, эквивалентная сеть Петри может иметь вид, представленный на рис. 2.

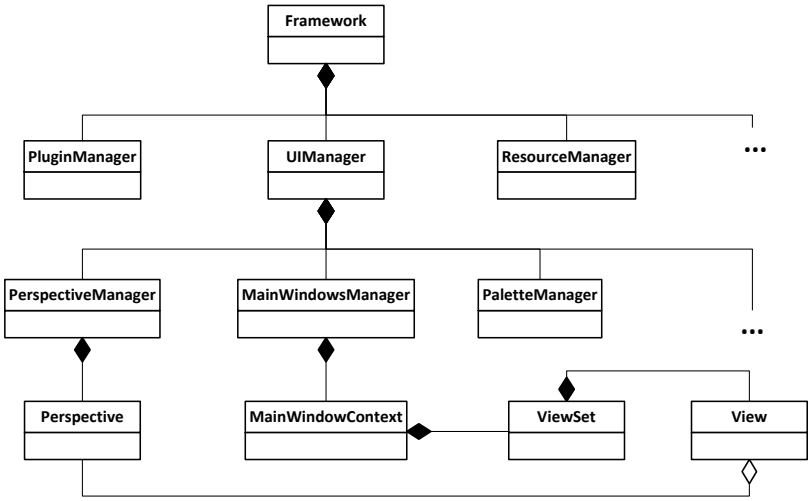


Рис. 3. Основные компоненты ядра VTMine

2. Интеграция языка DPMine в систему моделирования VTMine

VTMine — графический инструмент моделирования, функциональные возможности которого расширяются за счет динамически загружаемых компонентов — *плагинов* (plug-ins) [17].

Ядро VTMine определяет минимально необходимые для загрузки плагинов алгоритмы и декларирует интерфейсы (в том числе некоторые компоненты пользовательского интерфейса), все остальные возможности полностью определяются набором плагинов. В качестве основных моделей рассматриваются формализмы, наиболее часто используемые в тематике Process Mining: графовые структуры, системы переходов, конечные автоматы, сети Петри и др. При этом ядро системы не имеет специальной поддержки каких-то отдельных классов моделей, поэтому VTMine может использоваться для моделирования в разных предметных областях.

VTMine написан на языке C++, что, с одной стороны, позволяет гибко управлять распределением памяти, а с другой — разрабатывать быстрые и эффективные алгоритмы, что особенно важно при обработке больших данных (big data). Приложение построено на основе кроссплатформенной библиотеки Qt, из основных подсистем которой активно используются GUI для построения пользовательского интерфейса, объектно-ориентированная подсистема загрузки плагинов, инкапсулирующая взаимодействие с исходными форматами динамически загружаемых библиотек каждой операционной системы в отдельности и высокопроизводительная подсистема отрисовки графических схем Graphics View Framework.

2.1. Структура ядра приложения

Ядро приложения представляет собой иерархическую систему базовых компонентов — *менеджеров* (рис. 3). Такая иерархическая структура позволяет производить модификацию отдельных компонентов как ядра, так и плагинов без потери сов-

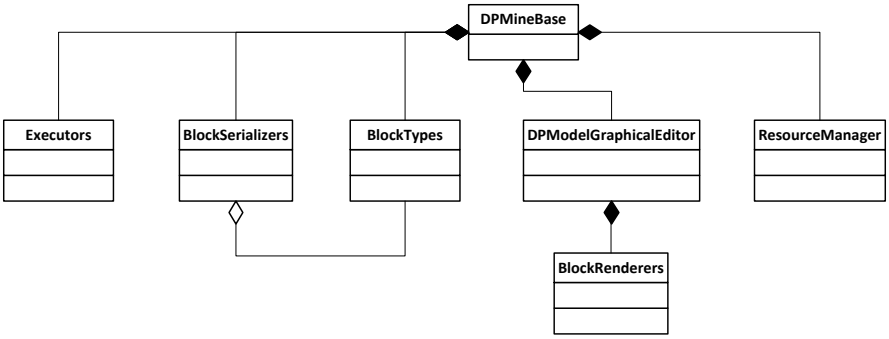


Рис. 4. Основные компоненты плагина `DPMineBase`

местимости между различными версиями на этапе позднего связывания. Любой компонент системы может быть заменен другим, реализующим тот же интерфейс.

Основным компонентом приложения является корневой объект `Framework`, представляющий *каркас* приложения. Этот объект содержит компоненты и менеджеры верхнего уровня, такие как логгер, менеджер плагинов (`PluginManager`), менеджер компонентов пользовательского интерфейса (`UIManager`), менеджер ресурсов `ResourceManager` и др. Каркас `Framework` доступен в большинстве модулей и передается в качестве параметра в точку входа каждого плагина, что дает возможность таким плагинам модифицировать компоненты верхнего уровня.

Подсистема управления плагинами управляет жизненным циклом плагинов VTMine, включающим процесс идентификации файловых контейнеров плагинов, загрузку плагинов с учетом зависимостей друг от друга, регистрацию плагинов в системе, модификацию плагинами других плагинов и выгрузку плагинов.

В VTMine учитывается зависимость плагинов друг от друга. Это означает, что если плагин P_A зависит от плагина P_B (где оба плагина имеют уникальные строковые идентификаторы), то плагин P_A может быть загружен только в том случае, если загружен плагин P_B. Как следствие, если плагин P_B по любой причине не может быть загружен, плагин P_A также не будет загружен. Частным случаем является ситуация с циркулярной зависимостью двух плагинов друг от друга, при которой ни один из них не загружается.

2.2. Плагины DPMine

Система моделирования DPMine подключается к приложению VTMine в виде набора плагинов, основным из которых является плагин `DPMineBase`. Структура основных компонентов плагина представлена на рис. 4.

Менеджер ресурсов: 1) выполняет регистрацию *типов ресурсов* языка DPMine, такие как *DPMine модель*; 2) регистрирует новые обработчики («doers») для некоторых типов ресурсов и реализует функционал этих обработчиков.

Список типов блоков хранит описания, дескрипторы и другую необходимую информацию о типах блоков, которые могут быть использованы при создании/исполнении модели DPMine. Список является динамически расширяемым и может быть дополнен новыми типами при помощи других плагинов. Основной плагин DPMine регистрирует только базовые типы блоков, такие как: *схема*, блоки управ-

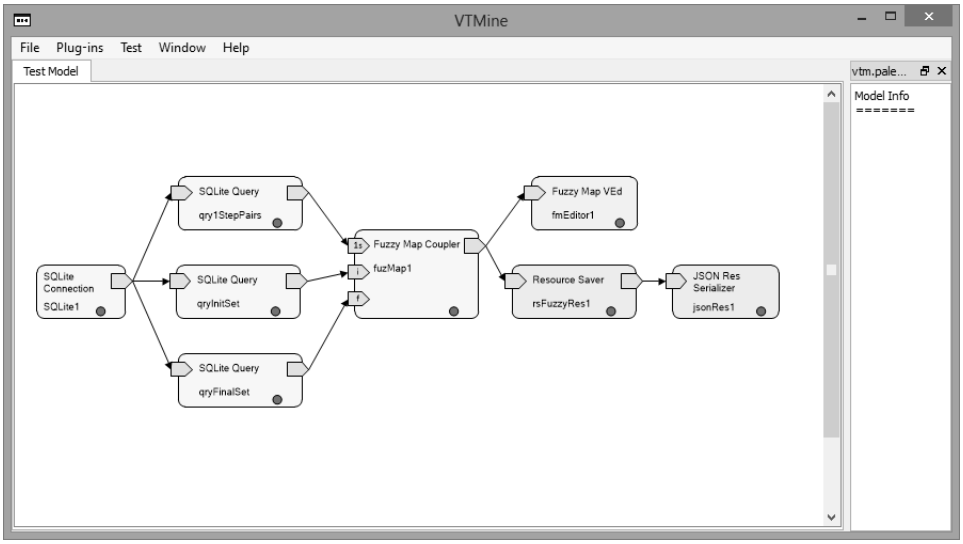


Рис. 5. Экранная форма приложения VTMine с *видом* модели DPMine

ления потоком исполнения, блоки, реализующие выражения, константы и т.д. Блоки, специфичные для отдельных предметных областей, например Process Mining, регистрируются при помощи других плагинов.

Сериализаторы — динамически расширяемые компоненты, осуществляющие сохранение/загрузку блоков из какого-либо формата сериализации (например XML или JSON). Блок сериализации предоставляет интерфейс, позволяющий регистрировать как новые форматы сериализации, так и связи между форматами и отдельными типами блоков при помощи *сериализаторов блоков* (block serializers).

Графический редактор — визуальный компонент, отвечающий за взаимодействие между моделями DPMine и приложением VTMine на уровне пользовательского интерфейса. Он позволяет создавать, изменять и исполнять модели DPMine, представленные в графической форме (рис. 5).

Редактор представляется в приложении VTMine как *вид*, создаваемый и добавляемый к списку видов текущего окна. Эта операция может быть осуществлена, например, если к ресурсу вида *модель DPMine* применяется обработчик *графический редактор моделей DPMine*. Работа с такой моделью осуществляется по аналогии с другими видами моделей, такими как графы и сети Петри.

Взаимодействие между пользователем и графической моделью осуществляется через команды, контекстные для каждого блока. Так, блоки специального типа, используемые для визуализации ресурсов модели, содержат обработчик команды, результатом вызова которой является создание нового вида (в терминах VTMine), содержащего (как правило, графическое) представление ресурса.

Визуальное представление любого блока модели определяется рядом факторов, включающих: тип блока, его текущие параметры (для блоков, имеющих настраиваемые параметры), текущий статус блока и выбранную схему рендеринга.

3. Построение нечеткой модели процесса по журналу событий, хранящемуся в БД

В [18] рассматривается подход к извлечению нечеткой модели (fuzzy map) [19] из лога данных, представленного в виде реляционной базы данных (РБД). Этот подход основан на последовательности SQL-запросов, формирующих необходимые наборы данных для визуализации такой модели.

Предложенный подход может быть реализован в виде модели DPMine, содержащей следующие блоки (тип/имя) (рис. 5):

1) блок `SQLite Connection/SQLite1` — источник данных в РБД SQLite;
2) блоки `SQLite Query/qry1StepPairs, qryInitSet, qryFinalSet` — табличные наборы данных, сформированные в результате выполнения соответственно запросов: а) отношения между всеми активностями за один шаг, б) множество начальных вершин, в) множество конечных вершин;
3) блок `Fuzzy Map Coupler/fuzMap1` — объединяет три источника данных, описанных выше, в виде единой (объектной) нечеткой модели;
4) блок `Fuzzy Map VEd/fmEditor1` — является точкой включения графического редактора нечетких моделей, связывающих объектную модель с графической;
5) блок `Resource Saver/rsFuzzyRes1` — компонент для сохранения временного ресурса в глобальном менеджере ресурсов;
6) блок `JSON Res Serializer/jsonRes1` — осуществляет сериализацию (сохранение) ресурса (нечеткой модели) в виде JSON-файла.

Для схемы, представленной на рис. 5, допустимо несколько различных последовательностей исполнения блоков с учетом требований, накладываемых семантикой исполнения моделей языка DPMine.

Первым блоком, подлежащим исполнению, является блок `SQLite Connection/SQLite1`, предоставляющий доступ к базе данных SQLite [20]. Блок включает информацию, необходимую для осуществления подключения к этой БД, такую как имя файла, параметры открытия и другие параметры, описанные в интерфейсе программирования (API) SQLite. Если все параметры блока `SQLite Connection/SQLite1` позволяют осуществить подключение к БД, блок в процессе исполнения осуществляет такое подключение и делает возможным использование БД блоками, связанными с этим подключением.

Блоки `SQLite Query` используются для формирования SQL-запроса к БД и предоставления результата таких запросов в виде таблицы данных, соответствующих интерфейсу `IDataSet`. В модели (рис. 5) используется три таких блока: `qry1StepPairs`, `qryInitSet`, `qryFinalSet`. Они формируют из исходной базы данных (представляющей лог) путем соответствующей выборки три различных набора данных, необходимых для формирования графа нечеткой модели. Это выборка 1-шаговых переходов между отдельными активностями, включенными в результирующий набор данных, и выборки начальных и конечных активностей.

Блок `Fuzzy Map Coupler` осуществляет связывание трех наборов данных в виде единой объектной модели. Необходимым условием для успешного исполнения данного блока является наличие на его входном порту `1s` (1-шаговых переходов)

подготовленного набора данных. Два других входных порта `i` и `r` — соответствуют начальным и конечным позициям — являются опциональными и могут быть опущены.

Нечеткие модели представляются в виде графа, содержащего активности в качестве вершин и отношения между ними в качестве дуг. Таким образом, в результате исполнения этого блока формируется выходной ресурс, являющийся графом, который определяется списком смежности, сформированным из множества 1-шаговых переходов. Граф дополняется связями из искусственной вершины «начало» к вершинам из множества начальных вершин (второй набор данных) и связями из множества конечных вершин к еще одной искусственной вершине «конец».

Блок `Fuzzy Map VEd` используется для визуализации полученной объектной модели. В основе этого блока лежит компонент-редактор графовых моделей, модифицированный для случая нечетких моделей.

В результате исполнения блока `Fuzzy Map VEd/fmEditor1` производится проверка доступности компонента редактирования графов, который загружается в приложение VTMine в виде отдельного плагина, но при этом не является обязательным для удовлетворения зависимостей плагинов, загружающих рассматриваемые в настоящем разделе блоки VTMine.

В случае наличия этого компонента производится необходимая инициализация и блок получает статус «*исполненный*». Это означает, что блок становится доступным для интерактивного взаимодействия с пользователем, который с помощью контекстного меню может выполнить команду «Отобразить редактор», в результате чего будет создан новый *вид* (с встроенным графическим редактором), который будет добавлен к остальным видам текущего окна приложения VTMine.

Блок `Resource Saver/rsFuzzyRes1` своим входом подключен непосредственно к выходу блока `Fuzzy Map Coupler/fuzMap1`, точно так же, как и блок `Fuzzy Map VEd/fmEditor1`. Это означает, что оба блока-потребителя (объектной) нечеткой модели работают с одним и тем же экземпляром ресурса (моделью).

Ресурсы, продуцируемые блоками модели DPMine, по умолчанию находятся в состоянии композиции с блоками, которые их порождают, и последние управляют временем жизни таких ресурсов. Соответственно, эти *вре́менные* ресурсы удаляются при повторном исполнении модели (за исключением некоторых специальных блоков) или при удалении самой модели. Блок `Resource Saver/rsFuzzyRes1` используется для сохранения копии этого ресурса модели с помощью *глобального менеджера ресурсов* для дальнейшего использования.

Блок `Resource Saver/rsFuzzyRes1` подключен своим выходным портом к входному порту последнего рассматриваемого в рамках данной схемы блока — `JSON Res Serializer/jsonRes1`. Этот блок в процессе своего выполнения получает на вход объект, реализующий интерфейс `IResource`, и осуществляет его сохранение в виде сериализованного JSON-файла.

Расширение компонента сериализации `JSON Res Serializer/jsonRes1` по типам ресурсов (которые он может сериализовывать) осуществляется таким же образом, как и в случае с расширением компонента сериализации блоков модели DPMine.

Результатом *исполнения модели*, ее главной схемы (рис. 5) и всех входящих в нее блоков осуществляется открытие базы данных с логом событий, формирование трех запросов с извлечением нужных наборов данных, агрегация этих наборов в ви-

де объектной Fuzzy-модели, являющейся ресурсом VTMine, предоставление графического компонента для визуализации этой модели, сохранение копии этой модели в виде постоянного ресурса и запись ее в виде сериализованного JSON-файла.

4. Заключение

В работе описана концепция языка DPMine и его реализация в виде библиотеки DPMine/C. Интеграция библиотеки в систему моделирования VTMine осуществлена на основе механизма плагинов.

Основная работа, которую предстоит сделать, заключается в разработке новых типов блоков для поддержки большего числа алгоритмов Process Mining. Следует улучшить инструменты VTMine для создания и анализа как моделей DPMine, так и моделей, традиционно используемых в области Process Mining. Необходимо выполнить работу для осуществления поддержки распараллеливания вычислений сложных алгоритмов.

Наконец, отдельной задачей является интеграция VTMine с существующими решениями в области Process Mining, такими как ProM.

Список литературы

1. *James Manyika, Michael Chui, Brad Brown et al.* Big data: The next frontier for innovation, competition, and productivity // Technical Report. 2011.

2. *Accorsi R., Damiani E., van der Aalst W.* Unleashing Operational Process Mining (Dagstuhl Seminar 13481) // Dagstuhl Reports. 2014. Vol. 3, No. 11. P. 154–192.

3. *van der Aalst W. M. P.* Process Mining — Discovery, Conformance and Enhancement of Business Processes. Springer, 2011. P. I–XVI, 1–352. ISBN 978-3-642-19344-6.

4. *IEEE Task Force on Process Mining:.* Process Mining Manifesto. BPM 2011 Workshops / Ed. by F. Daniel, S. Dustdar, K. Barkaoui // *Lecture Notes in Business Information Processing.* Vol. 99. Springer-Verlag, Berlin, 2011. P. 169–194.

5. *Mitsyuk A., Kalenkova A., Shershakov S., van der Aalst W.* Using process mining for the analysis of an e-trade system: A case study // *Business Informatics.* 2014. Vol. 3.

6. *Rubin V., Lomazova I., van der Aalst W. M.* Agile development with software process mining // ICSSP. 2014. Nanjing, Jiangsu, China. ACM, 2014.

7. *Rubin V., Mitsyuk A. A., Lomazova I. A., van der Aalst W. M. P.* Process mining can be applied to software too! // Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. NY: ACM, 2014.

8. *Мицюк А. А., Шугуров И. С.* Синтез моделей процессов по журналам событий с шумом // Моделирование и анализ информационных систем. 2014. Т. 21, № 4. С. 181–198. (*Mitsyuk A. A., Shugurov I. S.* On Process Model Synthesis Based on

Event Logs with Noise // Modeling and Analysis of Information Systems. 2014. Vol. 21, No 4. P. 181–198 [in Russian].)

9. *Shugurov I., Mitsyuk A. A.* Generation of a set of event logs with noise // Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering, SYRCoSE 2014 / Ed. by Alexander Kamkin, Alexander Petrenko, Andrey Terekhov. ISP RAS, 2014.

10. *Object Management Group (OMG)* Business process model and notation (BPMN) version 2.0 // Report, 2011.

11. *Alves A., Arkin A., Askary S. et al.* Web Services Business Process Execution Language Version 2.0 (OASIS Standard). WS-BPEL TC OASIS. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html. — 2007.

12. *Van der Aalst W. M. P., ter Hofstede A. H. M.* YAWL: Yet another workflow language // Inf. Syst. 2005. Vol. 30, No. 4. P. 245–275.

13. *Shershakov S.* DPMine: modeling and process mining tool // Proceedings of the 7th Spring/Summer Young Researchers Colloquium on Software Engineering. SYRCoSE 2013. — ISP RAS, 2013.

14. *Shershakov S.* DPMine/P: modeling and process mining language and ProM plug-ins // Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia / Ed. by Andrey N. Terekhov, Maxim Tsepkov. ACM, New York, USA, 2013.

15. *Verbeek H.M.W., Buijs J.C.A.M., van Dongen B.F., van der Aalst W.M.P.* ProM 6: The Process Mining Toolkit // Proceedings of BPM Demonstration Track 2010 / Ed. by M. La Rosa. CEUR Workshop Proceedings. 2010. Vol. 615 P. 34–39.

16. *Shershakov S.* DPMine/C: C++ library and graphical frontend for DPMine workflow language // Proceedings of the 8th Spring/Summer Young Researchers' Colloquium on Software Engineering, SYRCoSE 2014 / Ed. by Alexander Kamkin, Alexander Petrenko, Andrey Terekhov. ISP RAS, 2014. P. 96–101.

17. *Kim P., Bulanov O., Shershakov S.* Component-based VTMine/C framework: Not only modelling // Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering, SYRCoSE 2014 / Ed. by Alexander Kamkin, Alexander Petrenko, Andrey Terekhov. ISP RAS, 2014. P. 102–107.

18. *Shershakov S. A.* VTMine Framework as Applied to Process Mining Modeling // *International Journal of Computer and Communication Engineering*. 2014. In press.

19. *Günther C. W., van der Aalst W. M. P.* Fuzzy mining: Adaptive process simplification based on multi-perspective metrics // Proceedings of the 5th International Conference on Business Process Management, BPM'07. Berlin, Heidelberg : Springer-Verlag, 2007. P. 328–343.

20. URL: `http://www.sqlite.org/`.

DPMine Graphical Language for Automation of Experiments in Process Mining

Shershakov S. A.

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

Keywords: process mining, modeling language, automation, experiments, fuzzy model

Process mining is a new direction in the field of modeling and analysis of processes, where the use of information from event logs describing the history of the system behavior plays an important role. Methods and approaches used in the process mining are often based on various heuristics, and experiments with large event logs are crucial for the study and comparison of the developed methods and algorithms. Such experiments are very time consuming, so automation of experiments is an important task in the field of process mining. This paper presents the language DPMine developed specifically to describe and carry out experiments on the discovery and analysis of process models. The basic concepts of the DPMine language as well as principles and mechanisms of its extension are described. Ways of integration of the DPMine language as dynamically loaded components into the VTMine modeling tool are considered. An illustrating example of an experiment for building a fuzzy model of the process discovered from the log data stored in a normalized database is given.

Сведения об авторе:
Сергей Андреевич Шершаков,
Национальный исследовательский университет «Высшая школа экономики»,
науч. сотр. международной научно-учебной лаборатории
процессно-ориентированных информационных систем