MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 27, NO. 2, 2020
journal homepage: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

"VTMine for Visio": Graphical Tool for Modeling in Process Mining

S. A. Shershakov[1]

DOI: 10.18255/1818-1015-2020-2-194-217

[1]National Research University Higher School of Economics, 20 Myasnitskaya st., Moscow 101000, Russia.

MSC2020: 68U35, 93A30
Research article
Full text in Russian

Received May 25, 2020
After revision June 5, 2020
Accepted June 10, 2020

Process-Aware Information Systems (PAIS) is a special class of the IS intended for the support the tasks of initialization, end-to-end management and completion of business processes. During the operation such systems accumulate a large number of data that are recorded in the form of the event logs. Event logs are a valuable source of knowledge about the actual behavior of a system. For example, there can be found information about the discrepancy between the real and the prescribed behavior of the system; to identify bottlenecks and performance issues; to detect anti-patterns of building a business system. These problems are studied by the discipline called "Process Mining".

The practical application of the process mining methods and practices is carried out using the specialized software for data analysts. The subject area of the process analysis involves the work of an analyst with a large number of graphical models. Such work will be more efficient with a convenient graphical modeling tool. The paper discusses the principles of building a graphical tool "VTMine for Visio" for the process modeling, based on the widespread application for business intelligence Microsoft Visio. There are presented features of the architecture design of the software extension for application in the process mining domain and integration with the existing libraries and tools for working with data. The application of the developed tool for solving various types of tasks for modeling and analysis of processes is demonstrated on a set of experimental schemes.

Keywords: process modeling; process mining; experiment models; graphical tool; experiments automation.

INFORMATION ABOUT THE AUTHORS

Sergey A. Shershakov | orcid.org/0000-0001-8173-5970. E-mail: sshershakov@hse.ru
correspondence author | researcher.

Funding: This work is supported by the Basic Research Program at the National Research University Higher School of Economics.

For citation: S. A. Shershakov, ""VTMine for Visio": Graphical Tool for Modeling in Process Mining", *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 194-217, 2020.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 27, № 2, 2020
сайт журнала: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

“VTMine for Visio”: инструмент графического моделирования в области Process Mining

С. А. Шершаков[1]

DOI: 10.18255/1818-1015-2020-2-194-217

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

УДК 519.682.6+004.94
Научная статья
Полный текст на русском языке

Получена 25 мая 2020 г.
После доработки 5 июня 2020 г.
Принята к публикации 10 июня 2020 г.

Процессно-ориентированные информационные системы (ПОИС) – специальный класс ИС для поддержки задач по инициализации, сквозному управлению и завершению бизнес-процессов. В процессе функционирования такие системы накапливают большое число данных, которые записываются в виде журналов событий. Журналы событий являются ценным источником знаний о реальном поведении системы. Например, в них можно обнаружить информацию о несоответствии реального и желаемого поведения системы; определить узкие места и проблемы с производительностью; детектировать анти-паттерны построения бизнес-системы. Изучением этих задач занимается дисциплина «Извлечение и анализ моделей процессов» (Process Mining).
Практическое применение методов и практик Process Mining осуществляется с помощью специализированного программного обеспечения (ПО) для аналитиков данных. Предметная область анализа процессов подразумевает работу аналитика с большим числом графических моделей. Такая работа будет более эффективной при наличии удобного инструмента графического моделирования. В настоящей работе рассматриваются принципы построения графического инструмента «VTMine for Visio» моделирования процессов на базе распространенного приложения для бизнес-аналитики Microsoft Visio. Приводятся особенности проектирования архитектуры программного расширения для применения в области Process Mining и интеграции с существующими библиотеками и инструментами для работы с данными. Применение разработанного приложения для решения различного вида задач по моделированию и анализу процессов демонстрируется на наборе схем экспериментов.

Ключевые слова: моделирование процессов; извлечение и анализ моделей процессов; модели экспериментов; графический инструмент; автоматизация экспериментов.

ИНФОРМАЦИЯ ОБ АВТОРАХ

Сергей Андреевич Шершаков
автор для корреспонденции

orcid.org/0000-0001-8173-5970. E-mail: sshershakov@hse.ru
научный сотрудник.

Финансирование: Работа выполнена в рамках Программы фундаментальных исследований НИУ ВШЭ.

Для цитирования: S. A. Shershakov, ““VTMine for Visio”: Graphical Tool for Modeling in Process Mining”, *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 194-217, 2020.

Введение

Последние годы обозначились активным развитием информационных систем (ИС), которые проникли во все сферы человеческой деятельности. Это связано с повышением производительности оборудования, снижением его стоимости, появлением новых инструментов и подходов к разработке ИС и развитием кадрового рынка специалистов в информационных технологиях (ИТ).

Все это повлияло на ускорение окружающих нас бизнес-процессов и увеличение объемов связанных с ними данных. Для поддержки таких процессов было разработано большое количестве ИС, которые сформировали отдельный класс — *процессно-ориентированных информационных систем* (ПОИС). Они представляют собой сложные распределенные программные комплексы, принимающие на себя задачи по инициализации, сквозной поддержке и завершению бизнес-процессов. В процессе функционирования такие системы накапливают большое число побочных данных, которые записываются в *логах*. Логи могут рассматриваться с технической точки зрения (*системные логи*) или с точки зрения *следов поведения системы* относительно событий бизнес-процесса (*логи событий*). Последние являются ценным источником знаний о реальном поведении системы. Сравнивая реальное поведение с ожидаемым, можно обнаружить массу полезной информации: несоответствие реального и желаемого поведения системы; узкие места и проблемы с производительностью; анти-паттерны построения бизнес-системы и т. д.

Изучением этих задач занимается дисциплина «*извлечение и анализ моделей процессов*» (англ. *Process Mining*, *PM*) [1]. Субъектами этой дисциплины являются бизнес-аналитики, владельцы процессов, исследователи данных. Ее основными объектами являются бизнес-процессы. Процессы представляются различными абстракциями в виде статистических, графических и других видов моделей, а также конкретными выполненными экземплярами процессов, записанными в логах событий.

Несмотря на то, что извлечение процессов является относительно новым направлением, оно уже активно применяется при моделировании и анализе бизнес-процессов [2] в менеджменте, при разработке программного обеспечения [3, 4], в управлении технологическими процессами, в медицине [5, 6].

Практическое применение методов и практик Process Mining осуществляется с помощью специализированного программного обеспечения (ПО) для аналитиков данных. ПО может носить исследовательский (академический) характер, быть ориентировано на индустриальное применение, либо совмещать в себе две эти функции. В зависимости от назначения, к ПО могут предъявляться различные требования: по фунциональному наполнению, производительности, модульности, наличии интеграции с существующими инструментами бизнес-аналитики и т.д.

Предметная область анализа процессов подразумевает работу аналитика с большим числом графических моделей. Такая работа будет более эффективной при наличии удобного инструмента графического моделирования. В настоящей работе рассматриваются принципы построения графического инструмента *VTMine for Visio* моделирования процессов на базе распространенного приложения для бизнес-аналитики Microsoft Visio [7]. Показано, как можно расширить функционал существующего инструмента для применения в области Process Mining. Демонстрируется применение разработанного приложения для решения различного вида задач по моделированию и анализу процессов.

Существующие решения

Программные инструменты, связанные с областью Process Mining, которые наиболее часто упоминаются в литературе [8, 9], а также инструменты, получившие развитие в последнее время, можно разбить на условные группы по следующему принципу.

К первой группе относятся инструменты, представляющие графический пользовательский интерфейс (GUI) для моделирования и анализа. К этой группе относятся: *ProM* [10], *RapidProM* [11], *DPMine/P* [12], *Disco* [13], *Celonis* [14], *Minit* [15]. Ко второй группе относятся инструменты, реализованные в виде утилит командной строки: *Petrify* [16, 17], *Rbminer* [18], *genet* [19]. К третьей группе относятся инструменты, реализованные в виде пакетов-расширений к скриптовым языкам Python — *PMLab* [20], *PM4Py* [21] и R — *bupaR* [22].

Из представленных инструментов поддержку графического управления потоками задач (workflow) имеют только два инструмента — *RapidProM* и *DPMine/P*, оба являются надстройками над инструментом *ProM*.

Остальная часть статьи организована следующим образом. В разделе 1 рассматривается объектная модель инструмента MS Visio и ее особенности, позволяющие выполнять расширение этого инструмента для задач Process Mining. В разделе 2 обсуждаются особенности проектирования архитектуры приложения для графического моделирования с учетом специфики моделирования процессов. В частности, в разделе 2.2 рассматривается подсистема моделирования экспериментов *DPMine*, основанная на одноименном графическом языке. В разделе 3 представлены примеры моделей экспериментов, реализованных с помощью разработанного инструмента. Наконец, в заключении подводятся итоги проделанной работы и рассматриваются направления для будущей работы.

1. Объектная модель инструмента MS Visio

Все компоненты приложения MS Visio, включая элементы пользовательского интерфейса, представляются в виде COM-классов с экспортированными интерфейсами и доступны для взаимодействия с программными расширениями, разработанными в виде макросов с помощью интерпретируемого языка сценариев Visual Basic for Application (VBA), либо в виде расширений Visio Add-In на языках программирования платформы .NET (C#, C++/CLI и др.).

Основными компонентами, используемыми для представления документа Visio, являются следующие классы: `Document`, `Page` и `Shape`[1]. Объект класса `Document` представляет один загруженный в память Visio-документ, который может быть файлом с *графической моделью* или рисунком (расширения .vsd, .vdx), файлом с *коллекцией элементов-заготовок* (stencils, расширения .vst и .vtx) либо *файлом-шаблоном* (расширение .vst или .vtx). Документы состоят из «плоского» списка страниц, каждая из которых представляется объектом класса `Page`. Каждая страница содержит коллекцию фигур, имеющих графическое представление. В объектной модели каждой фигуре соответствует объект класса `Shape`. Фигуры на странице упорядочены по z-оси, что определяет их перекрытие друг другом. Фигура может быть *группировочной* (группой фигур), что позволяет задавать иерархию фигур. Каждая группа рассматривается в качестве одной фигуры — на странице, где она лежит, либо внутри другой группы.

1.1. Семантические атрибуты компонентов Visio

Все три ключевых компонента Visio-документа являются *атрибутированными*, то есть их *внешний вид*, *поведение* и взаимодействие с другими компонентами определяется набором атрибутов, приписанных конкретному объекту. Атрибуты в Visio представляются в виде специальной *таблицы свойств* (property sheet). На уровне объектной модели работа с атрибутами осуществляется путем обращения к соответствующим свойствам, полям и методам — в зависимости от вида Visio-расширения и языка программирования. Редактирование некоторых атрибутов доступно с помощью специального визуального редактора (property sheet editor).

[1] Все эти типы определены в пространстве имен `Microsoft.Office.Interop.Visio`

Атрибуты фигур, страниц и документов сгруппированы по функциональному назначению. Например, секция *Shape Transform* содержит атрибуты некоторой фигуры, определяющие ее геометрические размеры и позицию на родительском компоненте (странице или группе фигур); одна или несколько секций *Geometry X* определяют набор графических примитивов, используемых для отрисовки фигур; секции *Line Format*, *Fill Format* и *Text Block Format* определяют такие характеристики, как цвет линии, заливки, тип штриха, параметры шрифта и др.

Значение атрибутов объекта может задаваться как с помощью констант — числовых или символьных литералов, так и с помощью формул-выражений, включающих ссылки на другие атрибуты этого или другого объекта. Это позволяет взаимосвязывать значения атрибутов без необходимости осуществления программного контроля. В общем случае в качестве значения атрибута может выступать строка достаточно большого размера. Использование длинных строк позволяет сериализовать объект произвольной структуры, тип данных которого не является одним из базовых (целое или с плавающей точкой число, список-перечисление, булевый тип, строка и некоторые др.). Значения атрибутов, приписанных любому объекту, автоматически сериализуются и записываются как составная часть документа при сохранении его в долговременной памяти.

В рамках данной работы наибольший интерес представляют следующие важнейшие секции: *Shape Data* и *User-defined Cells*. Эти секции позволяют приписать соответствующему объекту дополнительные атрибуты заданных типов данных, которые будут отождествлены с объектом и также автоматически будут сохраняться как часть документа и восстанавливаться при его загрузке. Однако в отличие от стандартных атрибутов использование данных атрибутов является *опциональным* и определяется соответствующим программным расширением.

Различие между секциями *Shape Data* и *User-defined Cells* заключается в их назначении. Атрибуты секции *Shape Data* доступны для обращения через управляющий элемент пользовательского интерфейса *Shape Data Pane*. С помощью этой панели пользователь задает значения параметров, ассоциированных с конкретной фигурой. С этой целью каждый атрибут, описанный в данной секции, содержит помимо уникального имени ряд параметров, определяющих его визуальное представление: тип данных, способ форматирования, отображения и др. Атрибуты такого типа совмещают в себе одновременно элементы «модели, представления и контроллера» при рассмотрении их с точки зрения паттерна проектирования «модель-вид-представление» (MVC). Примерами таких атрибутов являются атрибуты «стоимость», «владелец», «дата», приписываемые фигурами модели блок-схемы; атрибуты «категории», «исполнители» блока «задача» модели BPMN и др.

В отличии от секции *Shape Data* секция *User-defined Cells* содержит набор атрибутов, которые не имеют визуального представления, и соответствуют только элементу «модель» паттерна MVC. Такие атрибуты определяются только своим именем и значением, тип которого является универсальным и требует соответствующей интерпретации при обращении к нему программным образом. Атрибуты секции *User-defined Cells* не отображаются в стандартной панели *Shape Data Pane*, однако доступны программно, а также посредством непосредственного обращения к таблице свойств соответствующего объекта. Назначением атрибутов такого типа является т. н. «семантическое атрибутирование», то есть добавление к некоторому объекту (фигуре или странице) произвольных дополнительных атрибутов, не определенных типами данных этих объектов (`Shape` и `Page` соответственно). Данные атрибуты используются для задания таких свойств объектов, непосредственный доступ к которым пользователя приложения не требуется либо нежелателен. Примерами моделей, использующих в основном атрибуты секции *User-defined Cells*, являются модели семейства UML, доступные в стандартной поставке Visio. С помощью таких атрибутов осуществляет управление поведением таких элементов UML-моделей, как *Class*, *Relation* и др. Задание свойств этих объектов осуществляется с помощью специальных визуальных управляющих окон, содержащих сложные специализированные элементы управления, комбинация значений которых записыва-

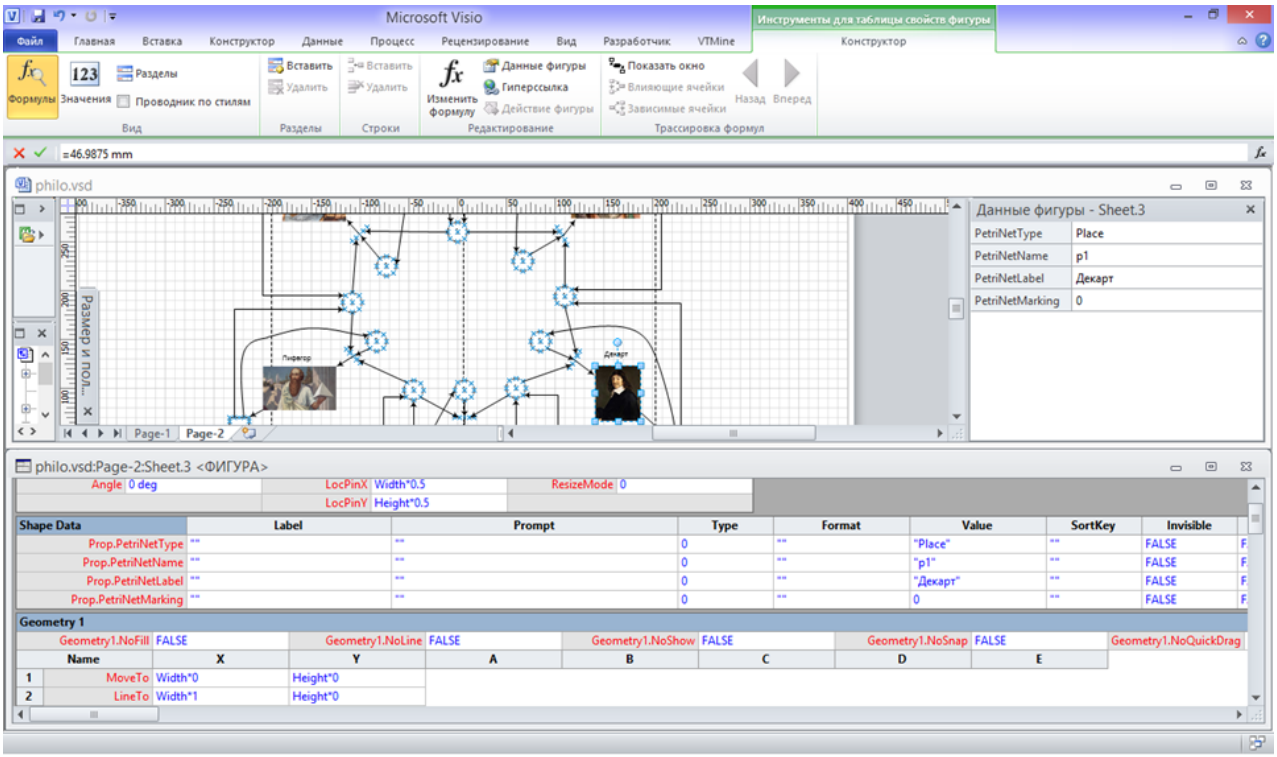


Fig. 1. Graphical representation of a Petri net modeling *the Dining Philosopher Problem*

Рис. 1. Графическое представление сети Петри, моделирующей *задачу об обедающих философах*

ется в соответствующий атрибут или набор атрибутов программно с помощью соответствующего программного расширения (add-in).

Программное обращение к обоим типам «семантических» атрибутов осуществляется практически идентично с помощью уникального имени атрибута и префикса, определяющего секцию: `User` для атрибутов секции *User-defined Cells* и `Prop` для атрибутов секции *Shape Data*. Далее мы будем ссылаться на них как на *пользовательские атрибуты закрытого* и *открытого* типов соответственно.

Использование *пользовательских атрибутов* позволяет рассматривать документ Visio не только в качестве векторного рисунка, но и наделяет его семантическими свойствами. На рисунке 1 изображен пример известной задачи «обедающих философов» [23, 24], которая моделируется с помощью *сети Петри*. Из иллюстрации видно, что рисунок модели содержит несколько изображений известных ученых и философов, каждое из которых является графическим примитивом Visio «растровый рисунок». Эти графические примитивы связаны с другими графическими примитивами (черными вытянутыми прямоугольниками, кругами, квадратами) с помощью направленных коннекторов. Вместе они образуют графическое представление сети Петри, в которой круги традиционно обозначают позиции, квадраты и прямоугольники — переходы, а направленные коннекторы — дуги.

Элементы пользовательских свойств этих фигур содержат именованные атрибуты, определяющие их семантическое назначение: `PetriNetType` задает тип элемента (позиция, переход, дуга); `PetriNetName` определяет уникальное имя в рамках данной модели; `PetriNetMarking`, имеющий смысл только для позиций, определяет текущую разметку в виде числа токенов и т.д. Таким образом, помимо собственно атрибутов, определяющих графическое представление, каждая фигура содержит дополнительные атрибуты, используемые расширением для анализа сетей Петри, позволяющее рассматривать данную графическую модель в виде абстрактной модели сети Петри с разметкой.

Рассмотренный подход лежит в основе программного расширения *VTMine for Visio*. Так, добавление *пользовательских атрибутов* на уровень *документа* позволяет определить такой документ как документ специального вида — *VTMine for Visio*-документ. При загрузке такого документа осуществляется активизация подсистемы программного расширения *VTMine for Visio*, загрузка соответствующих элементов пользовательского интерфейса и др. Далее мы будем обозначать эти документы как *VTMDoc*. Добавление *пользовательских атрибутов* на уровень *страницы* позволяет указать, что данная страница содержит не просто графический рисунок, но *графическую модель* (или ее часть), за взаимодействие с которой отвечает соответствующий компонент или подсистема *VTMine for Visio*. Наконец, добавление *пользовательских атрибутов* на уровень *фигуры* позволяет определить, что данная фигура является составной частью некоторой модели.

Таким образом, разработка программных расширений Visio совместно с использованием пользовательских атрибутов для ключевых компонентов объектной модели позволяет расширить его возможности до *инструмента графического моделирования*.

2. Разработка архитектуры приложения: задачи и особенности

При разработке архитектуры *VTMine for Visio* учитывались следующие требования и ограничения:

- Приложение представляет собой COM-расширение для Visio (то есть является плагином), разработанное на языке программирования платформы .NET.
- Функциональность приложения расширяется плагинами (принцип «плагины для плагина»).
- Ядро приложения содержит только базовые компоненты взаимодействия с Visio и не содержит каких-либо специфических предметным областям компонент (например, типы моделей).
- С целью рационального использования ресурсов приложение активизируется только при работе с документами *VTMine for Visio*.

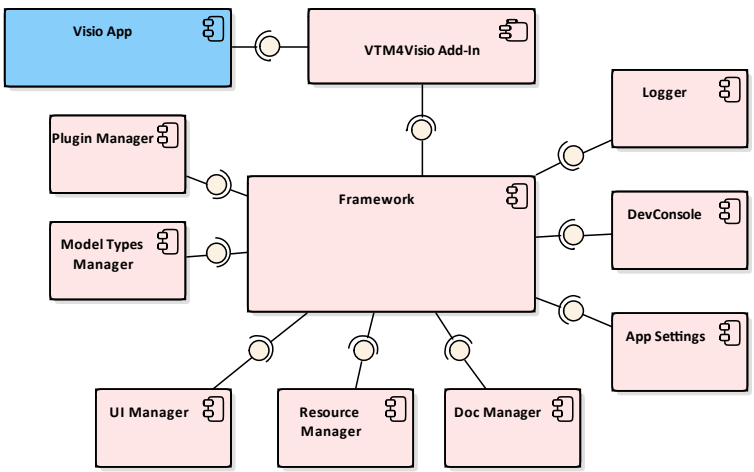


Fig. 2. *VTMine for Visio* basic components

Рис. 2. Ключевые компоненты *VTMine for Visio*

Разработанное приложение *VTMine for Visio* представляет собой Visio COM Add-In [25]. Основной язык программирования — C#. Технически, расширение является множеством сборок, оформленных в виде библиотек динамического подключения (.dll), манифестами и другими вспомогательными файлами. Основным файлом расширения является библиотека `ru.xidv.vtm4visio.dll`. Путь к директории с этой библиотекой обозначается символом `$(VtmAppDir)`.

Структурная схема компонентов ядра *VTMine for Visio* показана на рисунке 2. Далее рассматриваются назначение и особенности проектирования наиболее важных компонентов с учетом выделенных выше требований.

2.1. Расширение Visio (Add-In) и фреймворк приложения

Основным классом, связывающим приложение Visio (класс `Application`) с расширением, является класс `ThisAddIn` (на рисунке — *VTM4Visio Add-In*). Инициализация расширения производится посредством передачи управления коду расширения при возникновении события `Startup`. В обработчике события `Startup` происходит создание и инициализация главного компонента ядра расширения — объекта *Framework*.

Объект *Framework* (фреймворк) может находиться в нескольких состояниях. Изначальное состояние *Inactive* означает, что фреймворк не инициализирован. После успешной подготовки основных компонентов состояние меняется на *Initialized*, то есть инициализирован. Данное состояние подразумевает загрузку всех зависимостей компонента ядра (плагинов) в режиме минимального потребления ресурсов. При открытии документа *VTMDoc* активизируются все необходимые компоненты, распределяются ресурсы, подготавливаются необходимые элементы пользовательского интерфейса, а компонент фреймворка переходит в состояние *Active*. Наконец, после закрытия всех документов *VTMDoc* расширение переходит в состояние *Ready*, которое отличается от *Initialized* тем, что часть ресурсов являются активированными для быстрого старта при открытии следующего релевантного документа, а от состояния *Ready* — тем, что дорогостоящие ресурсы выгружены из памяти.

При начальной инициализации фреймворка (с состояния *Inactive* до состояния *Initialized*) происходит создание следующих компонентов ядра: *логгера, централизованного компонента настроек, отладочной консоли, менеджера документов, менеджера пользовательского интерфейса.* При последующей активации фреймворка (с состояния *Initialized* до состояния *Active*) создаются и инициализируются компоненты: *менеджер ресурсов, менеджер типов моделей, менеджер плагинов.*

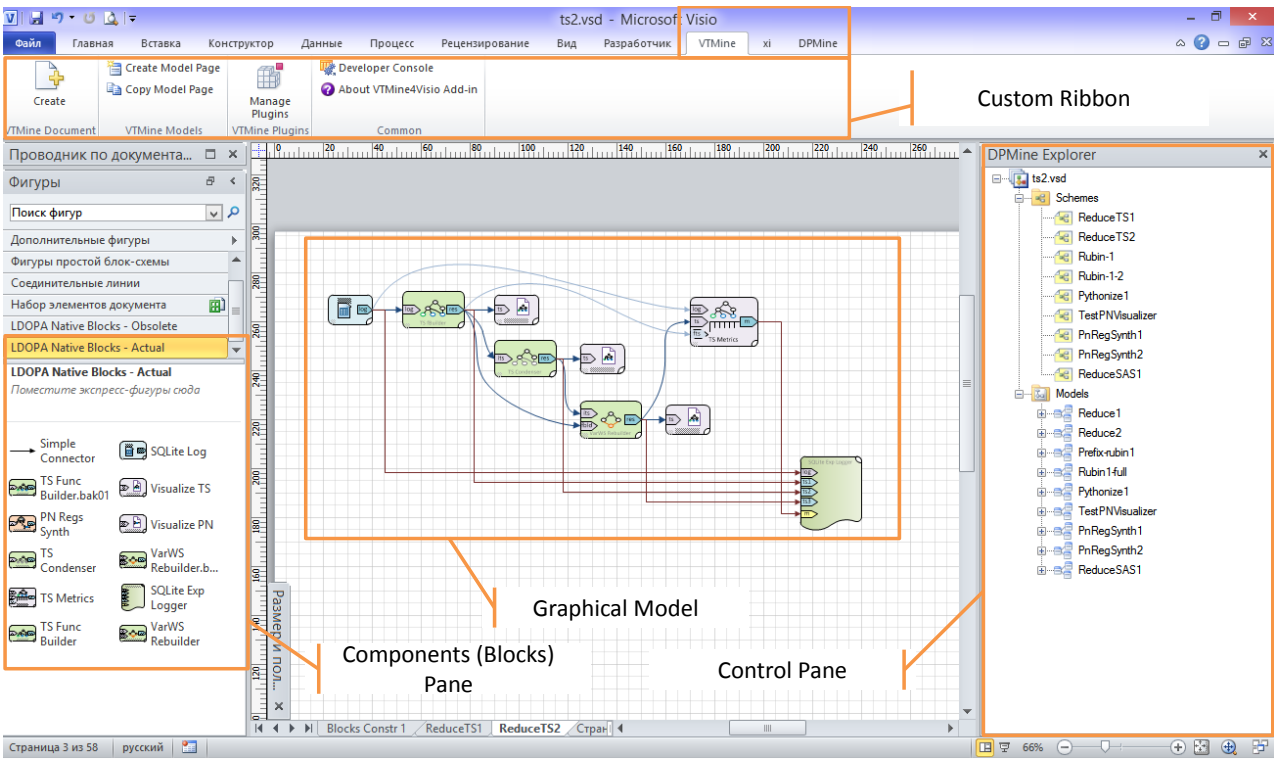


Fig. 3. Controls configured through *User Interface Manager*

Рис. 3. Элементы управления, настраиваемые через *менеджер пользовательского интерфейса*

Успешная загрузка расширения подразумевает создание расширения элемента управления «Лента»: туда добавляется новая вкладка *VTMine*, содержащая ряд управляющих компонентов для взаимодействия пользователя с надстройкой (см. рисунок 3).

Связь фреймворка с объектом Visio-приложения осуществляется путем подписки фреймворка на события приложения, среди которых события создания, открытия, закрытия документа; события переключения между окнами, смены активного окна; события добавления, удаления страницы и фигуры, смены активной страницы и изменения выделенных компонентов и др. Специальное событие `MarkerEvent` позволяет централизовано обрабатывать активацию пунктов пользовательского контекстного меню фигур и страниц, что является важным элементом взаимодействия непосредственно элементов моделей, разрабатываемых пользователем, и компонентов расширения, обрабатывающих такие модели.

В приложении используется иерархическая система распространения событий. Это означает, что события, порождаемые непосредственно объектом Visio-приложения, обрабатываются только фреймворком, который делегирует дальнейшую их обработку соответствующим компонентам, либо обрабатывает сам. Никакие другие компоненты системы не подписываются на события приложения, что позволяет управлять последовательностью их обработки и исключить проблемы, связанные с неопределенностью порядка их обработки. Последнее особенно важно с учетом возможного наличия других установленных расширений Visio.

2.1.1. Менеджер документов

Управление активными документами приложения Visio осуществляется компонентом *менджер документов* (*Document Manager*). Он является основным обработчиком следующих событий: *создание нового документа*, *открытие существующего документа* и *перед закрытием открытого документа*. При возникновении первых двух событий менеджер документов проверяет, является ли создаваемый или открываемый документ помеченным специальным образом. Для этого считывается его *пользовательский атрибут* `VTMDocument`, который определяет, что документ является документом *VTMDoc*. При первом обращении к такому типу документов осуществляет активацию фреймворка до состояния *Active*.

Создание документа *VTMDoc* имеет свои особенности. Visio предоставляет несколько способов создания нового документа. Пользователю доступны возможности создания нового документа на основе шаблона либо пустого документа. В первом случае атрибут `VTMDocument` добавляется в шаблон, поэтому все созданные на его основе документы также будут рассматриваться как *VTMDoc*. В случае создания пустого документа установка каких-либо пользовательских атрибутов невозможна. Для упрощения процедуры создания *VTMDoc*-документа без использования шаблона предусмотрена кнопка «Create» страницы *VTMine* ленты (рисунок 3).

Использование *семантических атрибутов* позволяет внедрить большое количество служебной информации, релевантной для документов *VTMDoc*, непосредственно в файл документа. Тем не менее в процессе моделирования часто создаются большие по объему ресурсы, хранение которых путем встраивания в документ не является целесообразным. Для связывания файла *VTMDoc*-документа с внешними используемыми им ресурсами разработан подход представления *VTMDoc*-документа в виде *решения* аналогично тому, как это происходит при объединении нескольких связанных файлов с исходными кодами в единый программный проект. С этой целью файл *VTMDoc*-документа (с расширением `.vsd`) рассматривается в качестве главного файла *проекта*, в дополнение к которому в директории с `.vsd`-файлом создается поддиректория с расширением .data. Например, для документа `model.vsd` его рабочая директория имеет имя `model.data`. При активизации документа `model.vsd` указанная директория доступна всем компонентам *VTMine for Visio*, включая плагины.

2.1.2. Менеджер плагинов

Одним из требований, предъявляемых к приложению *VTMine for Visio*, является возможность расширения его функционала за счет динамически загружаемых *плагинов*. Учитывая, что *VTMine for Visio* сам по себе является плагином для Visio, поставленная цель может быть достигнута двумя основными путями: 1) дополнительные модули загружаются в виде самостоятельных Visio Add-In; 2) существует единственный Visio Add-In (*VTMine for Visio*), который самостоятельно управляет загрузкой динамических расширений.

Первый подход обладает рядом недостатков, связанных с необходимостью синхронизации отдельных модулей друг с другом, отсутствием возможности контроля порядка загрузки отдельных расширений, что порождает ряд сложно решаемых проблем. Второй подход требует разработки подсистемы управления расширениями, однако позволяет централизованно ими управлять, что снимает ряд проблем и делает его предпочтительным. Именно он реализован в *VTMine for Visio*. В результате использования единственной точки загрузки динамических модулей для *VTMine for Visio* система плагинов приобретает вид, как на рисунке 4.

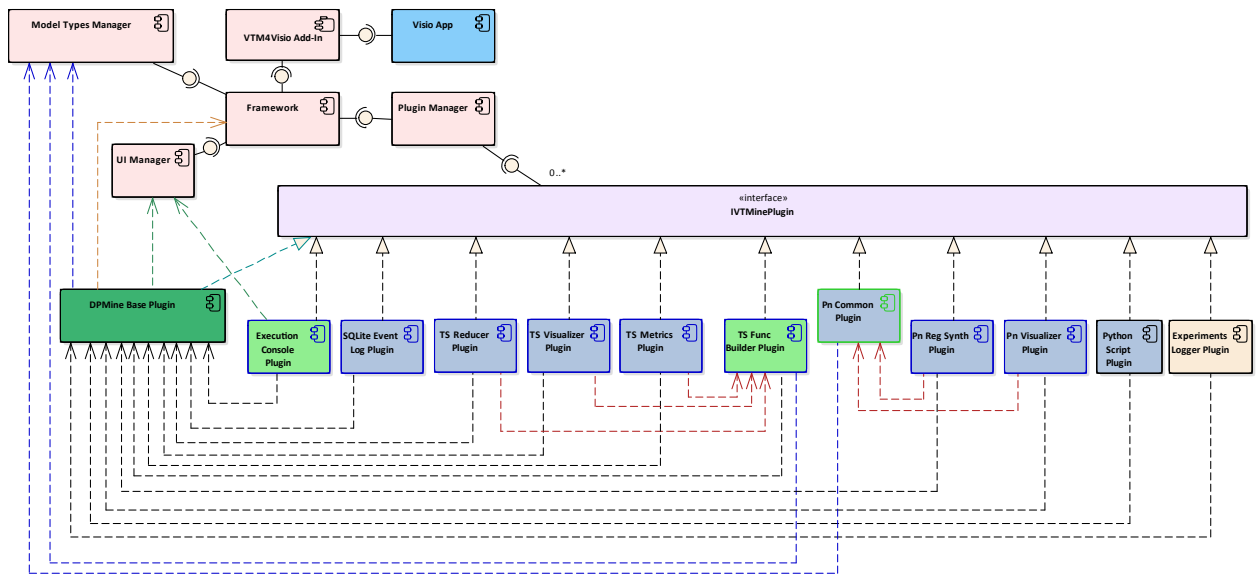


Fig. 4. Main *VTMine for Visio* plugins used in the paper

Рис. 4. Основные *VTMine for Visio*-плагины, используемые в данной работе

Плагин представляет собой специальным образом оформленный программный компонент, реализующий один или более интерфейсов, и имеющий уникальный строковый идентификатор. Основным интерфейсом, определяющим назначение компонента как VTM-плагина, является интерфейс `IVTMinePlugin`. Каждый плагин является одним из возможных *объектов расширений*, который подключается к совместимой по типу *точке расширения*. Плагины содержатся в файлах — библиотеках динамического подключения (.NET-сборках). Один такой файл может содержать несколько плагинов, что позволяет упростить распространение группы плагинов, связанных общей тематикой.

Плагины могут находиться в зависимости друг от друга, как например плагины, представленные на UML-диаграмме компонентов (рисунок 4). Большинство представленных на диаграмме плагинов имеют зависимость от плагина `DPMine Base Plugin`. Он реализует подсистему *DPMine* [12, 26] в приложении *VTMine for Visio*. Исключениями здесь являются сам плагин `DPMine Base Plugin` и плагин `Pn Common Plugin`. Это определяется тем, что в данной конфигурации приложения большинство

плагинов расширяют базовый плагин *DPMine* путем добавления новых *DPModel*-блоков либо (как в случае с плагином `Execution Console Plugin`) добавляют дополнительные компоненты пользовательского интерфейса (консоль исполнения DPM-моделей) для управления DPM-моделями.

Одним из наиболее важных компонентов ядра *VTMine for Visio* является *менеджер плагинов*. Он управляет загрузкой и инициализацией плагинов для приложения *VTMine for Visio*.

Менеджер плагинов получает управление от ядра приложения при активации последнего, то есть при переходе в состояние *Active*. Первым делом менеджер загружает конфигурационную информацию, которая хранится в специальном конфигурационном файле `plugins.json`.

Зависимость между плагинами не задается статически, а определяется динамически на этапе их загрузки. Это определяется тем фактом, что сразу несколько плагинов могут являться *объектом расширения* для одной и той же *точки расширения*. Как правило, только один из таких плагинов загружается в конкретной конфигурации, однако его идентификатор до момента загрузки неизвестен. Этот факт не позволяет построить граф зависимостей между плагинами [27] до начала их загрузки и выполнить последующую топологическую сортировку на нем для определения оптимального порядка загрузки плагинов. *Алгоритм загрузки плагинов* в упрощенном виде выглядит следующим образом.

1. Формируется список всех файлов с незагруженными плагинами — *кандидатами на загрузку*.
2. Для очередного файла из списка кандидатов загружается аннотация плагина. Аннотация включает подпрограмму, которая анализирует текущее окружение на предмет возможности удовлетворения зависимостей плагина.
3. Выполняется проверка возможности загрузки текущего плагина путем подпрограммы проверки удовлетворения зависимостей в аннотации плагина.
4. Если зависимости текущего плагина удовлетворены, он загружается и выставляется флаг наличия загруженных плагинов. Это означает, что какие-то плагины, которые не могли быть загружены на предыдущих итерациях, теперь могут быть загружены, так как загрузка текущего плагина удовлетворяет их зависимости.
5. Если есть еще плагины-кандидаты, которые не загружались в текущем цикле, выполняется переход к пункту 2.
6. Если все кандидаты проверены, остались незагруженные плагины и за текущий цикл был загружен хотя бы один плагин, выполняется переход переход к п. 1.
7. Если за текущий цикл а) все плагины были или б) ни один плагин не был загружен, — что означает наличие неразрешаемых зависимостей, — алгоритм завершается.

В худшем случае, когда последовательность плагинов для загрузки алгоритмом такова, что каждый текущий плагин зависит от следующего, сложность этого алгоритма будет квадратичной от числа плагинов. Вместе с этим в результате выполнения алгоритма плагины в текущей конфигурации загружаются в некотором определенном порядке. Если сохранить этот порядок и использовать его при следующей загрузке плагинов (при условии, что не были добавлены новые или исключены существующие плагины), то загрузка плагинов алгоритмом будет осуществлена за линейное от числа плагинов время. Сохранение оптимального порядка осуществляется в конфигурационном файле.

При загрузке некоторого плагина менеджер плагинов передает ему управление путем вызова метода `LoadPlugin`, имеющего двоичный параметр `NeedReg`. Реализация метода `LoadPlugin` индивидуальна для каждого плагина и позволяет ему взаимодействовать со всеми компонентами приложения, включая компоненты ядра и другие плагины. Взаимодействие с компонентами ядра позволяет плагину выполнять кастомизацию базовых подсистем приложения, например: регистрировать новые типы моделей (через *менеджер типов моделей*, раздел 2.1.4), добавлять перспективы и элементы управления (через *менеджер пользовательского интерфейса*, раздел 2.1.3), регистрировать

обработчики для событий документов (через *менеджер документов*, раздел 2.1.1) и т. д. Взаимодействие с другими плагинами позволяет расширять их функциональность посредством представляемого ими API. В качестве примера можно привести подсистему графического моделирования *DPMine*, которая полностью построена на плагинах и расширяется с помощью других плагинов (раздел 2.2).

Флаг `NeedReg` выставляется при самой первой загрузке этого плагина, что может быть им использовано для выполнения *первичных* операций по регистрации в системе: копирование необходимых файлов с ресурсами, инициализация конфигурационных параметров и др. При последующих загрузках зарегистрированный плагин выполняет только *динамические* операции по расширению соответствующих подсистем.

2.1.3. Менеджер пользовательского интерфейса

Взаимодействие *VTMine for Visio* с графическим пользовательским интерфейсом (ГПИ) Visio осуществляется посредством *менеджера пользовательского интерфейса*. Он включает уровень абстракции, позволяющий описывать расширение ГПИ Visio компонентами ядра и плагинами приложения, с использованием определенного набора компонентов. Среди них основными являются *перспективы*, *модификаторы ГПИ*, *элементы управления* и *фабрики элементов управления*.

К *элементам управления*, которые могут быть добавлены ядром и плагинами *VTMine for Visio*, относятся пользовательские *ленты* (ribbon), *панели управления* и *палитры компонентов* (рисунок 3). *Ленты* используются для добавления новых меню/кнопок/выпадающих списков и других простых элементов управления, посредством которых осуществляется взаимодействие пользователя и компонентов приложения. На рисунке 3 показано содержимое ленты *VTMine*, которая автоматически отображается для всех документов при инициализации основного add-in-компонента приложения. Также показано, что в меню доступна еще одна лента *DPMine*, выбор которой появляется только при активации страницы модели с типом *DPModel*. *Панели управления* — это более крупные элементы пользовательского интерфейса, позволяющие интегрировать в пользовательский интерфейс сложные диалоговые окна или масштабируемые панели со списками, иерархическими структурами и др. На рисунке показана панель *DPMine Explorer*, которая также как и лента *DPMine* активируется только для моделей типа *DPModel*. *Палитры компонентов* — это специальный тип документов, которые открываются в панели *Фигуры*, и содержат подготовленные к использованию заготовки фигур — так называемые *образцы* или *трафареты* (stencils).

Множество пользовательских элементов управления, отображаемых в каждый конкретный момент времени, определяется текущей активной *перспективой*. При переключении между страницами документа также происходит переключение между перспективами, заданными для конкретных страниц, поэтому набор видимых элементов меняется. Перспектива, назначенная некоторой странице, определяется, в первую очередь, видом модели, которая на этой странице отображается. Это позволяет отображать именно те элементы управления, которые релевантны данной конкретной модели, и скрывать остальные. Например, моделям *DPModel* по умолчанию назначается перспектива, включающая *ленту DPMine* и *панель управления DPMine Explorer*.

Менеджер пользовательского интерфейса регистрирует одну *перспективу по умолчанию*, которая отображается для всех страниц документа, для которых не назначена какая-либо другая перспектива. Эта перспектива включает ленту *VTMine*, которая содержит команду создания документа *VTMDoc*. Регистрация других перспектив осуществляется плагинами. Для этого определен специальный интерфейс *модификатора ГПИ* (`IUICustomizer`), реализовав который некоторый плагин включается в цикл управления пользовательским интерфейсом.

Таким образом, некоторый плагин, который осуществляет расширение пользовательского интерфейса, регистрирует в *менеджере пользовательского интерфейса* компонент — *модификатор ГПИ*, который реализует указанный выше интерфейс. Сам *менеджер* при активизации обращается ко всем

зарегистрированным модификаторам, запрашивая у них регистрацию новых перспектив, затем — модификацию всех зарегистрированных перспектив. При первой активизации некоторой перспективы включенные в нее элементы управления создаются путем обращения к соответствующему модификатору, после чего отображаются в приложении.

2.1.4. Менеджер типов моделей

Основное назначение *VTMine for Visio* — просмотр и редактирование графических моделей. *Документ* Visio рассматривается в качестве коллекции графических моделей, связанных общим предметом исследования. Одна модель может располагаться на одной или более *страницах* документа. В последнем случае речь идет о многостраничных моделях, например, иерархических.

VTMine for Visio не имеет фиксированного набора разновидностей моделей. Вместо этого новые типы моделей могут добавляться в систему с помощью плагинов. Управление типами моделей осуществляется компонентом ядра системы — *менеджер типов моделей*.

Разновидность (тип) модели описывается с помощью *дескриптора модели*. Дескриптором модели является объект, реализующий интерфейс `IModelDescriptor`. Такой объект предоставляет информацию об идентификаторе, наименовании типа модели, компоненте (плагине), зарегистрировавшем дескриптор. Одно из основных предназначений дескриптора модели — создавать правильным образом подготовленные страницы Visio. Запрос к дескриптору на создание модели может осуществляться со стороны компонента ядра (например, при выборе пользователем соответствующей команды) или плагина (например, при синтезе новой модели в результате работы некоторого алгоритма).

2.2. Подсистема моделирования *DPMine*

Одним из основных расширений приложения *VTMine for Visio*, реализованных в рамках настоящей работы, является подсистема поддержка моделирования экспериментов Process Mining с помощью графического языка *DPMine* [12, 26]. В настоящем разделе представляются базовые архитектурные подходы, лежащие в основе этого расширения. Демонстрируются возможности *VTMine for Visio* по интеграции с библиотекой *DPModel* [28], а также с библиотекой *LDOPA* алгоритмов и структур данных для Process Mining [29—31].

На рисунке 4 представлена компонентная диаграмма основных плагинов, относящихся к подсистеме *DPMine*. На диаграмме видно, что все плагины реализуют интерфейс `IVTMinePlugin`, посредством которого они регистрируются в *менеджере плагинов* и взаимодействуют с приложением. Представленные на рисунке плагины имеют следующее назначение (в скобках после названия плагина указан язык программирования, на котором он разработан, и библиотека, функциональность которой плагин вводит в подсистему *DPMine*).

- *DPMine Base Plugin* (C#) является *базовым плагином* подсистемы *DPMine* и определяет ее взаимодействие с приложением *VTMine for Visio*.
- *Execution Console Plugin* (C#) расширяет функциональность *базового плагина*, представляя реализацию *консоли исполнения модели*. Базовый плагин предоставляет только *точку расширения* для включения такой консоли, оставляя возможность для конкретной реализации другим плагинам.
- *SQLite Event Log Plugin* (C++/CLI, библиотека *LDOPA*) добавляет в подсистему *DPMine* определение нового блока *DPModel*, предоставляющего абстрактный интерфейс *журнала событий*, который реализован в виде журнала событий SQLite [29].
- *TS Func Builder Plugin* (C++/CLI, библиотека *LDOPA*) добавляет блок *DPModel* для синтеза систем переходов по журналу событий [32] и определение типа модели *VTMine "система переходов с частотными пометками"*. На примере этого плагина ниже приводятся особенности расширения подсистемы *DPMine*.

- *TS Reducer Plugin* (C++/CLI, библиотека *LDOPA*) добавляет блоки *DPModel* для поэтапного выполнения редукции систем переходов в соответствии с алгоритмом, представленным в [32].
- *TS Visualizer Plugin* (C++/CLI) добавляет блок *DPModel* для визуализации моделей — *систем переходов с частотными пометками*.
- *TS Metrics Plugin* (C++/CLI, библиотека *LDOPA*) добавляет блоки *DPModel* для расчета метрик систем переходов [32].
- *Pn Common Plugin* (C++/CLI) включает общие компоненты для работы с сетями Петри и добавляет определения типа соответствующей модели *VTMine*.
- *Pn Reg Synth Plugin* (C++/CLI, библиотека *LDOPA*) добавляет блок *DPModel* для выполнения синтеза сетей Петри по системам переходов модифицированной версией алгоритма регионов [33].
- *Pn Visualizer Plugin* (C++/CLI) добавляет блок *DPModel* для визуализации моделей — *сетей Петри*.
- *Experiments Logger Plugin* (C#) добавляет блок *DPModel* для конфигурируемого сбора и регистрации в базе данных расчетных значений, получаемых при выполнении эксперимента/исполнении модели *DPModel*.
- *Python Script Plugin* (C++/CLI) добавляет блок *DPModel* для выполнения произвольного Python-скрипта в контексте исполнения модели *DPModel* [31].

Все обозначенные плагины зависят от *базового плагина*, `DPMine Base Plugin`. Проверка на удовлетворение такой зависимости осуществляется процедурой `LoadPlugin` при загрузке очередного плагина. Если *базовый плагин* еще не загружен к моменту загрузки очередного плагина, то загрузка последнего откладывается. Помимо зависимости от *базового плагина* некоторые из отображенных на схеме плагинов имеют и другие зависимости. Так, плагины *TS Reducer Plugin*, *TS Visualizer Plugin* и *TS Metrics Plugin* зависят от плагина *TS Func Builder Plugin*, а плагины *Pn Reg Synth Plugin* и *Pn Visualizer Plugin* — от плагина *Pn Common Plugin* (рисунок 4). Характер зависимости плагинов друг от друга может быть разным. В одних случаях зависимые плагины расширяют функционал плагина-зависимости; большинство плагинов в примере выше расширяют функционал базового плагина. В других случаях зависимые плагины используют некоторые компоненты плагина-зависимости; так, *Pn Common Plugin* содержит базовые определения для моделей на основе сетей Петри, и зависящие от него плагины используют их.

2.2.1. Базовый плагин *DPMine* и его расширение

Принцип расширения одного плагина другими рассматривается на примере *базового плагина DPMine* (`DPMine Base Plugin`). Структура основных компонентов, входящих в состав этого плагина, представлена на диаграмме классов (рисунок 5). На диаграмме помимо компонентов базового плагина представлены некоторые связанные с ним компоненты ядра, другие плагины и библиотеки. Диаграмма включает только часть внутренних и внешних компонентов, которые позволяют проиллюстрировать механизм организации зависимостей компонентов и расширения их друг другом. Внешние по отношению к базовому плагину компоненты заключены в пунктирные прямоугольники и представляют следующие *ограниченные домены* (boundaries).

- `Visio` — содержит основные компоненты приложения Visio, такие как *документ* и *страница*.
- `VTM4Visio Core` — компоненты ядра, на которые влияют загружаемые плагины.
- `Execution Console Plugin` — плагин *консоли исполнения модели*. На диаграмме представлен только своим основным классом `ExecConsolePlugin`.
- `DPModel Library` — библиотека компонентов *DPModel* [28], включающая реализацию языка моделирования *DPMine* и некоторых базовых блоков [26]. Библиотека написана на языке C# и является независимой от инструмента *VTMine for Visio*. Плагины подсистемы *DPModel* инструмента используют ее как внешнюю зависимость.

- `LDOPA Library` — библиотека алгоритмов и структур данных для Process Mining [29, 31]. Библиотека написана на языке C++ и является внешней зависимостью для приложения.
- `TS Func Builder Plugin` — плагин, расширяющий функциональность ядра путем добавления нового типа моделей “системы переходов с частотами” (компонент `FreqTsMD`); также расширяет функциональность *базового плагина* путем регистрации нового типа *DPModel*-блока (`TsFuncBuilderBlock`), который выполняет синтез системы переходов с частотами *префиксного дерева* по журналу событий. Плагин осуществляет интеграцию компонента синтеза библиотеки *LDOPA* в систему моделирования экспериментов *DPMine*.

Базовый плагин DPMine включает следующие основные компоненты.

- *Проводник моделей* DPMine, представленный классом `DpmExplorer`, реализует интерфейс ядра `IUICustomizer`, осуществляет добавление в перспективу моделей *DPModel* управляющий элемент — *панель управления* с иерархическим проводником по моделям *DPMine*, зарегистрированным в текущем документе (см. рисунок 3, справа).
- *Консоль исполнения модели*, представленная интерфейсом `IExecConsole`, реализует расширение пользовательского интерфейса приложения путем добавления *панели управления*, отображающую информацию об исполнении модели *DPModel*. В отличие от *проводника моделей консоль исполнения* не является компонентом, встроенным в базовый плагин. Вместо этого другие плагины могут представлять свою специфическую реализацию консоли и подключать ее к базовому плагину. В данном случае в роли такого плагина выступает класс `ExecConsolePlugin`.
- *Дескриптор блока DPModel*, представленный интерфейсом `IBlockDescriptor`, используется для регистрации в подсистеме *DPMine* типа блока, который может быть инстанцирован в модели *DPModel*. Класс `BlockDescriptor` реализует интерфейс `IBlockDescriptor` и представляет базовый функционал для дескрипторов большинства блоков. Примерами блоков и соответствующих им дескрипторов, регистрируемых непосредственно *базовым плагином*, являются `DPMineBaseBlock`/`DPMineBaseBD` и `DPMineScheme`/`DPMineSchemeBD`. Они представляют *базовый блок* и *схемный блок* модели *DPModel* соответственно [12, 26].
- *Дескриптор типа модели* DPModel представлен классом `DPMineMD`, который реализует интерфейс `IModelDescriptor`. Дескриптор моделей *DPModel* регистрируется в *VTMine for Visio* с помощью *менеджера типов моделей*.
- *Модель* DPModel представлена классом `DPMineModel`. Это невизуальная объектная модель, в которую преобразуется графическая модель эксперимента, составленная в Visio с помощью блоков и коннекторов. Класс `DPMineModel` расширяет функционал класса `ModelImpl`, определенного в библиотеке *DPModel*, а также реализует интерфейс `IVTModel` моделей уровня приложения *VTMine for Visio*.
- *Менеджер моделей* DPModel представлен классом `ModelManager`, используется для регистрации и исполнения моделей *DPModel*, а также координирует взаимодействие этих моделей с окружением *VTMine for Visio*. Модели *DPModel* группируются по документам Visio с помощью вспомогательного компонента DPModel-*профиль документа*, который представлен классом `DocDpmProfile`.

Плагин `TS Func Builder Plugin` содержит компоненты для синтеза систем переходов по журналам событий. Алгоритм синтеза реализован как часть библиотеки *LDOPA* и представлен на диаграмме классами `TsBuilder` и `PrefixStateFunc`[2].

Вместе с этим плагин также расширяет систему *типов блоков* DPModel с помощью *дескриптора типа блоков* `TsFuncBuilderBD`, наследующего у класса `BlockDescriptor` по аналогии с дескрипторами `DPMineBaseBD` и `DPMineSchemeBD`. Одна из основных задач, которые возлагаются на *дескрипторы блоков* DPModel, — это конструирование блоков невизуальной (объектной) *модели* DPModel по их

[2]Используется для выведения состояния системы переходов с помощью префиксов [32, 34].

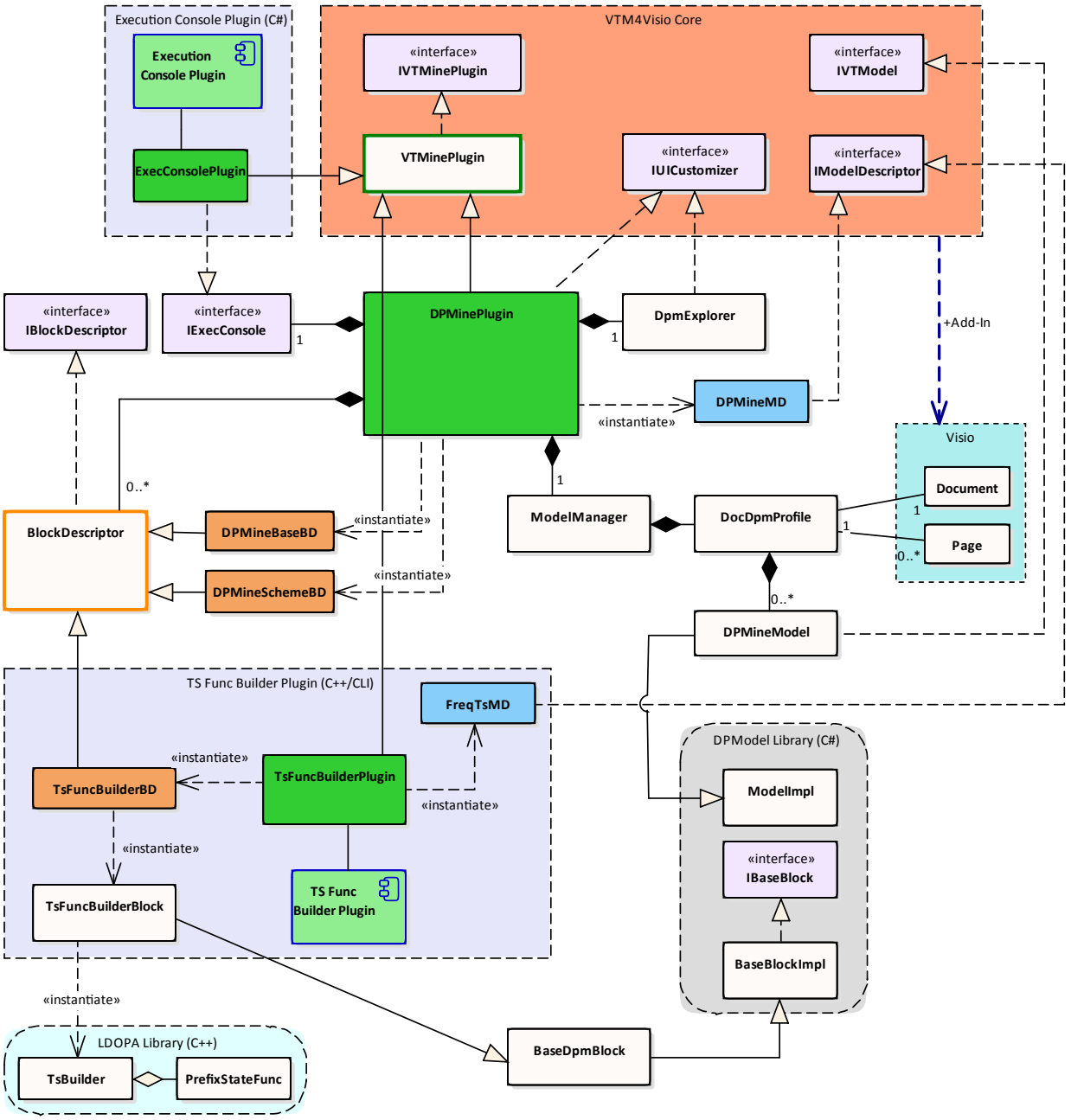


Fig. 5. Components of the *DPMine* Base Plugin and their interaction with libraries and other plugins (external dependencies are represented by dashed rectangles)

Рис. 5. Компоненты базового плагина *DPMine* и их взаимодействие с библиотеками и другими плагинами (внешние зависимости представлены пунктирными прямоугольниками с прямыми или скругленными углами)

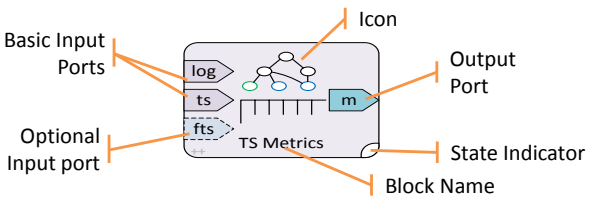
графическому представлению, которое задается фигурой Visio (классом `Shape`). Сконструированный таким образом блок (для плагина синтеза систем переходов задается классом `TsFuncBuilderBlock`) впоследствии добавляется в модель *DPModel* и соединяется коннекторами с другими блоками в соответствии с тем, как их визуальные представления соединены коннекторами на уровне графической модели.

3. Примеры моделей экспериментов Process Mining

В данном разделе приводятся примеры моделей экспериментов Process Mining, отражающие следующие операции: работу с журналом событий; синтез моделей процессов по журналу событий и по другим моделям процессов; визуализацию моделей процессов; расчет характеристик моделей (метрик); сохранение произвольной информации в структурированном виде (БД) для последующей статистической обработки.

Основные принципы организации *моделей DPModel* рассматриваются в [12, 26]. *Графическая модель DPModel* транслируется в *объектную модель* и затем исполняется. *Графическая модель* состоит из одной или нескольких схем, каждая схема представляется в виде одной страницы Visio. На рисунке 3 представлен пример схемы с именем `ReduceTS2`, расположенной на одноименной странице (список страниц отображается в нижней части экранной формы), которая является *главной схемой* для модели `Reduce2` (список моделей отображен в *проводнике моделей* в правой части окна).

Фигура Visio (объект класса `Shape`), представляющая блок, помечается атрибутом `User.DPMine_Type = ''block''`. Атрибут `User.DPMine_Block_Type` определяет конкретное значение идентификатора типа блока, регистрируемого в системе с помощью соответствующего дескриптора типа блока.



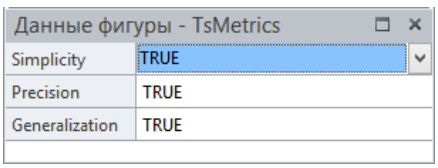


Fig. 6. Graphic notation (a) and the parameter panel (b) of the graphic block for *calculation of transition system metrics*

Рис. 6. Условное графическое обозначение (a) и панель параметров (b) графического блока *расчета метрик систем переходов*

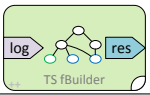
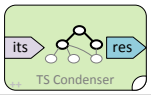
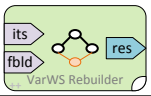
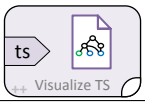
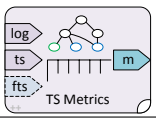
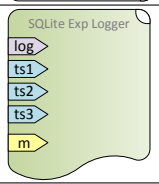
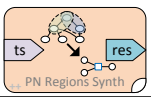
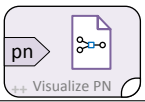
Графическое представление блока может быть любым, однако система конвертации ожидает наличие в фигуре блока вложенных фигур, атрибутированных специальным образом, которые представляют специальные элементы блока. На рисунке 6*a* представлен блок *расчета метрик систем переходов*, добавляемый в систему плагином *TS Metrics Plugin*. Фигура блока содержит ряд вложенных фигур, обозначающих *входные* и *выходные порты* (аннотированные атрибутами `User.DPMine_Type = ''port''` и `DPMine_Pore_Dir = ''in''` и `''out''` соответственно), *индикатор выполнения* (аннотирован атрибутом `User.DPMine_Type = ''execindicator''`) и текстовые, и графические обозначения блока. Блок содержит ряд параметров, доступных для редактирования пользователем через панель инструментов *«Данные фигуры»* (рисунок 6*b*). Параметры задаются с помощью переключателей `TRUE/FALSE`, что обозначает соответственно выполнение или невыполнение расчета каждой из трех метрик. Эти параметры ассоциируются с конкретной фигурой блока с помощью атрибутов `Prop.calc_simplicity`, `Prop.calc_precision` и `Prop.calc_general`.

Перечень блоков, используемых в разработанных моделях, приведен в таблице 1.

3.1. Модель «Синтез системы переходов по журналу событий»

Простейшая *DPModel*-модель синтеза системы переходов по журналу событий представлена на рисунке 7. Модель представлена одной схемой, которая состоит из трех последовательно соединенных блоков: *блока SQLite-журнала событий*, *блока параметрического синтеза систем переходов* и *блока визуализации помеченной системы переходов*.

Table 1. *DPModel* blocks description **Таблица 1.** Блоки *DPModel*

Блок	Описание
	Доступ к журналу событий в формате *SQLite EventLog* [29]. Плагин *SQLite Event Log Plugin*
	Синтез системы переходов по журналу событий [32]. Плагин *TS Func Builder Plugin*
	Построение конденсированной системы переходов (шаг 2 алгоритма редукции; [32]). Плагин *TS Reducer Plugin*
	Восстановление конденсированной системы переходов до редуцированной (шаг 3 алгоритма редукции; [32]). Плагин *TS Reducer Plugin*
	Визуализатор помеченной системы переходов. Плагин *TS Visualizer Plugin*
	Расчет метрик системы переходов [32]. Плагин *TS Metrics Plugin*
	Логирование результатов экспериментов в БД SQLite. Плагин *Experiments Logger Plugin*
	Синтез сети Петри по системе переходов методом регионов [33]. Плагин *Pn Reg Synth Plugin*
	Визуализатор сетей Петри. Плагин *Pn Visualizer Plugin*

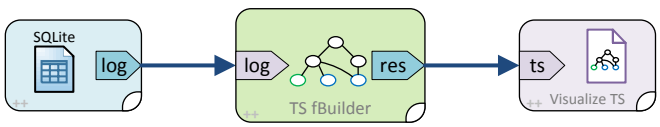


Fig. 7. Model of the synthesis of transition systems based on event logs, and their visualization

Рис. 7. Модель синтеза системы переходов по журналу событий и визуализация модели

Блок журнала событий содержит параметры: *имя файла журнала* (*File Name*) и *строка с конфигурационным SQL-выражением* (*Get Config Query*; [29]). *Блок синтеза систем переходов* содержит параметры: *размер окна* (*Wnd Size*) и *функция, используемая для выведения состояния* (*State Function*). Значение –1 в качестве размера окна обозначает, что оно не ограничено, а `Prefix` в качестве *функции выведения состоянии* задает использование префиксов [32, 34]. Эта совокупность параметров конфигурирует *блок синтеза системы переходов* на построение *префиксного дерева*. Такая конфигурация используется, например, при визуализации журнала событий.

При исполнении модели, *блок журнала событий* открывает файл БД журнала и подготавливает запросы в соответствии с конфигурацией, извлекаемой из журнала конфигурационным запро-

сом [29]. *Блок синтеза систем переходов*, подключенный при помощи коннектора своим входным портом к выходному порту *блока журнала событий* синтезирует систему переходов [32], которая подается на вход *блока визуализации системы переходов*. Этот последний блок создает новую страницу-модель уровня приложения *VTMine for Visio* и отрисовывает на ней синтезированную систему переходов с помощью аннотированных фигур Visio.

Сама страница модели также аннотируется с целью добавления справочной информации, содержащей временную метку создания модели, источник данных для нее, параметры алгоритма синтеза и др. Пример синтезированного с помощью такой модели префиксного дерева и аннотации к нему приведен на рисунке 8.

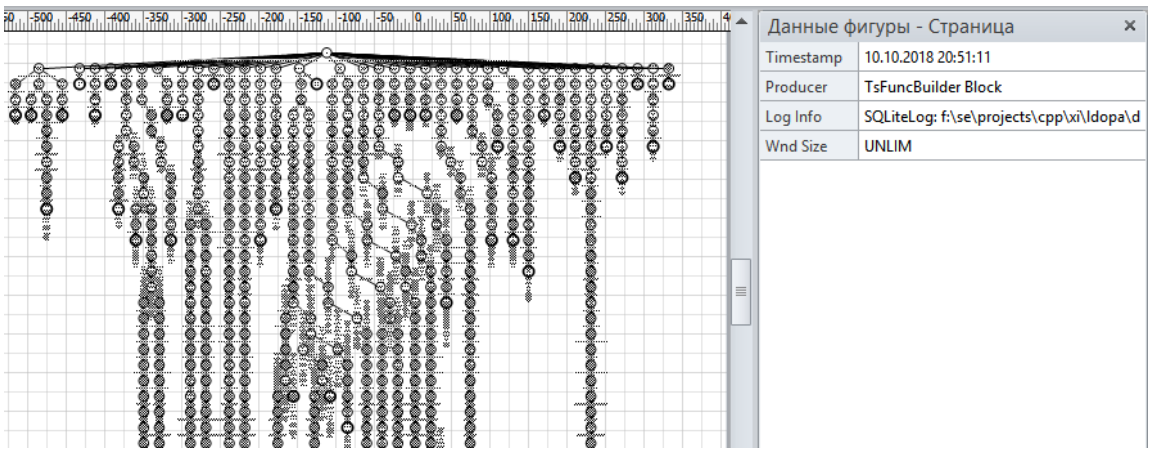


Fig. 8. A prefix tree model and its annotation with information about the source of the model

Рис. 8. Модель «префиксное дерево» и ее аннотация с информацией об источнике модели

3.2. Модель «Синтез и редукция системы переходов с расчетом метрик»

Более сложная версия модели эксперимента, включающая синтез префиксного дерева по журналу событий с последующей редукцией и расчетом метрик редуцированной системы [32], представлена на рисунке 3. Модель синтеза с редукцией является расширением модели синтеза (рисунок 7) и включает следующие новые блоки: *блок построения конденсированной системы*, *блок восстановления конденсированной системы переходов до редуцированной*, *блок расчета метрик системы переходов* и *блок логирования результатов эксперимента в БД SQLite* (см. таблицу 1 для расшифровки графических обозначений блоков).

Данная модель включает в себя полный цикл алгоритма редукции системы переходов, описанный в [32]. Шаги 1, 2 и 3 алгоритма выполняются отдельными блоками, позволяющими изменять отдельные параметры синтеза, конденсации и восстановления, а также — строить графические представления промежуточных моделей. Для этого в модели присутствует три *блока визуализации систем переходов*, подключенных коннекторами к соответствующим выходным портам блоков синтеза, конденсации и восстановления[3].

При исполнении модели подготавливается журнал событий и осуществляется синтез префиксного дерева. После этого становится активным *блок конденсации*, который на основе префиксного дерева строит конденсированное дерево. Оба этих дерева являются обязательными входными параметрами для *блока восстановления*, поэтому его входные порты соединены с выходными портами соответственно *блока синтеза системы переходов* и *блока конденсации*. Результатом работы *блока вос-*

[3]Разные цвета, форма и толщина линий коннекторов, изображенных на схеме модели, несут одинаковую семантическую нагрузку.

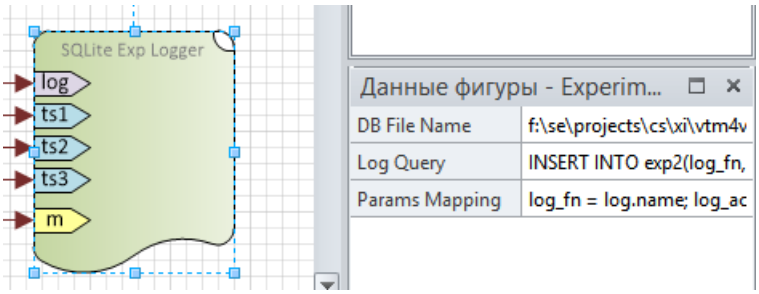


Fig. 9. *Experiment logging block* parameters

Рис. 9. Параметры *блока логирования результатов эксперимента*

становления является результирующая *редуцированная система переходов*, которая визуализируется вместе с двумя другими промежуточными деревьями.

После проведения полного цикла синтеза редуцированной системы переходов становится возможным рассчитать ее метрики, что осуществляется *блоком расчета метрик*. Он содержит три входных порта, которые соединяются с выходными портами *блока журнала событий* (для получения некоторой статистической информации об исходном журнале), *блоком синтеза системы переходов* (для получения характеристик полной системы/префиксного дерева) и *блоком восстановления* — метрики рассчитываются для редуцированной системы переходов, порождаемой именно этим блоком.

Самым последним блоком, который исполняется в этой модели эксперимента, является *блок логирования результатов эксперимента*. Этот блок может иметь произвольное число входных портов, которые должны иметь различимые имена. *Блок логирования* используется для извлечения контрольных данных из подключенных к нему блоков и сохранения их в базу данных, накапливающую результаты экспериментов для их последующего анализа. Этот блок имеет три параметра, настраиваемых пользователем (рисунок 9). Параметр *DB File Name* задает имя файла БД, в который осуществляется запись результатов. Параметр *Log Query* определяет SQL-выражение, осуществляющее добавление одной записи со значениями параметра экспериментов в БД (в одну или несколько таблиц). Параметр *Params Mapping* содержит конфигурационную строку, осуществляющую связь между именами портов и объектами, передаваемыми через них, с параметрами SQL-запроса.

Для рассматриваемой модели SQL-запрос на добавление записи в БД выглядит следующим образом:

```
INSERT INTO exp2(log_fn, log_acts_num, log_traces_num, .....)
VALUES (@log_fn, @log_acts_num, @log_traces_num, .....)
```

Этот запрос осуществляет вставку одной записи в таблицу `exp2` значений атрибутов, заданных именами: `log_fn`, `log_acts_num` и т. д. Значения связываются с запросом посредством соответствующих параметров: `@log_fn`, `@log_acts_num`. Эти параметры, в свою очередь, связываются с внешним окружением *блока логирования* с помощью его параметра *Params Mapping*, принимающего в данном примере следующее значение:

```
log_fn = log.name; log_acts_num = log.actsn; log_traces_num = log.tracesn; .....
```

Params Mapping представляет собой строку пар «параметр = значение», разделенных точкой с запятой. Первый элемент пары представляет собой одноименный, начинающийся со знака `@`, параметр приведенного выше SQL-выражения, второй — выражение, описывающее связь SQL-параметра с входным портом. Например, запись `log_fn = log.name` означает, что значение SQL-параметра `@log_fn` принимается с порта с именем `log`.

Table 2. Database fragment with logged experiment results

Таблица 2. Фрагмент БД с логированными результатами экспериментов

time_st	e...	log_fn	log_acts_num	log_traces_num	log_events_num	ts1_wndsize	ts1_sts_num	ts1_trs_
Click here to define a filter								
2018-10-07 22:59:19.000	(null)	SQLiteLog: F:\se\projects\cpp\xi\ldopa\dev\root\0.1\tests\gtest\work_files\logs\log05.sq3	7	8	41	6	16	
2018-10-07 23:03:51.000	(null)	SQLiteLog: F:\se\projects\cpp\xi\ldopa\dev\root\0.1\tests\gtest\work_files\logs\log05.sq3	7	8	41	6	16	
2018-10-26 13:55:22.000	(null)	SQLiteLog: F:\se\projects\cpp\xi\ldopa\dev\root\0.1\tests\gtest\work_files\logs\log05.sq3	7	8	41	6	16	
2019-04-28 18:37:48.000	(null)	SQLiteLog: F:\research\topics\SoftwarePM\SRA\logs\db\set1\db\all-shorts.sq3	143	243	2444	160	1337	

Входные порты *блога логгера* могут быть подключены к выходным портам тех блоков, которые предоставляют *мультиплексированный ресурс*. *Мультиплексированный* или *композитный ресурс* представляется с помощью объекта, реализующего более одного интерфейса типа ресурсов. Например, *блок синтеза системы переходов* формирует на своем выходном порту *мультиплексированный ресурс*, включающий следующие объекты: *систему переходов* (через нативный тип C++), *журнал событий* (через нативный абстрактный тип C++), *функцию выведения состояния* (через нативный абстрактный тип C++), *список тэгов* (через виртуальное свойство C#), *получатель некоторого свойства* (через виртуальный метод C#). *Выходной порт*, на котором формируется мультиплексированный ресурс, также называется *мультиплексированным*.

Нативные объекты C++, доступные через указанный мультиплексированный порт, могут быть использованы другими блоками, подключенными коннекторами к этому порту, если они умеют работать с нативными объектами (то есть такие блоки сами должны быть реализованы на языке C++/CLI). Так, *блок конденсации*, подключенный к выходному мультиплексированному порту *блока синтеза*, извлекает синтезированную им систему переходов в виде нативного объекта.

Список тегов представляется в виде C#-коллекции *троек строк*. В частном случае каждая такая строка описывает некоторую характеристику некоторого объекта, который полагается вместе со списком тегов на выходном мультиплексированном порте. В примере такой список будет характеризовать *синтезированную систему переходов* следующими атрибутами: временной отметкой создания модели, журнала события, на основе которого эта система синтезирована и т. д. Нетрудно заметить, что именно эта информация выводится *блоком визуализации системы переходов* в виде свойств страницы, на которой данная модель визуализируется (рисунок 8).

Получатель свойства декларируется интерфейсом уровня ядра с именем `IGetProperty` и описывает единственный метод с C#-сигнатурой `Object GetProperty(Object key)`. Этот метод получает произвольный параметр (например, строковый), и возвращает связанное с ним значение. Именно этот метод используется *блоком логирования* для извлечения необходимой ему информации. В рассматриваемом выше примере — выражении `log_fn = log.name` — SQL-параметр `@log_fn` принимает свое значение с *мультиплексированного порта* `log`, на котором формируется ресурс, реализующий интерфейс `IGetProperty`. Вторая часть записи `log.name` (а именно `name`) задает строковый параметр `key`, который передается в метод `GetProperty()`. Полученное значение приводится к совместимому с БД SQLite типу и привязывается к параметру `@log_fn`, после чего попадает в БД в виде зафиксированного значения эксперимента.

Таким способом осуществляется фиксация результатов отдельных экспериментов: характеристик журнала событий, параметров алгоритмов синтеза и редукции и рассчитанных метрик результирующей модели. Фрагмент БД с результатами экспериментов, зафиксированными с помощью *блока логирования*, представлен в таблице 2.

3.3. Модель «Синтез сети Петри по журналу событий методом регионов через промежуточную систему переходов»

На рисунке 10 представлена модель эксперимента, которая осуществляет синтез целевой модели — *сети Петри* алгоритмом регионов [33] через промежуточную систему переходов, построенную по журналам событий. Эта модель является расширением модели на рисунке 7: к выходному порту *блока синтеза системы переходов* помимо *блока визуализации* подключается *блок синтеза сети Петри*.

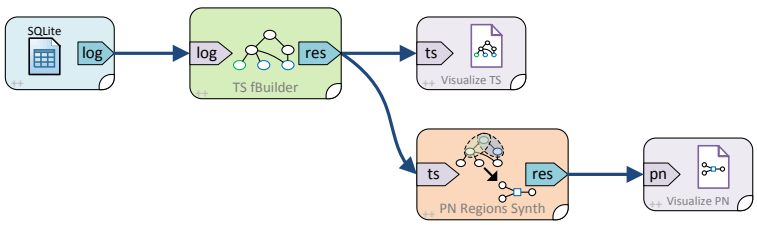


Fig. 10. Model for Petri nets synthesis based on event logs by the algorithms of regions using an intermediate transition system

Рис. 10. Модель синтеза сети Петри по журналу событий алгоритмом регионов через промежуточную систему переходов

Блок визуализации сетей Петри работает аналогично *блоку визуализации систем переходов*, создавая новую страницу-модель и размещая на ней графическое представление синтезированной сети Петри в виде набора связанных фигур. Пример сети Петри, синтезированной с помощью такой модели, представлен на рисунке 11.

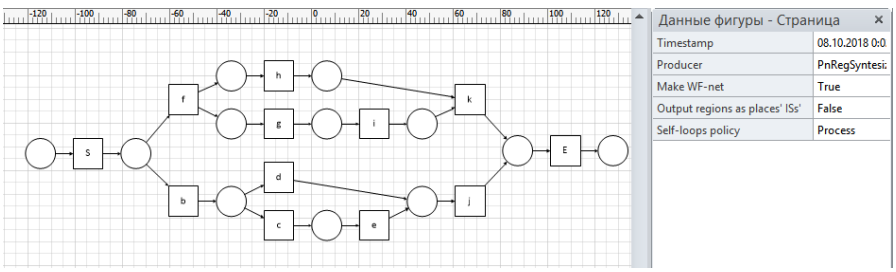


Fig. 11. A Petri net synthesized by the algorithm of regions using a reduced transition system

Рис. 11. Сеть Петри, синтезированная алгоритмом регионов по редуцированной системе переходов

Заключение

В работе рассмотрены принципы разработки графического инструмента моделирования для предметной области извлечения и анализа процессов. Предложен подход по расширению существующего инструмента бизнес-аналитики MS Visio, основанный на семантических атрибутах компонентов Visio. Предложенный подход реализован в виде приложения — расширения (add-in) для MS Visio и набора сопутствующих плагинов, ознакомиться с которыми можно на сайте проекта https://prj.xiart.ru/projects/vtmine4visio. Работа инструмента продемонстрирована на наборе схем экспериментов по синтезу и анализу моделей процессов. В предложенных примерах рассматриваются возможности автоматического построения графических моделей процессов, а также логирования информации о проведенных экспериментах в виде БД для последующей обработки.

Разработанный инструмент использует высокопроизводительную библиотеку *LDOPA* алгоритмов Process Mining, что позволяет расширять его функциональность за счет включения новых

блоков алгоритмов, реализованных в этой библиотеке. Среди направлений будущей работы можно отметить расширение числа поддерживаемых алгоритмов и типов моделей.

References

[1] W. M. P. Van Der Aalst, *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011, pp. I–XVI, 1–352, ISBN: 978-3-642-19344-6.

[2] M. A., K. A., S. S., and W. M. P. Van Der Aalst, "Using process mining for the analysis of an e-trade system: A case study", *Business Informatics*, vol. 29, no. 3, pp. 15–27, 2014.

[3] V. Rubin, L. I., and W. M. P. Van Der Aalst, "Agile Development with Software Process Mining", in *In proceedings: ICSSP 2014, Nanjing, Jiangsu, China*, ACM, 2014, pp. 70–74.

[4] V. Rubin, A. A. Mitsyuk, I. A. Lomazova, and W. M. P. Van Der Aalst, "Process Mining Can Be Applied to Software Too!", in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, NY: ACM, 2014.

[5] R. S. Mans, M. H. Schonenberg, M. Song, W. M. P. Van Der Aalst, and P. J. M. Bakker, "Application of Process Mining in Healthcare – A Case Study in a Dutch Hospital", in *Biomedical Engineering Systems and Technologies*, A. Fred, J. Filipe, and H. Gamboa, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 425–438, ISBN: 978-3-540-92219-3.

[6] R. S. Mans, W. M. P. Van Der Aalst, R. J. B. Vanwersch, and A. J. Moleman, "Process Mining in Healthcare: Data Challenges When Answering Frequently Posed Questions", in *ProHealth/KR4HC*, R. Lenz, S. Miksch, M. Peleg, M. Reichert, D. Riaño, and A. ten Teije, Eds., ser. Lecture Notes in Computer Science, vol. 7738, Springer, 2012, pp. 140–153, ISBN: 978-3-642-36437-2. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36438-9_10.

[7] *Visio Website*. [Online]. Available: https://products.office.com/en-us/visio/flowchart-software.

[8] M. Kebede and M. Dumas, "Kebede, M. and Dumas, M", *University of Tartu*, 2015.

[9] U. Celik and E. Akçetin, "Process mining tools comparison", *Online Academic Journal of Information Technology*, vol. 9, pp. 97–104, 2018.

[10] H. Verbeek, J. Buijs, B. Van Dongen, and W. Van Der Aalst, "ProM 6: The Process Mining Toolkit", in *Proc. of BPM Demonstration Track*, ser. CEUR Workshop Proceedings, vol. 615, 2010, pp. 34–39.

[11] R. Mans, W. M. P. Van Der Aalst, and H. Verbeek, "Supporting Process Mining Workflows with RapidProM", in *Proceedings of the BPM Demo Sessions 2014*, vol. 56, 2014.

[12] S. Shershakov, "DPMine/P: modeling and process mining language and ProM plug-ins", in *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, ACM New York, NY, USA, 2013, ISBN: 978-1-4503-2641-4. DOI: 10.1145/2556610.2556622. [Online]. Available: http://dl.acm.org/citation.cfm?id=2556622&CFID=415147702&CFTOKEN=35395117.

[13] *Fluxicon*. [Online]. Available: http://fluxicon.com/disco.

[14] *Celonis*. [Online]. Available: www.celonis.com.

[15] *Minit*. [Online]. Available: www.minit.io.

[16] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers", *IEICE Transactions on Information and Systems*, vol. E80-D, no. 3, pp. 315–325, 1997.

[17] J. Cortadella, M. Kishinevsky, A. Kondratyev, and L. Lavagno, "Introduction to asynchronous circuit design: specification and synthesis (tutorial)", in *Proceedings of 6th. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, Eilat (Israel)*, 2000.

[18] M. Solé and J. Carmona, "Rbminer: A Tool for Discovering Petri Nets from Transition Systems", in *Automated Technology for Verification and Analysis*, Springer Berlin Heidelberg, 2010, pp. 396–402, ISBN: 978-3-642-15643-4.

[19] J. Carmona, J. Cortadella, and M. Kishinevsky, "Genet: A Tool for the Synthesis and Mining of Petri Nets", in *2009 Ninth International Conference on Application of Concurrency to System Design*, 2009, pp. 181–185.

[20] J. Carmona Vargas and M. Solé, "PMLAB: An Scripting Environment for Process Mining", in *Proceedings of the BPM Demo Sessions'14*, 2014, pp. 16–16.

[21] A. Berti, S. J. Van Zelst, and W. M. P. Van Der Aalst, "Process Mining for Python (PM4Py): Bridging the Gap Between Process-and Data Science", Tech. Rep., 2019.

[22] G. Janssenswillen and B. Depaire, "bupaR: Business Process Analysis in R", in *BPM*, 2017.

[23] E. W. Dijkstra, "Cooperating Sequential Processes", in *TR EWD-123*. 1965.

[24] W. Reisig, *Understanding Petri Nets, Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2010. DOI: 10.1007/978-3-642-33278-4.

[25] *Visio documentation*. [Online]. Available: https://docs.microsoft.com/en-us/office/dev/add-ins/visio/.

[26] S. A. Shershakov, "DPMine graphical language for automation of experiments in process mining", *Automatic Control and Computer Sciences*, vol. 50, no. 7, pp. 477–485, 2016, ISSN: 1558-108X. DOI: 10.3103/S014641161607018X. [Online]. Available: http://dx.doi.org/10.3103/S014641161607018X.

[27] P. Kim, O. Bulanov, and S. Shershakov, "Component-based VTMine/C Framework: Not Only Modelling", in *Proceedings of the 8th Spring/Summer Young Researchers' Colloquium on Software Engineering, SYRCoSE*, ISP RAS, 2014, pp. 102–107. [Online]. Available: http://syrcose.ispras.ru/2014/files/SYRCoSE2014_Proceedings.pdf.

[28] *DPModel Official Website*. [Online]. Available: https://prj.xiart.ru/projects/dpmodel.

[29] S. Shershakov, "Multi-Perspective Process Mining with Embedding Cofigurations into DB-based Event Logs", in *Communications in Computer and Information Science*, In Press, Springer, 2020.

[30] *LDOPA Official Website*. [Online]. Available: https://prj.xiart.ru/projects/ldopa.

[31] S. Shershakov, "Enhancing Efficiency of Process Mining Algorithms with a Tailored Library: Design Principles and Performance Assessment", National Research University Higher School of Economics, Tech. Rep., 2018. DOI: 10.13140/RG.2.2.18320.46084. [Online]. Available: https://www.researchgate.net/publication/332869308_Enhancing_Efficiency_of_Process_Mining_Algorithms_with_a_Tailored_Library_Design_Principles_and_Performance_Assessment_Technical_Report.

[32] S. A. Shershakov, A. A. Kalenkova, and I. A. Lomazova, "Transition Systems Reduction: Balancing Between Precision and Simplicity", in *Transactions on Petri Nets and Other Models of Concurrency XII*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 119–139, ISBN: 978-3-662-55862-1. DOI: 10.1007/978-3-662-55862-1_6. [Online]. Available: https://doi.org/10.1007/978-3-662-55862-1_6.

[33] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, "Deriving Petri Nets from Finite Transition Systems", *IEEE Trans. Comput.*, vol. 47, no. 8, pp. 859–882, 1998, ISSN: 0018-9340. DOI: 10.1109/12.707587. [Online]. Available: http://dx.doi.org/10.1109/12.707587.

[34] W. M. P. Van Der Aalst, V. Rubin, H. M. W. Verbeek, B. F. Dongen, E. Kindler, and C. W. Günther, "Process mining: a two-step approach to balance between underfitting and overfitting", *Software & Systems Modeling*, vol. 9, no. 1, pp. 87–111, 2010, ISSN: 1619-1374. [Online]. Available: http://dx.doi.org/10.1007/s10270-008-0106-z.