УДК 004.65 **DOI** 10.15622/ia.20.4.7

В.В. Миронов, А.С. Гусаренко, Г.А. Тугузбаев

ИЗВЛЕЧЕНИЕ СЕМАНТИЧЕСКОЙ ИНФОРМАЦИИ ИЗ ГРАФИЧЕСКИХ СХЕМ

Миронов В.В., Гусаренко А.С., Тугузбаев Г.А. **Извлечение семантической информации из графических схем.**

Аннотация. Рассматривается задача извлечения семантической информации из электронного документа, заданного в формате векторной графики и содержащего графическую модель (схему), построенную с помощью графического редактора. Задача состоит в программном извлечении определенных структурных и параметрических свойств схемы и занесении их в базу данных для последующего использования. На основе проведенного анализа возможностей графических редакторов сделан вывод об актуальности этой задачи для универсальных редакторов, не привязанных к конкретным графическим нотациям и использующих открытые графические форматы документов, что допускает программную обработку. Предлагаемый подход рассматривает графические документы на трёх уровнях абстракции: концептуальном (семантические свойства схемы), логическом (представление семантических свойств на внутреннем уровне документа) и физическом (внутренняя организация графического документа). Решение задачи основано на построении концептуально-логического отображения, то есть отображения концептуальной модели схемы в логическую модель графического документа с учетом его физической модели. В рамках подхода разработан алгоритм построения указанного отображения, представленный в виде объектно-ориентированного псевдокода. Исследование внутренней разметки в открытых графических форматах позволило построить модели идентификации элементов схемы и их соединений между собой, что необходимо для конкретного применения алгоритма. Получены выражения для адресации элементов схемы и доступа к их свойствам. Предложенный подход реализован на основе ситуационно-ориентированной парадигмы, в рамках которой процесс извлечения управляется иерархической ситуационной моделью. Обрабатываемые данные задаются в ситуационной модели в виде виртуальных документов, отображаемых на разнородные внешние источники данных. Для решаемой задачи рассматривается отображение на два варианта форматов векторной графики: на «плоский» файл разметки и на набор таких файлов в электронном архиве. Практическое использование результатов иллюстрируется на примере извлечения семантической информации из графических моделей, разрабатываемых на различных этапах проектирования баз данных.

Ключевые слова: блок-схема, векторная графика, извлечения свойств, ситуационно-ориентированная парадигма, ситуационная модель, виртуальный документ.

1. Введение. В процессе разработки систем различного назначения широко применяются схемы (графические модели) различных видов (информационные, электрические, гидравлические, пневматические) и типов (структурные, функциональные, принципиальные и др.), которые с помощью условных графических обозначений показывают на том или ином уровне абстракции, какие компоненты содержит система и как они соотносятся между собой (ГОСТ 2.701–2008 ЕСКД. Схемы. Виды и типы. Общие требования к выполнению). В настоящее время схемы,

как правило, строятся разработчиками в виде электронных документов в формате векторной графики с помощью графических редакторов. В дальнейшем визуальное представление схемы используется как само по себе, так и в качестве основы для разработки других схем и иной конструкторской документации. В условиях стремления к автоматизации процессов разработки систем такие документы должны не только четко и ясно доносить до пользователей информацию в визуальной форме (презентационный аспект), но и предоставлять формальные спецификации системы, заложенные в схеме (семантический аспект).

В связи с этим возникает задача извлечения формальных спецификаций (семантической информации) из графического документа, которая рассматривается в данной статье. Ниже анализируется состояние области исследования, обсуждается задача извлечения и предлагается подход к ее решению. Предлагается модель, задающая отображение концептуальной модели схемы в логическую модель графического документа, и рассматривается общий алгоритм задания концептуальной модели. Обсуждается идентификация элементов схемы для различных графических форматов. Описывается реализация предложенного подхода на основе ситуационно-ориентированной парадигмы и применение результатов в учебном процессе.

2. Состояние области исследования. Графические редакторы, применяемые для построения графических моделей, можно разделить на специализированные и универсальные.

Специализированные редакторы ориентированы на создание схем (диаграмм) в узкой предметной области. Например, популярный редактор Erwin Data Modeler дает возможность построения ER-диаграмм баз данных в нотации IDEF1X, редактор IBM Rational Rose XDE предоставляет разработчику среду разработки программного обеспечения в нотации UML. Эти редакторы имеют специфическую функциональность, такую как генерация диаграмм из программного кода в сочетании с генерацией программного кода из диаграмм (roundtrip engineering). В доступных научных публикациях исследуется повышение надежности автоматизированных систем в ходе трансформации диаграмм за счет сегментирования модели [1]. Обсуждаются программные средства для автоматического создания визуальных композиций на основе диаграмм [2]. Рассматриваются инструментарии, допускающие использование информационных сообщений, встроенных в графику [3, 4]. Анализируются преимущества и недостатки наиболее популярных редакторов объектно-ориентированного моделирования [5].

В целом специализированные редакторы ориентированы на жестко заданные графические нотации (системы условных обозначений и правил записи схем) и предоставляют ограниченный набор сведений о свойствах схемы (таких, как статистика — количество элементов, связей и т. п.). С их помощью нельзя задавать схемы в непредусмотренной нотации и нельзя извлекать семантические свойства схемы, не предусмотренные исходной функциональностью.

Универсальные графические редакторы не ограничены какой-либо графической нотацией. Наборы фигур, позволяющие задавать условные графические обозначения в той или иной нотации, поставляются вместе с редактором или сторонними разработчиками, а также могут быть созданы самими пользователями. Популярными универсальными редакторами векторной графики являются Microsoft Office Visio (проприетарный), а также его свободно распространяемые аналоги OpenOffice Draw, LibreOffice Draw и др. [6]. В многочисленных публикациях отражается эффективное применение этих редакторов для создания схем в самых разных предметных областях: учебном процессе [7, 8], разработке и документировании программного обеспечения [9], моделировании бизнес-процессов [10], проектировании автоматизированных систем [11], разработке и анализе моделей рабочих процессов [12–14], интеграции баз данных [15–17], обработке запросов на естественном языке [18] и др.

Универсальные редакторы не привязаны к конкретным графическим нотациям, поэтому с их помощью можно задавать схемы в заранее не предусмотренной нотации. Другой их особенностью является использование открытых графических форматов для создания результирующих схем. Это дает принципиальную возможность автоматизированной обработки графических документов с целью извлечения семантических свойств из содержащихся в них схем. Вместе с тем, как показал анализ публикаций, вопросы извлечения свойств схемы практически не освещены в литературе (ни в техническом, ни в научном аспектах). Отсутствуют как общие принципы, так и практические аналоги решения этой задачи. Это обуславливает актуальность исследования по извлечению семантических свойств схемы применительно к универсальным графическим редакторам.

3. Задача извлечения. Целью исследования являлось достижение понимания того, как может быть извлечена семантическая информация из схемы, содержащейся в электронном документе, созданном в среде универсального графического редактора в формате векторной графики. Под семантической информацией здесь понимаются структурные и па-

раметрические свойства схемы, очищенные (абстрагированные) от несущественных деталей ее визуального представления, таких как размеры фигур, толщина линий, цвет заливки и т. п. Результатом решения задачи должен быть граф, вершины которого соответствуют элементам схемы, а дуги — соединениям элементов друг с другом. Вершины и дуги должны быть нагружены параметрами, заданными в схеме. В техническом аспекте задача состоит в программном извлечении интересующих структурных и параметрических свойств графической схемы и занесение их в базу данных для последующего использования. При этом следует различать два уровня представления информации в электронном графическом документе: внешний, соответствующий визуальному отображению схемы, и внутренний, соответствующий внутреннему кодированию элементов схемы согласно используемому графическому формату.

Идея решения задачи состоит в том, чтобы исследовать особенности задания условных обозначений на внутреннем уровне в различных форматах векторной графики и на этой основе научиться идентифицировать условные обозначения, примененные в конкретной схеме. Предлагаемый подход в общем виде иллюстрируется на рисунке 1.

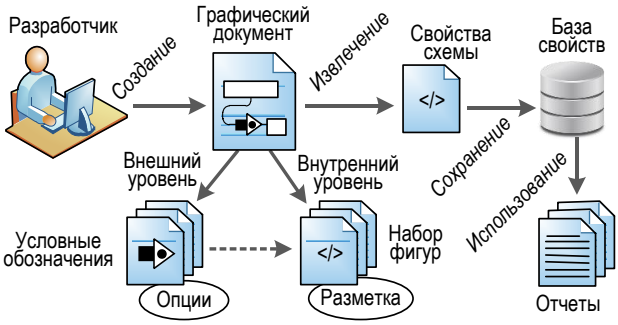


Рис. 1. Подход к извлечению семантической информации из электронного графического документа

Создание. Предполагается, что электронный документ содержит блок-схему, построенную в определенной графической нотации на основе определенного набора фигур-элементов (условных обозначений). Разработчику схемы в среде графического редактора (как специализированного, так и универсального) предоставляется набор фигур, задающих условные обозначения. Условным обозначениям на внешнем, визуальном уровне соответствует набор закодированных фигур изображения

на внутреннем уровне документа. При этом установленные разработчиком опции (допустимые модификации) условных обозначений определенным образом отражаются во внутреннем представлении (разметке) набора фигур. Использование стандартного набора фигур на внешнем, визуальном уровне дает возможность идентифицировать как сами фигуры, так и их опции при последующей программной обработке документа на внутреннем уровне. Поэтому возникает важный вопрос о том, каким образом условные обозначения, используемые для визуального представления схемы, реализуются на внутреннем уровне графического документа.

Извлечение. Возможность программной обработки графического документа с целью извлечения из него свойств обусловлена применением в универсальных графических редакторах открытых форматов. Открытые графические форматы, как правило, основаны на использовании XML-разметки и двух способов организации:

• плоский формат, при котором графический документ представляет собой несжатый XML-файл. Например, формат VDX (Visio Drawing based on XML) представляет XML-файл графического редактора Visio на языке разметки DataDiagramML. Этот формат применяется в версии Microsoft Visio 2010, он достаточно распространен, хотя в поздних версиях Visio не поддерживается. Или формат FODG (Flat ODG), применяемый в редакторе LibreOffice Draw — свободно распространяемом аналоге редактора Visio;

• иерархический формат, продвинутый, при котором графический документ организован в виде сжатой в ZIP-архиве иерархии вложенных папок, содержащих XML-файлы компонентов документа. Например, формат VSDX в редакторе Visio, соответствующий стандарту Open Packaging Conventions и применяемый в версиях Visio 2013 и более поздних. Или формат ODG, соответствующий стандарту Open Document Format и применяемый в свободно распространяемых редакторах Apache OpenOffice Draw и LibreOffice Draw.

Сохранение. Результаты извлечения необходимо сохранить в базе данных в виде, удобном для дальнейшего использования. Извлечение предполагает абстрагирование, то есть получение из графического документа информации о существенных структурно-параметрических свойствах схемы, игнорируя при этом несущественные стилистические и эстетические особенности графического изображения. К структурно-параметрическим свойствам схемы, которые нужно выявить в результате извлечения, относятся:

• элементы — экземпляры условных графических обозначений, использованные в схеме;

- установленные опции и параметры каждого элемента схемы;
- соединение элементов между собой с помощью коннекторов.

Использование. Результаты извлечения свойств могут быть использованы для различных целей, таких как проверка корректности схемы, выявление ошибок; генерация проектной документации; генерация программного кода, соответствующего схеме; формирование персонализированных моделей для следующих этапов проектирования.

Отметим, что рассматриваемая задача извлечения ориентирована не на какую-то конкретную схему, а на множество схем определенного типа. То есть графическая схема здесь рассматривается как тип или класс, предполагающий множество конкретных экземпляров его реализации. Все экземпляры выполнены в одной графической нотации, то есть основываются на общей системе условных обозначений. При этом при построении экземпляров схемы применяется один и тот же набор фигур, реализующих условные обозначения. Экземпляры схемы отличаются только составом фигур, их опциями и параметрами, предусмотренными графической нотацией, а также соединениями фигур между собой с помощью коннекторов. Такие допущения позволяют единообразно обрабатывать экземпляры схемы в процессе извлечения свойств.

Отметим также, что задача извлечения может относиться не ко всей схеме, а к некоторой ее части. Она может касаться, к примеру, только определенных свойств или аспектов схемы и/или относиться только к определенным частям или элементам схемы.

4. Модель концептуально-логического отображения. В основе предлагаемого подхода используется рассмотрение графического документа на трёх уровнях абстракции: концептуальном (семантические свойства схемы), логическом (представление семантических свойств на внутреннем уровне документа) и физическом (внутренняя организация графического документа). Решение задачи основано на построении концептуально-логического отображения, то есть построении модели, задающей отображение концептуальной модели схемы в логическую модель графического документа с учетом его физической модели (рис. 2).

Концептуальная модель. Под концептуальной моделью схемы нами понимается модель верхнего уровня абстракции, задающая набор элементов, их свойства, а также отношения друг с другом. Каждый элемент схемы — это экземпляр условного графического изображения. Концептуальная модель задает, какие элементы присутствуют на схеме; каковы свойства (опции, параметры) этих экземпляров; как экземпляры вложены друг в друга; как экземпляры соединены между собой. На ри-

сунке 2 справа концептуальная модель иллюстрируется с помощью дерева, где корень Schema содержит множество дочерних элементов (Element) и множество дочерних соединений (Connect), которые, в свою очередь, содержат множества свойств (Property). Отношение вложенности одного элемента в другой задается необязательной ссылкой на родителя (Parent). Отношение соединения одного элемента с другим задается парой ссылок на эти элементы (From и To), которые содержатся в соединении. Таким образом, концептуальная модель отражает структуру того, что должно быть получено в результате извлечения.

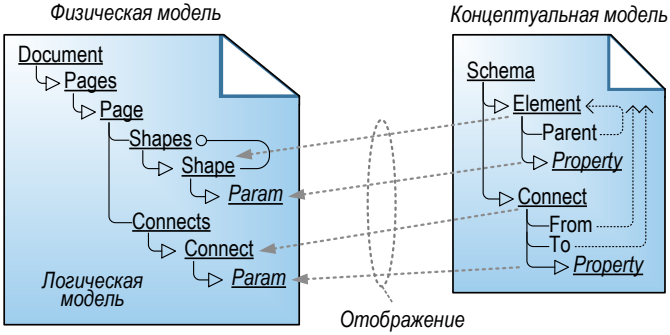


Рис. 2. Концептуально-логическое отображение

Физическая модель. Физическая модель в нашем понимании — это общая модель графических документов с учетом используемого графического формата и используемого набора фигур условных обозначений. Одному и тому же внешнему (визуальному) представлению схемы соответствуют разные физические модели при использовании разных графических форматов, разных наборов фигур для представления элементов (условных обозначений), а также разного визуального представления и размещения фигур на листах документа. Экземплярами этой модели являются внутренние представления (в виде XML-деревьев) конкретных графических документов, построенных с использованием определенного набора фигур, представляющих элементы определенной графической нотации. Таким образом, физическая модель соответствует исходному документу, содержащему схему, которую нужно подвергнуть извлечению.

Логическая модель. Под логической моделью здесь понимается та часть физической модели, которая имеет отношение к элементам

и свойствам схемы, отраженным в концептуальной модели. В логической модели опущены свойства, задающие внешнее, визуальное представление схемы (такие как размеры элементов, толщина линий, координаты положения на листе, цвет, заливка и др., если только они не существенны с точки зрения концептуальной модели). Как правило, в логической модели игнорируется значительное количество свойств внешнего визуального представления документа, поэтому логические модели существенно проще физических. Однако и они зависят от графического формата и способа задания условных обозначений. На рисунке 2 слева логическая модель иллюстрируется с помощью дерева, являющегося XML-поддеревом физической модели. Корневой XML-элемент Document содержит дочерний XML-элемент Pages, который, в свою очередь, включает множество дочерних XML-элементов Page, задающих изображение на отдельных листах графического документа (префикс XML введен, чтобы отличать понятие «элемента» в языке XML от элемента схемы). Вложенный XML-элемент Shapes включает множество дочерних XML-элементов Shape, задающих отдельные фигуры изображения. Фигуры могут быть сложными, состоящими из других фигур. В этом случае применяется рекурсивная вложенность: внутри XML-элемента Shape размещается XML-элемент Shapes с вложенными Shape, задающими детализацию. Соединения фигур на листе задаются с помощью XML-элемента Connects, включающего множество дочерних XML-элементов Connect, каждый из которых определяет пару соединенных фигур Shape, которые ранее уже определены. Таким образом, логическая модель отражает те особенности исходного графического документа, которые существенны в задаче извлечения.

Концептуально-логическое отображение. Для решения задачи необходимо построить концептуально-логическое отображение, которое ставит в соответствие объектам концептуальной модели (элементам, соединениям и их свойствам) объекты логической модели (а также физической модели, поскольку логическая модель есть часть физической). Тем самым концептуально-логическое отображение дает понимание того, где в графическом документе находится XML-разметка фигур, составляющих тот или иной элемент схемы, и как параметры этой разметки задают свойства (опции и параметры) элемента схемы.

На рисунке 3 представлена модель «сущность–связь», задающая основные типы сущностей, участвующих в задаче извлечения, (фигуры-прямоугольники) и бинарные связи между ними (фигуры-стрелки).

Связи указывают направление от сущности-родителя к сущности-ребенку с кардинальностью «0, 1, M» (треугольник — связь «ко

многим») или «0, 1» (трапеция — условная связь). Кружок внутри треугольника задает кардинальность связи в направлении от ребенка к родителю: темный — «1, 1» (недопустимы «сироты» — каждый ребенок должен иметь родителя); светлый — «0,1» (допустимы «сироты» — ребенок может не иметь родителя). Темный квадратик обозначает идентифицирующую связь (когда идентификатор родителя входит в состав идентификатора ребенка), а светлый — неидентифицирующую (когда ребенок идентифицируется независимо от родителя).

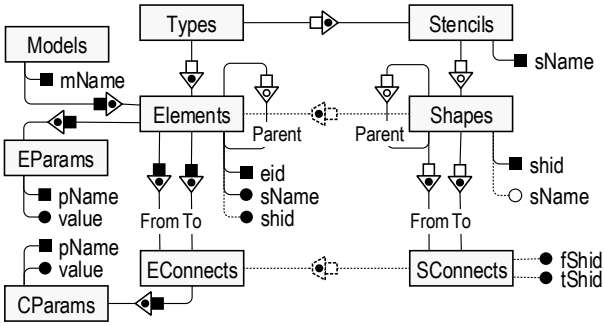


Рис. 3. Взаимосвязь сущностей на концептуальном (слева) и логическом/физическом (справа) уровнях

Графическая нотация. Сущность типа `Types` задает множество типов элементов, то есть типов условных графических изображений графической нотации, применяемой в схеме. Каждый экземпляр этой сущности соответствует конкретному типу. Множество экземпляров `Types` для задачи извлечения известно априори.

Сущность типа `Stencils` задает множество трафаретов (шаблонов, образцов) фигур, соответствующих условным графическим обозначениям, которые используются разработчиком для построения схемы путем копирования на лист графического документа в среде графического редактора. Наборы трафаретов могут существовать в различных видах: в виде фигур, хранящихся в отдельном графическом документе, или (если позволяет функциональность графического редактора) в виде специальных наборов фигур, которые могут прикрепляться к графическому документу и использоваться для его построения. Каждому экземпляру сущности `Stencils` на внутреннем уровне соответствует некоторое дерево XML-разметки. Экземпляр трафарета идентифицируется ключевым атрибутом `sName` («имя трафарета») и относится к определенному типу, то есть ему соответствует один и только один экземпляр

сущности Types. Вместе с тем возможно несколько трафаретов одного типа, например, которые отличаются визуальным стилем исполнения или конкретными значениями параметров соответствующего условного графического обозначения.

Графический документ. Сущность типа Shapes задает множество фигур физической модели схемы. Каждый экземпляр этой сущности соответствует конкретной фигуре, присутствующей в графическом документе. На внутреннем уровне экземпляр соответствует поддереву XML-разметки XML-дерева документа. Экземпляр фигуры идентифицируется ключевым атрибутом shid («идентификатор фигуры»). Каждому экземпляру фигуры, если он получен путем копирования некоторого трафарета, соответствует один экземпляр сущности Stencils, этот факт отражен с помощью виртуального атрибута sName, значение которого копируется из сущности Stencils. Вместе с тем в графическом документе возможны фигуры, не относящиеся к какому-либо трафарету. Рекурсивная связь Parent задает вложенность фигур — каждый экземпляр сущности Shapes может иметь (или не иметь) родителя — экземпляр сущности Shapes, в который он вложен; при этом каждый экземпляр сущности Shapes может иметь ноль или несколько вложенных дочерних экземпляров Shapes.

Сущность типа SConnects задает множество пар соединенных друг с другом фигур, то есть каждому экземпляру этой сущности соответствует пара экземпляров сущности Shapes: один экземпляр, являющийся первым (From), и один экземпляр, являющийся вторым (To) компонентом пары. Из экземпляра соединения доступны, таким образом, идентификаторы двух родителей, представленные на рисунке в виде виртуальных атрибутов fShid и tShid. При этом каждая фигура может участвовать в ноль или нескольких экземплярах соединений в качестве как первого, так и второго компонента.

Результирующая концептуальная модель. Сущность типа Models задает множество концептуальных моделей схемы. В общем случае из одного графического документа можно получить несколько концептуальных моделей, отличающихся детальностью или полнотой извлекаемых сведений. Экземпляры этой сущности идентифицируются ключевым атрибутом mName («имя модели»).

Сущность типа Elements задает множество элементов концептуальной модели схемы. Каждый экземпляр этой сущности соответствует конкретному элементу, присутствующему на схеме, и принадлежит определенной модели (экземпляру сущности Models). Экземпляры этой

сущности идентифицируются ключевым атрибутом `eid` («идентификатор элемента»). Атрибут `sName` задает имя трафарета, использованного при задании соответствующей фигуры в графическом документе, а атрибут `shid` задает идентификатор самой фигуры. Элементы схемы относятся к определенному типу, то есть каждому экземпляру сущности `Elements` соответствует один и только один экземпляр сущности `Types`. Рекурсивная связь `Parent` задает вложенность элементов — каждый экземпляр сущности `Elements` может иметь (или не иметь) родителя — экземпляр сущности `Elements`, в который он вложен; с другой стороны, каждый экземпляр сущности `Elements` может иметь ноль или несколько вложенных дочерних экземпляров того же типа. Множество экземпляров сущности `Elements` должно быть построено в результате решения задачи извлечения.

Сущность типа `EParams` задает множество параметров элементов схемы. Каждый экземпляр этой сущности соответствует конкретному параметру конкретного элемента, присутствующего на схеме. Экземпляры этой сущности в пределах одного родительского элемента идентифицируются ключевым атрибутом `pName` («имя параметра»). Атрибут `value` задает значение параметра. Множество экземпляров `EParams` должно быть построено в результате решения задачи извлечения.

Сущность типа `EConnects` задает множество пар соединенных друг с другом элементов схемы, то есть каждому экземпляру этой сущности соответствует пара экземпляров сущности `Elements`: один экземпляр, являющийся первым (`From`), и один экземпляр, являющийся вторым (`To`) компонентом пары. Экземпляры этой сущности идентифицируются парой ключевых атрибутов родителей через связи `From` и `To`. При этом каждый элемент может участвовать в ноль или нескольких экземплярах соединений в качестве как первого, так и второго компонента. Каждому экземпляру сущности `EConnects` соответствует экземпляр сущности `SConnects`, но не наоборот. Множество экземпляров сущности `EConnects`, как и множество экземпляров сущности `Elements`, должно быть построено в результате решения задачи извлечения.

Сущность типа `CParams` задает множество параметров соединений схемы. Каждый экземпляр этой сущности соответствует конкретному параметру конкретного соединения из множества `EConnects`. Экземпляры этой сущности в пределах одного родительского соединения идентифицируются ключевым атрибутом `pName` («имя параметра»). Атрибут `value` задает значение параметра. Множество экземпляров сущности `CParams` должно быть построено в результате решения задачи извлечения.

Допущения. Отметим существенное допущение, принятое в этой модели, а именно, предположение, что каждому элементу схемы соответствует некоторая (единственная) фигура внутреннего представления. На рисунке 3 это обстоятельство отмечено темным кружком на символе условной связи `Shapes-Elements` (эта связь изображена пунктиром, поскольку она только подразумевается, ее реализация не требуется явно). Условные графические изображения сами по себе могут быть сложными, состоящими из множества графических примитивов, их приходится рисовать средствами графического редактора на основе сочетания нескольких фигур. Указанное допущение требует, чтобы в этом случае все фигуры-компоненты, составляющие элемент схемы как условное графическое обозначение, были заключены в единую фигуру-контейнер, представляющую этот элемент. Разработчик схемы должен не просто нарисовать элемент схемы в соответствии с визуальными правилами выполнения условных графических изображений, но оформить его в виде единой внутренней фигуры. Графические редакторы предоставляют для этого возможность группирования нескольких фигур изображения в единую фигуру-контейнер.

Другое допущение связано с соединением фигур. Разработчик должен не просто зрительно разместить рядом соединенные элементы, но закрепить это соединение так, чтобы этот факт отразился особым образом во внутреннем представлении графического документа, мог быть обнаружен в ходе программной обработки и зафиксирован в концептуальной модели в виде соответствующего экземпляра сущности `EConnects`. Графические редакторы предоставляют для этого возможность размещения в фигурах точек соединения, линий-коннекторов, а также предусматривают специальный механизм фиксации соединений.

5. Общий алгоритм задания концептуальной модели. Итак, множества экземпляров сущностей `Types` и `Stencils` заданы априори для используемой графической нотации. Множество экземпляров сущностей `Shapes` и `SConnects` задано в XML-разметке обрабатываемого графического документа. Множество экземпляров сущностей `Elements` и `EConnects` требуется определить в результате извлечения на основе анализа экземпляров сущностей `Shapes` и `SConnects`. Для этого должна быть возможность идентифицировать фигуры графического документа на предмет их соответствия элементам схемы определенного типа, а также идентифицировать параметры элементов, основываясь на свойствах XML-разметки соответствующих фигур. В соответствии с этим был разработан алгоритм задания результирующей концептуальной модели схемы, объектно-ориентированный псевдокод представлен ниже в листинге 1.

```
000 Begin
001   m = New Model ('MName')
002   ForEach s: s ∈ Shapes, s.sName ∈ StencilNames
003      e = m.AddElement (s.shid)
004      e.AddType (EType (s.sName))
005      ForEach ep: ep ∈ EParamNames (s.cName)
006         e.AddEParam (p, s.pvalue (p))
007   ForEach c: c ∈ SConnects
008      For fe: fe ∈ m.Elements, fe.shid = c.fShid
009         For te: te ∈ m.Elements, te.shid = c.tShid
010            cm = m.AddEConnect (fe, te)
011            ForEach cp: cp ∈ CParamNames (fe.sName, te.sName)
012               cm.AddCParam (cp, c.pvalue (cp))
013 End
```

Листинг 1. Псевдокод алгоритма задания концептуальной модели

Строки `000` и `013` задают контейнер для кода в целом. В строке `001` создается новая (пока еще пустая) результирующая модель в виде объекта `m`; имя модели задается с помощью параметра `'MName'`.

Фрагмент `002`–`006` обрабатывает фигуры графического документа и заполняет модель обнаруженными элементами и их параметрами. Конструкция `ForEach` (строка `002`) перебирает все фигуры графического документа, отбирая те из них, которые основаны на каком-либо трафарете используемой графической нотации. В качестве критерия проверяется принадлежность атрибута `sName` фигуры множеству имен используемых трафаретов `StencilNames`. Далее для каждой отобранной фигуры `s` выполняется следующее:

- (строка `003`) в модель `m` добавляется новый элемент `e`, соответствующий фигуре `s`, идентификатор фигуры `shid` передается в качестве параметра;
- (строка `004`) для элемента `e` устанавливается ссылка на его тип, для чего используется функция `EType`, отображающая имена шаблонов в типы элементов;
- (строки `005`–`006`) формируется список параметров элемента `e`, для чего перебираются имена параметров `EParamNames`, предусмотренных для трафарета `s.sName` элемента `e`, и с помощью метода `AddEParam`

в объект e добавляются новые объекты-параметры с заданными именами pName и значениями value. Значение параметра формируется с помощью функции pvalue на основе XML-разметки фигуры s;

Фрагмент 007–012 обрабатывает соединения фигур графического документа и заполняет модель соединениями элементов и параметрами соединений. Конструкция ForEach (строка 007) перебирает все соединения фигур графического документа. Следующие конструкции For (строки 008 и 009) отбирает те соединения, которые относятся к фигурам-элементам и отыскивает соответствующую пару элементов fe (первый) и te (второй). Далее для каждой отобранной пары элементов выполняется следующее:

• (строка 010) в модель m добавляется новое соединение элементов cm;

• (строки 011–012) формируется список параметров соединения элементов fe и te, для чего перебираются имена параметров CParamNames, предусмотренных для соответствующей пары трафаретов, и с помощью метода AddCParam в объект cm добавляются новые объекты-параметры с заданными именами pName и значениями value. Значение параметра формируется с помощью функции pvalue на основе XML-разметки соединения cp.

6. Идентификация элементов схемы. При обработке графического документа согласно рассмотренному алгоритму, необходимо выполнить перебор составляющих его фигур с целью идентификации элементов схемы. Этот процесс зависит, во-первых, от используемого графического формата, во-вторых, от конкретного представления фигур условных графических обозначений. Таким образом, возникает задача построения моделей идентификации элементов схемы для конкретных графических форматов. Рассматриваемые ниже модели получены в результате непосредственного исследования внутренней XML-разметки графических документов для открытых форматов редакторов Visio, LibreOffice Draw и OpenOffice Draw.

Особенности идентификации трафарета фигуры. Для каждой обрабатываемой фигуры s необходимо узнать значение s.sName, то есть соответствует ли фигура одному из трафаретов и каково уникальное имя sName этого трафарета. Рассмотрим организацию именования трафаретов и фигур в различных графических форматах.

Форматы VDX/VSDX (графический редактор Visio). В этих форматах сложные фигуры существуют в документе в двух формах: 1) в форме так называемого мастера (Master — «хозяин») — исходного

трафарета фигуры; 2) в форме фигуры-экземпляра (`Shape`), представляющего конкретный экземпляр мастера-трафарета, который размещен на листе со своими индивидуальными настройками. Один мастер-трафарет может служить основой для множества фигур-экземпляров. Имя мастера задается разработчиком при визуальном создании фигуры и сохраняется во внутреннем XML-представлении мастера (атрибут `NameU` — «универсальное имя» — у XML-элемента `Master`). Внутренняя XML-разметка фигуры-экземпляра не всегда содержит атрибут `NameU`, но всегда содержит ссылку на свой мастер-трафарет, если фигура была порождена на основе трафарета. Поэтому имя трафарета фигуры может быть идентифицировано по имени `NameU` его мастера. Реализация этого подхода различается для форматов VDX и VSDX.

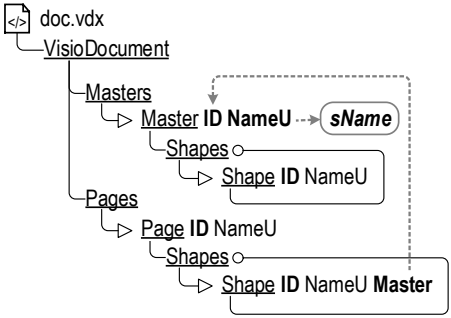


Рис. 4. Извлечение имени трафарета фигуры в формате VDX

В первом случае мастера размещаются в том же XML-файле, что и порожденные из них фигуры (рис. 4). Мастера размещаются в элементах `/VisioDocument/Masters/Master` и идентифицируются уникальным XML-атрибутом `ID`. Фигуры размещаются в элементах `/VisioDocument/Pages/Page/Shapes/Shape` и имеют XML-атрибут `Master`, содержащий значение идентификатора мастера. Таким образом, чтобы получить имя трафарета, нужно извлечь значение атрибута `NameU` у мастера, идентификатор которого совпадает со значением атрибута `Master` обрабатываемой фигуры. На языке XPath (языке адресации узлов в XML-деревьях) это будет выглядеть, например, так:

```
//Master[@ID = ./@Master]/@NameU.
```

При этом предполагается, что контекстным узлом, обозначенным как «.», является обрабатываемая фигура `Shape`.

Во втором случае список мастеров, их описания, а также содержимое отдельных листов документа размещено в отдельных XML-файлах в составе ZIP-архива (рис. 5).

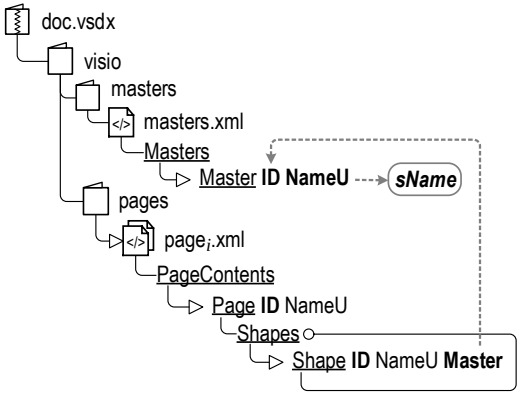


Рис. 5. Получение имени трафарета фигуры в формате VSDX

Здесь общие сведения о мастере, в том числе и необходимый нам атрибут `NameU`, находятся в элементах `/Masters/Master` файла `masters.xml`, который размещен в папке `masters` ZIP-архива графического документа. Сведения о фигуре находятся в элементах `/PageContents/Page/Shapes/Shape` файла `page`*i*`.xml`, который соответствует *i*-му листу графического документа (размещен в папке `Pages`). Таким образом, извлечение имени трафарета в этом случает требует совместной обработки двух XML-файлов.

Формат ODG (графические редакторы LibreOffice / OpenOffice Draw). В этом формате, как и во многих других, используемых в графических редакторах — аналогах Visio, внутреннее представление графического документа не предусматривает отдельного сохранения трафарета-мастера — исходного представления скопированного изображения. XML-разметка фигуры просто копируется в список фигур страницы документа. Однако в составе внутреннего XML-представления скопированных фигур присутствует необязательный XML-элемент `svg:title` (рис. 6), который может быть задан разработчиком при создании исходного изображения фигуры-трафарета и который наследуется во всех его копиях. Этот XML-элемент может быть использован в качестве имени трафарета.

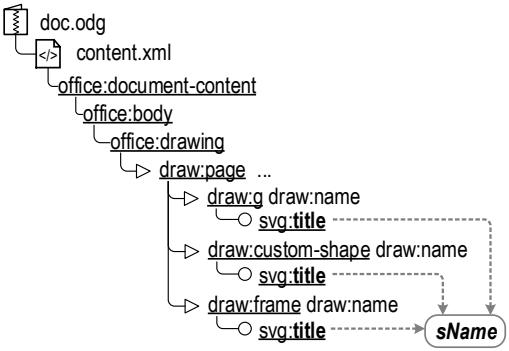


Рис. 6. Получение имени трафарета фигуры в формате ODG

В разметке листа `draw:page` графического документа формата ODG конкретные фигуры могут быть представлены различными XML-элементами: `draw:g`, `draw:custom-shape`, `draw:frame` (рис. 6). Фигуры идентифицируются уникальными именами `draw:name`, а вложенные XML-элементы `svg:title` содержат искомое имя трафарета. Таким образом, в этом случае имя трафарета фигуры содержится в XML-разметке самой фигуры, что облегчает идентификацию. Соответствующее XPath-выражение выглядит так:

```
./svg:title .
```

Для сравнения на рисунке 7 приведена аналогичная модель для документа в формате GraphML, который применяется, в частности, в популярном редакторе yEd Graph Editor.

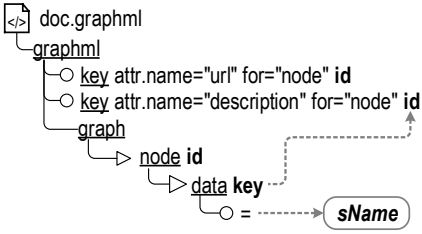


Рис. 7. Получение имени трафарета фигуры в формате GraphML

Здесь отдельные фигуры заданы с помощью XML-элементов `/graphml/graph/node`, а их содержимое задается с помощью вложенных XML-элементов `data`. Каждый элемент `data` ссылается на XML-элемент

key, поясняющий его назначение. В частности, элементы key с атрибутами attr.name = "url" или attr.name = "description", соответствуют элементам data, содержащим пользовательские данные. Их можно использовать для задания имени фигуры-трафарета. Так, если имя трафарета задано разработчиком в поле description, то его можно извлечь как текстовое содержимое соответствующего элемента data, например, с помощью следующего XPath-выражения:

```
./data[@key=//key[@attr.name="description"]/@id] .
```

Таким образом, идентификация трафарета фигуры существенно зависит от используемого графического формата.

Особенности идентификации соединений. Идентификация соединений требует выяснить на основе анализа соединений фигур графического документа, какие элементы схемы, выявленные и внесенные в концептуальную модель ранее, соединены между собой. Это несложно сделать при совместной обработке графического документа и концептуальной модели, поскольку внесенные элементы схемы содержат ссылки на соответствующие им фигуры графического документа (рис. 4, листинг 1). Однако детали зависят от используемого графического формата (рис. 8).

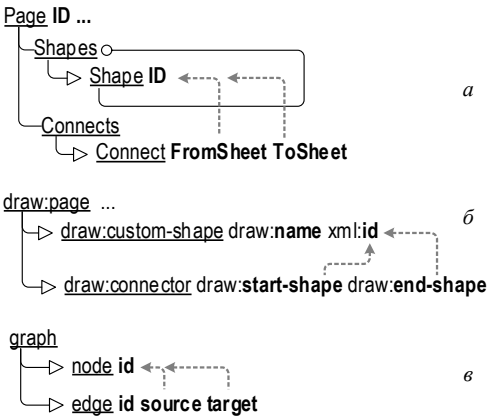


Рис. 8. Организация соединения фигур в различных форматах: а - VDX/VSDX, б - ODG, в - GraphML

Форматы VDX/VSDX. В этих форматах каждое соединение пары фигур отражается в XML-элементе Connects/Connect, который вложен в XML-элемент Page. Атрибуты FromSheet и ToSheet XML-элемента

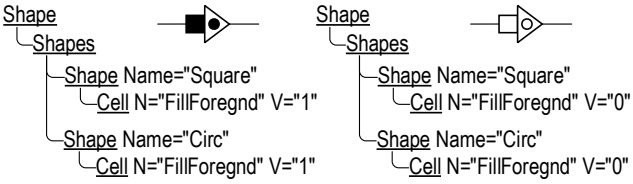
`Connect` ссылаются на идентификаторы первой и второй соединяемых фигур `Shapes/Shape` (рис. 8, *a*).

Форматы ODG и GraphML. В этих форматах каждое соединение отражается с помощью соответствующих атрибутов в фигурах-коннекторах. В формате ODG каждый коннектор задается XML-элементом `draw:connector`, вложенным в XML-элемент `draw:page`. Атрибуты `draw:start-shape` и `draw:end-shape` XML-элемента `draw:connector` ссылаются на идентификаторы первой и второй фигур, соединяемых данным коннектором (рис. 8, *b*). В формате GraphML каждый коннектор задается XML-элементом `edge`, вложенным в XML-элемент `graph`. Атрибуты `source` и `target` XML-элемента `edge` ссылаются на идентификаторы первой и второй фигур, соединяемых коннектором (рис. 8, *c*).

Идентификация параметров элемента схемы. Идентификация параметров элемента должна ответить на вопрос о том, какие особенности имеет данный элемент схемы как конкретный экземпляр условного графического обозначения определенного типа. Идентификация параметров выполняется путем исследования внутреннего представления фигуры, соответствующей элементу. Поскольку состав параметров определяется используемым трафаретом элемента, идентификация параметров зависит от результата предшествующей идентификации имени трафарета элемента, то есть для каждого трафарета элемента используется свой алгоритм идентификации того или иного параметра. Значение определенного параметра элемента определяется особенностями внутренней XML-разметки фигуры этого элемента — текстовыми значениями определенных XML-атрибутов или вложенных XML-элементов или их сочетаний. Для определения значения параметра строится XPath-выражение, которое адресует соответствующие XML-атрибуты или элементы, позволяя извлекать их значения. Аналогичным образом идентификация параметров соединения выполняется путем исследования внутреннего представления соединения, а также внутреннего представления соединяемых фигур. Для концептуальной модели схемы может быть существенным то, к каким частям фигур или к каким точкам соединения прикреплены коннекторы.

Пример. В качестве примера на рисунке 9 приведены модели XML-разметки двух вариантов задания элемента «связь», используемого при задании моделей базы данных на концептуальном уровне абстракции (модель «сущность–связь») в так называемой «уфимской» нотации. Символ связи представляет собой сочетание фигур-компонентов: квадрата, треугольника, круга и двух коннекторов. Параметры связи отображаются цветом заливки фигур-компонентов. Например, темная заливка квадрата означает идентифицирующую связь, а светлая

— не идентифицирующую; темная заливка круга означает обязательную связь от ребенка к родителю, а светлая — необязательную. Внутренняя разметка `Shape` фигуры «связь» в формате VSDX содержит разметку отдельных фигур-компонентов `Shape`, вложенных в контейнер `Shapes`.



а) *б)*

Рис. 9. Пример задания параметров условного графического обозначения «связь» в формате VSDX: а) идентифицирующая обязательная; б) неидентифицирующая необязательная

На рисунке 9 для примера представлена разметка двух фигур-компонентов: квадрата с атрибутом `Name="Square"` и круга с атрибутом `Name="Circ"`. Заливка фигур в формате VSDX задается с помощью XML-элементов `Cell` с атрибутом `N="FillForegnd"`, при этом значение цвета заливки задается атрибутом `V`. Таким образом, значение цвета заливки квадрата можно получить с помощью следующего XPath-выражения:

```
./Shapes/Shape[@Name="Square"]/Cell[@N="FillForegnd"]/@V ,
```

где предполагается, что контекстным узлом, обозначенным как «.», является обрабатываемая фигура «связь» `Shape`. Соответственно, значение цвета заливки круга можно получить с помощью XPath-выражения

```
./Shapes/Shape[@Name="Circ"]/Cell[@N="FillForegnd"]/@V .
```

7. Реализация на основе ситуационно-ориентированной парадигмы. Исследование процесса извлечения, рассмотренное выше, имело в качестве побочной цели развитие ситуационно-ориентированного подхода в плане распространения его на обработку документов векторной графики. Ситуационно-ориентированный подход исследуется авторами в рамках проекта интеграции разнородных данных на основе иерархической ситуационной модели (HSM — Hierarchical Situation Model) [18]. HSM определяет функционирование ситуационно-ориентированной базы данных на основе мониторинга текущей ситуации и сохранения текущего состояния ситуации между сеансами интерпретации. Архитектура HSM-модели иллюстрируется на рисунке 10.

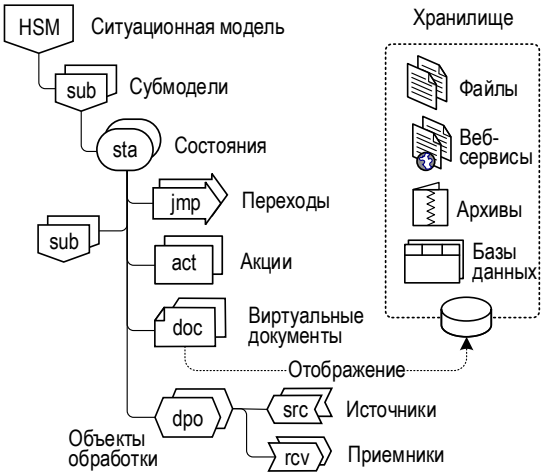


Рис. 10. Архитектура HSM

Субмодели `sub`, составляющие модель HSM, образуют иерархию — каждое состояние `sta` в субмодели в свою очередь может содержать вложенные субмодели. Элементы `jmp` задают переходы состояний, а элементы `act` — действия, ассоциированные с состояниями. Для рассматриваемой задачи существенными являются разновидности акций `doc` (виртуальные документы) и `dpo` (объекты обработки). В процессе сеанса интерпретации HSM контролируются текущие состояния субмоделей и выполняются ассоциированные акции, в том числе задающие обработку внешних данных. Для доступа к внешним данным применяются виртуальные документы `doc`, которые могут отображаться на внешние данные гетерогенной природы, исследованные в работах [18–20] (рис. 10): локальные и удаленные файлы, веб-сервисы, архивы, реляционные базы данных. Для обработки данных в объектах `dpo` предусмотрены элементы `src` (источники), задающие загрузку внешних данных, и элементы `rcv` (приемники), обеспечивающие вывод результатов. Эти элементы включают спецификации обработки данных.

В рассматриваемой задаче для нас важна возможность задавать в HSM отображение виртуальных документов на графические документы в различных открытых графических форматах. В работе [21] авторы рассматривали подобное отображение с целью генерации персо-

нализированных заготовок графических документов; в этой статье задача рассматривается в аспекте извлечения концептуальных свойств графического изображения.

Возможность отображения виртуальных документов на открытые графические форматы обусловлена тем, что все они в основе используют XML-разметку и организованы либо в виде плоского XML-файла, либо в виде архива XML-файлов. В первом случае следует использовать возможность отображения виртуального документа на «плоский» XML-файл, а во втором — на ZIP-архив XML-файлов. Например, HSM-декларация

```
<doc:VdxDocument type = "xml" path = "vdx/doc123.vdx/>"
```

задает отображение виртуального документа `doc:VdxDocument` целиком на VDX-документ `doc123.vdx`, находящийся в папке `vdx`. Декларация

```
<doc:VsdxDocument type = "zip" path = "vdx/doc456.vsdx">
    <ent:Master1 path = "visio/masters/master1.xml"/>
    <ent:Page1 path = "visio/pages/page1.xml"/>
</doc:VsdxDocument>
```

задает отображение виртуального документа `doc:VsdxDocument` на два компонента VSDX-документа `doc456.vdx`, находящегося в папке `vsdx`. Отображение интерпретирует целевой документ как ZIP-архив. Компонент `ent:Master1` указывает в архиве на файл мастеров `master1.xml`, а компонент `ent:Page1` — на разметку первой страницы документа `page1.xml`.

Обработка файлов XML выполняется с помощью объектов обработки DOM (Document Object Model), предоставляющих стандартизованный интерфейс доступа к XML-дереву. Загрузка в DOM-объект документа в формате единого XML-файла выполняется с помощью, например, такой HSM-директивы:

```
<dom:VdxProc srcDoc = "VdxDocument">
    <rcv:VdxExtract method="xslt" saveTo="dom:Extr" />
</dom:VdxProc>
```

Здесь создается DOM-объект `dom:VdxProc` и в него загружается XML-файл из виртуального документа `doc:VdxDocument`. заданного ранее. Далее с помощью приемника `rcv:VdxExtract` выполняется обработка загруженного файла и сохранение результата в другом DOM-объекте — `dom:Extr`. Атрибут `method="xslt"` указывает на то, что обработка (извлечение) выполняется путем XSL-трансформации на основе таблицы

стилей VdxExtract.xsl. При использовании формата на основе ZIP-архива HSM-директива может быть, например, такой:

```
<dom:VsdxProc srcDoc = "VsdxDocument.Page1">
    <src:Master1 srcDoc = "VsdxDocument.Master1"/>
    <rcv:VsdxExtract method="xslt" saveTo="dom:Extr"/>
</dom:VsdxProc>
```

Здесь в DOM-объект загружается XML-разметка первой страницы документа. После этого к ней с помощью источника src:Master1 подгружается XML-файл первого мастера, необходимый для выполнения извлечения.

Таким образом, сам процесс извлечения в иерархической модели не детализируется, указывается лишь метод его выполнения и ссылка на выполняющий преобразование компонент (в рассмотренном примере — на XSL-трансформатор). Компонент, выполняющий извлечение и построение концептуальной модели (в соответствии с рассмотренным выше алгоритмом, см. листинг 1, рис. 3), может быть реализован в виде таблицы стилей XSL-трансформации XML-документа (декларативный подход) или в виде модуля обработки, выполняющего манипуляции с деревом XML-документа через программный интерфейс DOM (процедурный подход).

Достоинства и недостатки подхода. Предложенный подход к извлечению семантической информации из графических документов обладает следующими *преимуществами*:

• Подход дает новую функциональность — возможность программным путем извлекать семантику схемы из ее графического представления, то есть формировать граф, отражающий элементы, соединения и существенные параметры схемы. Это может служить основой для автоматизированной проверки корректности схемы, генерирования технической документации и др.

• Подход применим к различным открытым графическим форматам на основе XML, которые предоставляют возможность программного доступа к внутреннему содержимому графических документов. Такие возможности имеются у большинства форматов, используемых в Visio-подобных графических редакторах.

• Экстракция схем не требует использования «родного» графического редактора, в среде которого получен графический документ. Эта особенность устраняет необходимость запуска графических редакторов или подключения их библиотек доступа на веб-сервере для программной обработки документов в веб.

• Простота реализации на основе ситуационно-ориентированной парадигмы. Реализованная в этой парадигме концепция виртуальных документов позволяет единообразно обрабатывать векторную графику в различных графических форматах и на этой основе генерировать результирующие документы — тоже в различных форматах.

Необходимо также отметить *ограничения и сложности* предложенного подхода. Данный подход применим только к открытым графическим форматам на основе XML, что снижает его универсальность. Другое ограничение в плане универсальности связано с допущениями об идентификации элементов схемы на основе трафаретов: подход работает, если при составлении схемы разработчик использовал для условленных графических обозначений заранее подготовленный набор фигур-трафаретов. Следует также отметить «хакерский» (в положительном смысле) оттенок подхода, затрудняющий его применение неквалифицированными пользователями. А именно, разработчик должен предварительно выполнить исследование внутренней структуры графической XML-разметки: во-первых, используемого графического формата, чтобы знать, как на внутреннем уровне задаются и идентифицируются фигуры и соединения графического изображения; во-вторых, набора трафаретов условных графических обозначений, используемых для построения схемы, чтобы знать, как на внутреннем уровне задаются их идентификационные и опциональные параметры. Только на базе этого можно правильно построить XPath-выражения для доступа к внутренним данным, которые необходимы для корректной работы алгоритма обработки графического документа и построения концептуальной модели схемы.

8. Практическое использование результатов. Рассмотренный выше подход был опробован и получил практическое применение в учебном процессе в Уфимском государственном авиационном техническом университете для извлечения свойств графических моделей в ходе курсового проектирования.

На рисунке 11 в качестве примера приведен фрагмент графического документа, содержащего схему — реляционную модель базы данных, которую студенты строят в среде графического редактора Visio на одном из этапов курсового проекта по дисциплине «базы данных». Здесь модель-диаграмма представлена в «уфимской» нотации: имена отношений (таблиц) представлены в прямоугольниках; имена атрибутов (столбцов) прикреплены к таблицам выносными линиями; концевые символы выносных линий задают свойства атрибутов; связи ссылочной целостности обозначены треугольниками, соединяющими атрибуты первичного

ключа родителя с атрибутами внешнего ключа ребенка. SQL-код создания соответствующих таблиц базы данных размещен в прямоугольных выносках, прикрепленных к таблицам пунктирными линиями. Метаданные документа представлены в «основной надписи», выполненной в соответствии со стандартом оформления конструкторской документации. Студентам-разработчикам предоставляется персонализированный бланк документа и набор фигур-трафаретов, содержащих фигуры таблиц, атрибутов, связей ссылочной целостности, выноски для записи SQL-кода. При составлении схемы на бланк переносятся копии трафаретов, соединяются между собой, вводятся имена таблиц и атрибутов, а также текст SQL-кода.

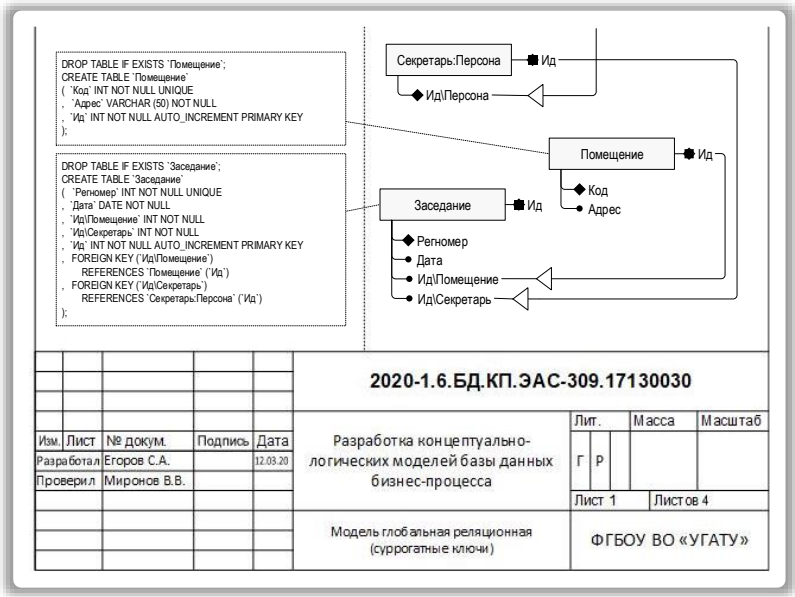


Рис. 11. Фрагмент примера схемы — реляционной модели базы данных: основная надпись документа (*внизу*); диаграмма реляционной модели (*справа*); SQL-код создания таблиц (*слева*)

В зависимости от решаемой задачи выполняется извлечение из этого графического документа следующих сведений:

• структуры реляционной модели — используется для автоматической проверки ее синтаксической корректности, а также соответствия заданию и результатам предшествующих этапов разработки;

• текста SQL-кода — используется для автоматической проверки соответствия SQL-кода, составленного студентом, диаграмме реляционной модели, а также для автоматической генерации заготовок проектной документации (например, документа «текст программы»);

• метаданных из основной надписи документа — для использования в отчетах и проектной документации. В результате достигается заметное уменьшение рутинных действий по контролю правильности составленной схемы и подготовки отчетной документации как у преподавателей, так и у студентов.

Таким образом, практическое применение предложенного подхода дает положительный эффект в виде нового качества за счет автоматизированного извлечения семантической информации из графических документов.

9. Заключение. В данной статье изложен научно обоснованный подход к извлечению семантической информации (знаний) из исходных графических данных. Этот подход обладает новизной, поскольку впервые позволяет в автоматизированном режиме извлекать структурно-параметрические свойства схем из электронных документов векторной графики, выполненных в открытых графических форматах на базе XML. Подход предполагает использование наборов заранее разработанных трафаретов для условных обозначений схемы. Предложенный в рамках подхода алгоритм построения концептуально-логического отображения предусматривает перебор фигур внутреннего представления графического документа и идентификацию условных графических обозначений, их соединений и свойств. В плане практической реализации подхода разработаны модели идентификации условных графических обозначений применительно к различным графическим форматам. Практическая реализация выполнена на основе ситуационно-ориентированной парадигмы, предусматривающей отображение виртуальных документов на графические файлы. Подход прошел успешную проверку работоспособности и получил практическое применение.

Литература

1. *Pieris D., Wijegunesekera M.C., Dias N.G.J.* ER model partitioning: Towards trustworthy automated systems development // International Journal of Advanced Computer Science and Applications. 2020. vol. 11, № 6. pp. 286–293. DOI: 10.14569/IJACSA.2020.0110638.
2. *Pérez R., Guerrero R.* A computer agent that develops visual compositions based on the ER-model // Annals of Mathematics and Artificial Intelligence. 2020. vol. 88, no. 5–6. P. 549–588. DOI: 10.1007/s10472-019-9616-3.
3. *Coelho D., Mueller K.* Infomages: Embedding Data into Thematic Images // Computer Graphics Forum. 2020. Vol. 39, no. 3. pp. 593–606. DOI: 10.1111/cgf.14004.

4. *Tsandilas T.* StructGraphics: Flexible Visualization Design through Data-Agnostic and Reusable Graphical Structures // IEEE Transactions on Visualization and Computer Graphics. 2021. vol. 27, no. 2. pp. 315–325. DOI: https://doi.org/10.1109/TVCG.2020.3030476.
5. *Yu Z., Xiong Z.* Comparative Analyses for the Performance of Rational Rose and Visio in Software Engineering Teaching // Journal of Physics: Conference Series. IOP Publishing. 2018. vol. 1087. no. 6. pp. 062–041. DOI: https://doi.org/10.1088/1742-6596/1087/6/062041.
6. *Parker D. J.* Mastering Data Visualization with Microsoft Visio Professional 2016 // Packt Publishing Ltd. 2016. P. 334.
7. *He L., Lian J.* Instructional Design of Practice Course of Logistics System Planning and Design Based on Visio // The 9th International Conference on Information Technology in Medicine and Education (ITME'2018), IEEE, 2018. pp. 526–530. DOI: 10.1109/ITME.2018.00122.
8. *Ruiz Ledesma E.F. et al.* Educational tool for generation and analysis of multidimensional modeling on data warehouse // International Journal of Advanced Computer Science and Applications. 2020. vol. 11, no. 9. pp. 261–267. DOI: 10.14569/IJACSA.2020.0110930.
9. *Shafiee S. et al.* Evaluating the benefits of a computer-aided software engineering tool to develop and document product configuration systems // Computers in Industry. 2021. vol. 128. DOI: 10.1016/j.compind.2021.103432.
10. *Medoh C., Telukdarie A.* Business Process Modelling Tool Selection: A review // IEEE International Conference on Industrial Engineering and Engineering Management (IEEM'2017). IEEE. 2017. pp. 524–528. DOI: 10.1109/IEEM.2017.8289946.
11. *Afanasyev A., Voit N., Gaynullin R.* The analysis of diagrammatic of workflows in design of the automated systems // Uncertainty Modelling in Knowledge Engineering and Decision Making. 2016. pp. 509-514. DOI: 10.1142/ 9789813146976_0082.
12. *Voit N., Bochkov S., Kirillov S.* Temporal Automaton RVTI-Grammar for the Diagrammatic Design Workflow Models Analysis // IEEE 14th International Conference on Application of Information and Communication Technologies (AICT'2020), Tashkent: Uzbekistan, 2020, pp. 1-6. DOI: 10.1109/AICT50176.2020.9368810.
13. *Afanasyev A., Voit N., Ukhanova M., Ionova I.* Development of the approach to check the correctness of workflows // Data Science and Knowledge Engineering for Sensing Decision Support (ITIDS'2018), pp. 1392-1399. DOI: 10.1142/9789813273238_0173.
14. *Shah R., Kesan J.* Interoperability challenges for open standards: ODF and OOXML as examples // Proceedings of the 10th Annual International Conference on Digital Government Research: Social Networks: Making Connections between Citizens, Data and Government (dg.o'09). Puebla: Digital Government Society of North America. 2009. pp. 56–62.
15. *Doncevic J., Fertalj K.* Database integration systems // Proceedings of 43rd International Convention on Information, Communication and Electronic Technology, (MIPRO'2020), 2020. pp. 1617–1622. DOI: .https://doi.org/10.23919/MIPRO48935.2020.9245245.
16. *Kolonko M., Mullenbach S.* Polyglot Persistence in Conceptual Modeling for Information Analysis // Proceedings of 10th International Conference on Advanced Computer Information Technologies, (ACIT'2020), 2020. pp. 590–594. DOI: https://doi.org/10.1109/ACIT49673.2020.9208928.
17. *Kosmerl I., Rabuzin K., Sestak M.* Multi-model databases - Introducing polyglot persistence in the big data world // Proceedings of 43rd International Convention on Information, Communication and Electronic Technology, (MIPRO'2020), 2020. pp. 1724–1729. DOI: 10.23919/MIPRO48935.2020.9245178.

18. *Montgomery C., Isah H., Zulkernine F.* Towards a Natural Language Query Processing System // Proceedings of 1st International Conference on Big Data Analytics and Practices (IBDAP'2020), 2020. DOI: 10.1109/IBDAP50342.2020.9245462.
19. *Миронов В.В., Гусаренко А.С., Юсупова Н.И.* Структурирование виртуальных мультидокументов в ситуационно-ориентированных базах данных с помощью entry-элементов // Труды СПИИРАН. 2017. № 53. С. 225–243. DOI: 10.15622/sp.53.11.
20. *Mironov V.V., Gusarenko A.S., Yusupova N.I.* Situation-oriented databases: document management on the base of embedded dynamic model // CEUR Workshop Proceedings (CEUR-WS.org): Selected Papers of the XI International Scientific-Practical Conference Modern Information Technologies and IT-Education (SITITO'2016), Moscow: Russia. 2016. vol. 1761. 2016. pp. 238-247.
21. *Mironov V., Gusarenko A., Yusupova N.* JSON Documents Processing Using Situation-Oriented Databases // Acta Polytechnica Hungarica. 2020. vol. 17. No. 8. pp. 29–40. DOI: 10.12700/APH.17.8.2020.8.3.
22. *Mironov V., Gusarenko A., Tuguzbaev G.* Graphic Documents Parametric Personalization for Information Support of Educational Design Using Situation-Oriented Databases // Advances in Intelligent Systems Research – Proceedings of the 8th Scientific Conference on Information Technologies for Intelligent Decision-Making Support (ITIDS'2020). pp. 260–267. DOI: 10.2991/aisr.k.201029.050

Миронов Валерий Викторович — д-р техн. наук, профессор, кафедра автоматизированных систем управления, факультет информатики и робототехники, УГАТУ. Область научных интересов: иерархическое моделирование, динамические модели, разработка баз данных и обработка данных в NoSQL СУБД. Число научных публикаций — 100. mironov@ugatu.su, http://hsm.ugatu.su; УГАТУ, ул. К. Маркса, 12, Уфа, 450008, РФ; р. т.: +7(347)273-77-17.

Гусаренко Артем Сергеевич — канд. техн. наук, доцент, кафедра автоматизированных систем управления, факультет информатики и робототехники, УГАТУ. Область научных интересов: иерархическое моделирование, динамические модели, разработка баз данных и обработка данных в NoSQL СУБД. Число научных публикаций — 50. gusarenko@ugatu.su, http://hsm.ugatu.su; http://itids.ugatu.su; http://csit.ugatu.su; УГАТУ, ул. К. Маркса, 12, Уфа, 450008, РФ; р. т.: +7(347)273-77-17.

Тугузбаев Гаяз Ахтямович — аспирант, кафедра автоматизированных систем управления, факультет информатики и робототехники, УГАТУ. Область научных интересов: разработка баз данных и обработка данных в NoSQL. Число научных публикаций — 3. tuguzbaev.g@ugatu.su; Карла Маркса, 12, Уфа, 450008, РФ; р. т.: +7(347)273-77-17.

Поддержка исследований. Работа выполнена при финансовой поддержке РФФИ (проект № 19-07-00682-а).

DOI 10.15622/ia.20.4.7

V. Mironov, A. Gusarenko, G. Tuguzbaev

EXTRACTING SEMANTIC INFORMATION FROM GRAPHIC SCHEMES

Mironov V., Gusarenko A., Tuguzbaev G. **Extracting Semantic Information from Graphic Schemes.**

Abstract. The problem of extracting semantic information from an electronic document specified in the vector graphics format and containing a graphic model (diagram) built using a graphic editor is considered. The problem is to program retrieving certain structural properties and parametric circuit and entering them into a database for later use. Based on the analysis of the capabilities of graphic editors, a conclusion has made about the relevance of this task for universal editors that are not tied to specific graphic notations and use open graphic document formats, which allows program processing. The proposed approach considers graphic documents at three levels of abstraction: conceptual (semantic properties of a schema), logical (presentation of semantic properties at the internal level of the document) and physical (internal organization of a graphic document). The solution to the problem is based on the construction of a conceptual-logical mapping, i.e., mapping a conceptual model of a circuit to a logical model of a graphic document, according to its physical model. Within the framework of the approach, an algorithm for constructing the indicated mapping is developed, presented in the form of an object-oriented pseudocode. The study of internal markup in open graphic formats made it possible to build models for identifying circuit elements and their connections to each other, which is necessary for a specific application of the algorithm. Expressions for addressing schema elements and accessing their properties are obtained. The proposed approach is implemented on the base of a situation-oriented paradigm, within which the extraction process is driven by a hierarchical situational model. The processed data is specified in the situational model in the form of virtual documents displayed on heterogeneous external data sources. For the problem being solved, we consider the mapping to two variants of vector graphics formats: to a "flat" markup file and to a set of such files in an electronic archive. The practical use of the results is illustrated by the example of extracting semantic information from graphical models developed at various stages of database design.

Keywords: Block Diagram, Vector Graphics, Property Extraction, Situational Paradigm, Hierarchical Situational Model, Virtual Document.

Mironov Valeriy — Dr. Tech. Sci., Professor, Department of Automated Control Systems, Faculty of Informatics and Robotics, USATU. Research interests: hierarchical modeling, dynamic models, database development and data processing in NoSQL DBMS. The number of scientific publications - 100. mironov@ugatu.su, http://hsm.ugatu.su; Karl Marx, 12, Ufa, 450008, Russian Federation; office phone: +7 (347) 273-77-17.

Gusarenko Artem — Candidate of Technical Sciences, Associate Professor, Department of Automated Control Systems, Faculty of Informatics and Robotics, USATU. Research interests: hierarchical modeling, dynamic models, database development and data processing in NoSQL DBMS. The number of scientific publications - 50. gusarenko@ugatu.su, http://hsm.ugatu.su; http://itids.ugatu.su; http://csit.ugatu.su; Karl Marx, 12, Ufa, 450008, Russian Federation; office phone +7 (347) 273-77-17.

Tuguzbaev Gayaz — Post-graduate student, Department of Automated Control Systems, Faculty of Informatics and Robotics, USATU. Research interests: object-oriented databases, DBMS,

database development and data processing in NoSQL. Number of scientific publications - 3. tuguzbaev.g@ugatu.su; Karl Marx, 12, Ufa, 450008, RF; office phone: +7 (347) 273-77-17.

Acknowledgements. This research is supported by RFBR (grant 19-07-00682-a).

References

1. Pieris D., Wijegunesekera M.C., Dias N.G.J. ER model partitioning: Towards trustworthy automated systems development // *International Journal of Advanced Computer Science and Applications*. 2020. vol. 11, № 6. pp. 286–293. DOI: https://doi.org/10.14569/IJACSA.2020.0110638.
2. Pérez R., Guerrero R. A computer agent that develops visual compositions based on the ER-model // *Annals of Mathematics and Artificial Intelligence*. 2020. vol. 88, no. 5–6. P. 549–588. DOI: https://doi.org/10.1007/s10472-019-9616-3.
3. Coelho D., Mueller K. Infomages: Embedding Data into Thematic Images // *Computer Graphics Forum*. 2020. vol. 39, no. 3. pp. 593–606. DOI: 10.1111/cgf.14004.
4. Tsandilas T. StructGraphics: Flexible Visualization Design through Data-Agnostic and Reusable Graphical Structures // *IEEE Transactions on Visualization and Computer Graphics*. 2021. vol. 27, no. 2. pp. 315–325. DOI: https://doi.org/10.1109/TVCG.2020.3030476.
5. Yu Z., Xiong Z. Comparative Analyses for the Performance of Rational Rose and Visio in Software Engineering Teaching // *Journal of Physics: Conference Series. IOP Publishing*. 2018. vol. 1087. no. 6. pp. 062–041. DOI: https://doi.org/10.1088/1742-6596/1087/6/062041.
6. Parker D. J. *Mastering Data Visualization with Microsoft Visio Professional 2016* // Packt Publishing Ltd. 2016. P. 334.
7. He L., Lian J. Instructional Design of Practice Course of Logistics System Planning and Design Based on Visio // The 9th International Conference on Information Technology in Medicine and Education (ITME'2018), IEEE, 2018. pp. 526–530. 10.1109/ITME.2018.00122.
8. Ruiz Ledesma E.F. et al. Educational tool for generation and analysis of multidimensional modeling on data warehouse // *International Journal of Advanced Computer Science and Applications.* 2020. vol. 11, no. 9. pp. 261–267. DOI: 10.14569/IJACSA.2020.0110930.
9. Shafiee S. et al. Evaluating the benefits of a computer-aided software engineering tool to develop and document product configuration systems // *Computers in Industry*. 2021. vol. 128. DOI: 10.1016/j.compind.2021.103432.
10. Medoh C., Telukdarie A. Business Process Modelling Tool Selection: A review // IEEE International Conference on Industrial Engineering and Engineering Management (IEEM'2017). IEEE. 2017. pp. 524–528. DOI: 10.1109/IEEM.2017.8289946.
11. Afanasyev A., Voit N., Gaynullin R. The analysis of diagrammatic of workflows in design of the automated systems // *Uncertainty Modelling in Knowledge Engineering and Decision Making*. 2016. pp. 509-514. DOI: 10.1142/9789813146976_0082.
12. Voit N., Bochkov S., Kirillov S. Temporal Automaton RVTI-Grammar for the Diagrammatic Design Workflow Models Analysis // IEEE 14th International Conference on Application of Information and Communication Technologies (AICT'2020), Tashkent: Uzbekistan, 2020, pp. 1-6. DOI: 10.1109/AICT50176.2020.9368810.
13. Afanasyev A., Voit N., Ukhanova M., Ionova I. Development of the approach to check the correctness of workflows // Data Science and Knowledge Engineering for Sensing Decision Support (ITIDS'2018), pp. 1392-1399. DOI: 10.1142/9789813273238_0173.
14. Shah R., Kesan J. Interoperability challenges for open standards: ODF and OOXML as examples // Proceedings of the 10th Annual International Conference on Digital Government Research: Social Networks: Making Connections between Citizens, Data and

Government (dg.o'09). Puebla: Digital Government Society of North America. 2009. pp. 56–62.

15. Doncevic J., Fertalj K. Database integration systems // Proceedings of 43rd International Convention on Information, Communication and Electronic Technology, (MIPRO'2020), 2020. pp. 1617–1622. DOI: 10.23919/MIPRO48935.2020.9245245.
16. Kolonko M., Mullenbach S. Polyglot Persistence in Conceptual Modeling for Information Analysis // Proceedings of 10th International Conference on Advanced Computer Information Technologies, (ACIT'2020), 2020. pp. 590–594. DOI: 10.1109/ACIT49673.2020.9208928.
17. Kosmerl I., Rabuzin K., Sestak M. Multi-model databases - Introducing polyglot persistence in the big data world // Proceedings of 43rd International Convention on Information, Communication and Electronic Technology, (MIPRO'2020), 2020. pp. 1724–1729. DOI: 10.23919/MIPRO48935.2020.9245178.
18. Montgomery C., Isah H., Zulkernine F. Towards a Natural Language Query Processing System // Proceedings of 1st International Conference on Big Data Analytics and Practices (IBDAP'2020), 2020. DOI: 10.1109/IBDAP50342.2020.9245462.
19. Mironov V.V., Gusarenko A.S., Yusupova N.I. Structuring virtual multi-documents in situationally oriented databases by means of entry-elements // *SPIIRAS Proceedings*. 2017. vol. 4, № 53. pp. 225–242. DOI: 10.15622/sp.53.11.
20. Mironov V.V., Gusarenko A.S., Yusupova N.I. Situation-oriented databases: document management on the base of embedded dynamic model // CEUR Workshop Proceedings (CEUR-WS.org): Selected Papers of the XI International Scientific-Practical Conference Modern Information Technologies and IT-Education (SITITO'2016), Moscow: Russia. 2016. vol. 1761. 2016. pp. 238-247.
21. Mironov V., Gusarenko A., Yusupova N. JSON Documents Processing Using Situation-Oriented Databases // *Acta Polytechnica Hungarica*. 2020. vol. 17. no. 8. pp. 29–40. DOI: 10.12700/APH.17.8.2020.8.3.
22. Mironov V., Gusarenko A., Tuguzbaev G. Graphic Documents Parametric Personalization for Information Support of Educational Design Using Situation-Oriented Databases // Advances in Intelligent Systems Research – Proceedings of the 8th Scientific Conference on Information Technologies for Intelligent Decision-Making Support (ITIDS'2020). pp. 260–267. DOI: 10.2991/aisr.k.201029.050.