Отображение графовых моделей данных в каноническую модель в системах с интенсивным использованием данных[1]

УДК 004.652

С. А. Ступников – к.т.н., с.н.с. ИПИ РАН, E-mail: ssa@ipi.ac.ru

Для интеграции неоднородных информационных ресурсов, содержащих большие данные, необходима *унификация* их моделей данных - отображение в каноническую информационную модель, сохраняющее информацию и семантику языка определения данных и операций языка манипулирования данными. Данная работа посвящена унификации важного вида моделей, называемых *графовыми*. Обсуждаются основные современные графовые модели, их черты и области применения, особенности использования графовых моделей при манипулировании большими данными. Рассмотрены вопросы доказательства сохранения информации и семантики операций при отображении графовых моделей в объектно-фреймовую каноническую модель с использованием формального языка спецификаций AMN.

Ключевые слова: графовые модели данных, атрибутированные графы, большие графы, отображения моделей данных, интеграция информационных ресурсов

Keywords: graph data models, attributed graphs, large scale graphs, data model mappings, information resource integration

1. Введение

Роль данных в различных областях деятельности человека – научных исследованиях, здравоохранении, образовании, промышленности и т.д. – непрерывно растет в последние годы. Укрепляется новая парадигма в науке и

[1] Работа выполнена при поддержке РФФИ (гранты 13-07-00579, 14-07-00548) и Президиума РАН (Программа фундаментальных исследований Президиума РАН № 16 «Фундаментальные проблемы системного программирования».

информационных технологиях, связанная с интенсивным использованием данных – так называемая *четвертая парадигма* [1]. *Эмпирическая парадигма,* господствовавшая в науке в течение тысяч лет, состояла в описании природных явлений. Сотни лет назад ее расширила *теоретическая парадигма*, предполагающая использование моделей и обобщений. Десятки лет назад возникла *вычислительная парадигма*, дополнившая способы научного познания симуляцией сложных явлений. Современная научная парадигма объединяет теорию, эксперимент и симуляцию. Данные поступают от инструментов или симуляторов, обрабатываются (очищаются) программным обеспечением, а затем ученые производят анализ извлеченной информации. Основная проблема новой парадигмы состоит в том, что объемы поступающих данных значительно превышают возможности современных методов и средств хранения, обработки и анализа данных.

Развиваются новые информационные технологии, в которых данные становятся доминирующим фактором, новые подходы к концептуализации, организации и реализации информационных систем. При этом требуется не только создание методов и средств оперирования данными, объемы которых выходят за рамки возможностей современных технологий баз данных, но и разработка новых подходов, позволяющих справляться с разнообразием массово и хаотично развивающихся языков и моделей данных.

Данная статья[2] продолжает исследования по унификации моделей данных, применяемых в системах с интенсивным использованием данных, для виртуальной или материализованной интеграции ресурсов при создании федеративных баз данных или хранилищ данных. К таким моделям относятся разнообразные NoSQL-модели; онтологические и семантические модели; графовые модели; модели, основанные на многомерных массивах и т.д. *Унификацией* модели данных ресурса называется ее отображение в *каноническую информационную модель* (служащую общим языком в среде разнообразных моделей ре-

[2] Статья является расширенной версией доклада, опубликованного в трудах конференции RCDL 2014.

сурсов), сохраняющее информацию и семантику операций языка манипулирования данными (ЯМД) [2]. В качестве канонической модели в данной работе рассматривается объектно-фреймовая модель данных, а именно - язык СИНТЕЗ [3], нацеленный на разработку предметных посредников или хранилищ данных для решения задач в средах неоднородных ресурсов и поддержанный программными средствами исполнительной среды предметных посредников. Более подробно мотивация и общие принципы унификации различных классов нетрадиционных моделей изложены в работе, включенной в настоящий выпуск журнала [4].

В данной статье рассматривается важный класс моделей данных – *графовые модели*. Исследования графовых моделей начались в середине 1980-х годов. Математическими основаниями для них послужила теория графов, а наибольшее влияние оказали так называемые *семантические модели* (например, модель «сущность-связь») [5]. Целью графовых моделей было преодоление ограничений, налагаемых традиционными моделями данных, связанных с представлением исходных графовых структур данных. Важность графовых баз данных обусловлена разнообразием приложений – это и социальные сети, рекомендательные системы, геопространственные приложения и даже обнаружение мошенничества. Современные графовые СУБД обычно ориентированы на обработку и анализ *больших графов*, насчитывающих миллиарды вершин и ребер. Например, сеть Facebook насчитывает 1.3 миллиарда профилей пользователей и 150 миллиардов связей «дружбы» между профилями. Весь Веб можно также рассматривать как граф страниц (вершины), соединенных гиперссылками (ребрами). Ведущим интернет-компаниям, таким, как Google, приходится осуществлять вычисления именно на таком графе.

Статья организована следующим образом.

В разделе 2 рассматриваются современные графовые СУБД и графовые модели данных.

В разделе 3 рассмотрена и проиллюстрирована на примере модель данных атрибутированных графов.

В разделе 4 рассмотрены и проиллюстрированы основные принципы отображения модели данных атрибутированных графов в язык СИНТЕЗ.

В разделе 5 рассмотрены вопросы доказательства сохранения информации и семантики операций при отображении графовых моделей в объектные с использованием формального языка спецификаций AMN.

В разделе 6 рассмотрены родственные исследования и направления дальнейшей работы.

2. Графовые СУБД и графовые модели данных

Основными отличительными чертами графовых моделей данных являются следующие [5]:

- данные и/или схема данных представляются в виде графов или структур данных, обобщающих понятие графа (гиперграфы или гипервершины);
- манипулирование данными выражается в виде трансформаций графов или при помощи операций, основными параметрами которых являются такие характерные графовые структуры и свойства, как пути, подграфы, связность и т.д.;
- ограничения целостности тесно связаны с графами как структурой данных. Так, ограничениями могут быть уникальность меток ребер и вершин, типизация ребер и вершин, ограничения на область определения и область значений свойств ребер и вершин.

Графовые модели данных применяются в тех случаях, когда информация о взаимосвязях между данными или их топологии является более важной (или настолько же важной), как сами данные. Поводом к использованию графовых моделей может быть также недостаточная выразительная сила языков запросов традиционных моделей. Наиболее распространенными примерами применения графовых моделей являются системы управления и анализа сложных

сетей – социальных, биологических, информационных, транспортных, телекоммуникационных и других [6].

Социальные приложения графовых баз данных (БД) позволяют оперировать информацией о связях между людьми, поддерживать совместную работу и потоки информации, предсказывать поведение. Социальные сети помогают выделять прямые и непрямые отношения между людьми и группами, позволяя пользователям оценивать, рецензировать, находить интересующие вещи и друг друга. Понимание того, как люди связаны и как они взаимодействуют друг с другом, как представитель группы действует или осуществляет свой выбор на основании агрегированного поведения группы, позволяет выявить скрытые силы, влияющие на поведение индивидуумов.

Рекомендательные алгоритмы устанавливают связи между людьми и вещами (продуктами, сервисами, медиа-контентом) – всем тем, что релевантно предметной области, в которой используется рекомендательная система. Связи устанавливаются на основании поведения пользователей (покупки, оценки и т.д.). Как и в случае социальных сетей, эффективная рекомендация зависит от понимания связей между вещами, а также качества и силы связей. Такие структуры наилучшим образом представляются в виде атрибутированных графов. Запросы в графах являются преимущественно локальными, т.к. их отправной точкой являются один или несколько идентифицируемых объектов, а дальнейший поиск производится в окрестности этих объектов.

Геопространственные приложения графовых БД варьируются от вычисления маршрутов в транспортных или логистических сетях до пространственных операций, таких, как поиск всех интересующих точек в ограниченной области, нахождение центра региона, вычисление пересечения регионов. Геопространственные операции зависят от используемых структур данных, начиная с простых взвешенных и направленных связей и до пространственных индексов, таких, как R-деревья. Такие индексы естественным образом представляются в виде графов. Геопространственные данные могут существовать в графовой БД совместно с другими видами данных, например, социальными.

При этом становятся возможными сложные многомерные запросы над несколькими предметными областями одновременно. Геопространственные приложения особенно актуальны в областях телекоммуникаций, логистики, путешествий, планирования маршрутов.

Основными данными (master data) называются данные, являющиеся критическими для бизнес-операций. Основные данные включают данные о пользователях, покупателях, продуктах, поставщиках, отделах, сайтах и т.д. В больших организациях такие данные сильно распределены и неоднородны по форматам, качеству и средствам доступа. *Управление основными данными* включает идентификацию, очистку, хранение и собственно, управление данными (governance). Управление основными данными должно адаптироваться к изменению организационной структуры, слиянию организаций, изменению бизнес-правил, включению новых информационных ресурсов и т.д. Графовые БД хорошо подходят для моделирования, хранения и запросов к основным метаданным и моделям основных данных.

Графовые БД предлагают решение для создания нового поколения систем *обнаружения мошенничества* (fraud detection). В таких системах применяются интеллектуальные рассуждения, в отличие от алгоритмов статистического анализа или распознавания образцов. Из сведений о транзакциях и пользователях обычно выводятся данные о связях между ними. Эти связи обходят с целью обнаружения потенциальных образцов мошенничества. Дополнительная информация о пользователях (принадлежность к организации; персональные, профессиональные и семейные связи; история адресов и т.д.) существенно способствует обнаружению образцов. Транзакция связывается с известным мошенником, либо образец транзакции связывается с мошенническими образцами. Графовые БД хорошо подходят для быстрого обхода связей (миллионы связей в секунду), обнаружения образцов и их визуализации.

Наибольшего разнообразия в своем развитии графовые модели достигли в 1990-х годах. Наряду с обычными графами, представляющими собой множе-

ства вершин (помеченных или непомеченных), соединенных ребрами (направленными или ненаправленными, помеченными или непомеченными), развитие в графовых моделях получили такие структуры, как *гипервершины* и *гиперграфы*. Гипервершина представляет собой вложенный граф, а ребра гиперграфа могут соединять не две, а произвольное количество вершин [7].

В настоящее время в мире наибольший интерес вызывают графовые СУБД, ориентированные на обработку и анализ больших графов. Масштаб «больших» графов можно представить на примере сети Facebook – на вторую половину 2013 г. она насчитывает 1.3 миллиарда профилей пользователей и 150 миллиардов связей «дружбы» между профилями. Весь Веб можно также рассматривать как граф страниц (вершины), соединенных гиперссылками (ребрами). В этом случае речь также идет о графе с миллиардами вершин и ребер. Ведущим интернет-компаниям, таким, как Google, приходится осуществлять вычисления именно на таком графе.

Разрабатываются и исследуются различные инфраструктуры масштабируемой распределенной обработки и анализа графов.

Графовая модель системы *Pregel* [8], разработанная компанией Google, обладает следующими особенностями:

- входными данными Pregel-программы является ориентированный граф, каждая вершина которого идентифицируется уникальным идентификатором строкового типа;
- с вершиной ассоциируется изменяемое значение типа, задаваемого пользователем;
- направленные ребра ассоциируются с исходной вершиной;
- с ребром ассоциируется изменяемое значение типа, задаваемого пользователем, и идентификатор целевой вершины.

Основой модели вычислений Pregel является модель BSP (Bulk Synchronous Parallel). Вычисление состоит из последовательности итераций (супершагов). Перед началом вычисления все вершины являются активными.

На каждом шаге для каждой активной вершины вызывается определенная пользователем функция, вызовы для всех вершин могут быть параллельными. Функция определяет поведение в отдельной вершине *V* на шаге *S*. Функция может читать сообщения, посланные вершине *V* другими вершинами на шаге *S-1*, посылать сообщения другим вершинам, которые будут получены на шаге *S+1*, изменять состояние *V* и исходящих ребер, добавлять и удалять ребра и вершины. Функция может перевести вершину в неактивное состояние. Вершина вновь становится активной, если ей придут новые сообщения. Вычисление завершается, когда все вершины становятся неактивными. Результатом вычисления является преобразованный граф. Такая модель вычислений называется *вершинно-центрированной*, поскольку основными объектами манипулирования в ней являются вершины.

Система Pregel является проприетарной, однако, корпорацией Apache разработан *Giraph* [9] – реализация Pregel с открытым кодом. Giraph основан на известной инфраструктуре распределенных масштабируемых вычислений *Hadoop* [10], запросы пользователя транслируются в задачи программной модели MapReduce. Именно Giraph используется в настоящее время для анализа социального графа Facebook.

Еще один вариант вершинно-центрированной вычислительной модели предлагается в системе *Trinity* [11], разработанной Microsoft Research. Отличие от модели Pregel состоит в том, что на каждом супершаге вершина может получать или принимать сообщения, а также изменять значения только у *ограниченного* множества вершин (обычно, своих соседей). Это позволяет значительно оптимизировать алгоритмы на графах, укладывающиеся в такую модель (например, PageRank или поиск кратчайших путей). Однако, по утверждению разработчиков, Trinity не ограничивается данной моделью. Благодаря тому, что для хранения графов используется так называемое *распределенное облако памяти* [11], в системе может быть реализована любая вычислительная модель, в том числе модель Pregel или модель асинхронных вычислений [12].

Графовая модель данных, используемая в системе Trinity [13], обладает следующими особенностями:

- в модели изначально не выделяются вершины и ребра, эти понятия обобщены и объединены в понятие *ячейка* (cell);
- ячейка определяется уникальным идентификатором;
- с ячейкой могут быть связаны различные множества других ячеек, например:
    - для моделирования ненаправленного графа – множество соседних ячеек (соединенных ребром с данной ячейкой);
    - для моделирования направленного графа – множество ячеек, соединенных с данной ячейкой исходящим ребром, и множество ячеек, соединенных с данной ячейкой входящим ребром;
- с ячейкой может быть связана дополнительная информация (имя, описание и т.д.).

Если для решения задачи ребра графа необходимо аннотировать информацией, то ребра также могут быть представлены в виде ячеек. В этом случае каждая ячейка-вершина связывается с множеством инцидентных ячеек-ребер, а каждая ячейка-ребро – с множеством инцидентных ячеек-вершин. Очевидным образом модель позволяет представлять гиперграфы – с ячейкой-ребром можно связывать не две вершины (которые соединяет ребро), а произвольное количество вершин.

По утверждению разработчиков, система Trinity, в отличие от Pregel, позволяет решать задачи из обоих основных классов задач над графами [14]:

- запросы в режиме реального времени, основывающиеся на обходе графа;
- автономная (offline) аналитика на графах.

Эффективная аналитика обеспечивается использованием ограниченной вершинно-центрированной модели вычислений. Эффективный обход графа в режиме реального времени возможен за счет хранения графа в памяти распределенной системы (распределенное облако памяти) и распараллеливания запро-

сов. В настоящее время ведутся работы по поиску и реализации эффективных алгоритмов решения задач реального времени на больших графах. Например, новый метод поиска подграфов, изоморфных данному, с использованием инфраструктуры Trinity, предложен в работе [15].

Однако, несмотря на важность работ в области вершинно-центрированных моделей вычислений и обобщенных графовых моделей данных, обзоры состояния современных графовых СУБД [5] показывают, что большинство существующих баз данных основаны на простых или атрибутированных графах (*attributed graph* или *property graph*), в которых атрибуты (свойства) приписываются ребрам и/или вершинам графа. Именно такие модели и были выбраны в данной статье в качестве исходных, подлежащих унификации моделей. К числу систем, основанных на простых или атрибутированных графах, относятся Neo4j [16], Dex [17], InfiniteGraph, OrientDB, VertexDB, Filament, OQGraph, Horton [18], InfoGrid и другие.

Такие системы также ориентируются в настоящее время на распределенную обработку больших графов. В качестве основы системы используют различные распределенные инфраструктуры.

Например, система *Horton* [18], разрабатываемая в Microsoft Research, использует распределенную инфраструктуру облачных вычислений *Orleans* [19].

В системе *Titan* [20] в качестве графового хранилища могут использоваться различные NoSQL базы данных: Apache Cassandra, Apache HBase, Oracle BerkleyDB, которые, в свою очередь, предназначены для работы на вычислительных кластерах (БД Cassandra и HBase в качестве инфраструктуры хранения и вычислений используют Hadoop).

Широко распространенная графовая БД Neo4j [16], являющаяся наиболее эффективной среди аналогов [21], использует собственную кластерную архитектуру вида «главный-подчиненный» (master-slave). Ее отличительной особенностью является репликация графа на каждый узел кластера. В настоящее

время система поддерживает графы размером в десятки миллиардов вершин и ребер [22]. Потребности крупных компаний, например Google, превышают данные ограничения, однако, этот размер пока достаточен для подавляющего числа компаний, использующих графовые БД.

Отметим, что системы, использующие сегментирование (sharding) графов в кластере вместо репликации на каждом узле, критикуются за то, что в них практически невозможно поддерживать ссылочную целостность между данными на разных узлах. Запрос к распределенному графу фактически превращается в набор локальных запросов к отдельным узлам. К тому же, задача оптимального разбиения графа по узлам кластера является NP-полной, что делает ее практически нерешаемой на больших графах.

Системы, использующие сегментирование, могут быть эффективно использованы для решения задач, данные в которых являются сравнительно «плоскими», когда известно, что характерные запросы к графам будут сопровождаться прохождением по путям небольшой длины. Neo4j напротив, позволяет оперировать с «глубокими» структурами данных и очень быстро осуществлять глубокий поиск в графе. Тем самым обеспечивается практически неограниченная гибкость в моделировании графов предметной области.

3. Модель данных атрибутированных графов

В настоящее время существует большое количество СУБД, модели данных которых основаны на простых или атрибутированных графах. Языки определения данных (ЯОД), языки манипулирования данными (ЯМД), прикладные интерфейсы пользователя (API) различаются в этих системах весьма существенно. Для того, чтобы обеспечить общность подхода по унификации графовых моделей, в данной статье рассматривается синтетическая модель данных атрибутированных графов. Структуры данных модели покрывают возможности моделей таких известных систем, как, например, Neo4j [16], Dex [17], InfiniteGraph, OrientDB, VertexDB, Filament, OQGraph, Horton [18], InfoGrid.

В качестве ЯМД синтетической модели рассматривается декларативный язык Cypher [23], развиваемый в системе Neo4j. Поэтому, фактически, рассматриваемая модель является расширением графовой модели Neo4j. С точки зрения общности по отношению к другим графовым языкам запросов, язык Cypher покрывает такие классы возможностей, как смежность (adjacency) вершин и ребер, достижимость по путям фиксированной длины (fixed-length paths reachability), достижимость по простым регулярным путям (regular simple paths), поиск кратчайших путей, поиск подграфов по образцу (pattern matching) [5]. Это также означает, что язык Cypher покрывает возможности языков запросов основных современных графовых баз данных (в том числе, перечисленных в предыдущем параграфе).

Итак, синтетическая модель выбрана таким образом, чтобы рассматриваемые методы отображения ее в каноническую (раздел 3) можно было применить для унификации различных реальных графовых моделей систем, упомянутых выше.

Заметим, что в данной работе не рассматривается важный класс СУБД, основанных на модели RDF [24], включающий такие системы, как AllegroGraph, G-Store, BrightstarDB. Часто RDF относят к графовым моделям. Однако, ввиду обширности области применения и развития приложений RDF, а также специфики ЯОД, ЯМД и семантики RDF по сравнению с основной массой графовых моделей, вопросы унификации RDF следует рассматривать отдельно [4][25].

Рассмотрим сначала вопросы определения данных в модели данных атрибутированных графов.

База данных в модели есть граф, вершины и ребра которого *типизированы*. Тип вершины или ребра представляет собой, фактически, совокупность атрибутов (свойств), приписываемых вершине или ребру. Определим формально множество всевозможных типов вершин *VertexTypes* и множество типов ребер *EdgeTypes*.

Так, *VertexTypes* представляет собой множество троек вида *<id, name, A>*, где *id* – идентификатор типа (например, целое число), *name* – имя типа (строка символов), *A* – подмножество множества всевозможных атрибутов *Attributes*. Идентификатор необходим для описания атрибута, поскольку его имя может быть не уникальным.

Множество атрибутов *Attributes* есть множество кортежей вида *<id, name, type>*, где *id* и *name* – идентификатор и имя атрибута соответственно, *type* $\in B$ – тип атрибута, *B* – множество встроенных типов (например, *boolean, int, float, string,*типы массивов и т.д.)

Множество *EdgeTypes* есть набор кортежей вида *<id, name, A, directed, restricted, head, tail>,* где *id* – идентификатор типа; *name* – имя типа; $A \subseteq Attributes$; *directed* $\in$ *{true, false}* – флаг направленности ребра; *restricted* $\in$ *{true, false}* – булевский флаг определенности типов вершин, которые связывает ребро; *head* $\in$ *VertexTypes* – тип исходящей вершины ребра; *tail* $\in$ *VertexTypes* – тип входящей вершины ребра.

Для любого типа $T \in EdgeTypes$ если *restricted(T) = true*, то значения *head(T), tail(T)* определены; если же *restricted(T) = false*, то значения *head(T), tail(T)* не определены.

Произвольная *схема S = <VT(S), ET(S)>* обобщенной графовой модели включает два множества: множество типов вершин $VT(S) \subseteq VertexTypes$ и множество типов ребер $ET(S) \subseteq EdgeTypes$.

База данных (граф) *G*, удовлетворяющий схеме *S* имеет вид *G = <V, E>*, где

- $V = \{v \mid \exists T.(T \in VT(S) \ \& \ v: T)\}$ – множество вершин графа такое, что любая вершина имеет тип из *VT(S)*;
- $E = \{<e, t, h> \mid \exists T.(T \in ET(S) \ \& \ e: T \ \& \ t \in V \ \& \ t: tail(T) \ \& \ h \in V \ \& \ h: head(T))\}$ – множество ребер графа такое, что любое ребро имеет тип из *ET(S)* и соединяет вершины из *V*.

Типизация *x: T* означает, что для вершины (ребра) *x* могут быть определены атрибуты из *A(T)* (атрибуты типа *T*).

Рассмотрим пример схемы *Cinema* базы данных фильмов [26] в обобщенной модели. Для упрощения будем опускать в примерах идентификаторы типов и атрибутов, считая, что в данном примере имена однозначно идентифицируют типы:

```
VT(Cinema) = { people, movie }
ET(Cinema) = { cast, directs }
A(movie) = {<id, long>, <title, string>,   <year, integer>}
A(people) = {<id, long>,    <name, string>}
cast = <{<character, string>}, false, false, undefined,undefined>
directs = <∅, true, true, people, movie>
```

Схема включает два типа вершин (*people, movie*) и два типа ребер (*cast, directs*). Тип *movie* включает три атрибута (*id, title, year*), *people* – два (*id, name*), *cast* – один (*character*), *directs* – ни одного. Ребра типа *cast* являются ненаправленными и типы вершин, которые они связывают, не определены; ребра типа *directs* – направлены от вершины типа *people* к вершине типа *movie*.

Пример простого графа *g*, удовлетворяющего схеме *Cinema* выглядит следующим образом:

```
g = <{m, p}, {e}>
m: movie = <id: 1, title: “Lost in Translation”, year: 2003>
p: people = <id: 1, name: “Scarlett Johansson”>
e: cast = <<character: “Charlotte”>, m, p>
```

Вопрос выбора ЯМД для обобщенной графовой модели достаточно сложен. Графовые языки запросов развивались в течение многих лет вместе с самими графовыми моделями [27]. Существуют работы по анализу и сравнению выразительной силы и вычислительной сложности графовых ЯМД [28].

Однако, в современных популярных графовых СУБД языки манипулирования представлены в большинстве случаев просто прикладным интерфей-

сом пользователя (API), предоставляющим доступ к структуре графа, методы обхода графа и инкапсулирующим основные алгоритмы на графах. Не существует стандарта графового языка запросов.

В данной работе в качестве ЯМД модели атрибутированных графов выбран язык Cypher [23]. С одной стороны, этот язык поддерживается и развивается в одной из самых популярных графовых СУБД с открытым кодом – Neo4j. С другой стороны, язык является декларативным, в отличие от существующих графовых API или скриптовых языков (как Gremlin). Основные конструкции и ключевые слова имеют сходство с такими широко распространенными языками, как SQL и SPARQL.

4. Отображение модели атрибутированных графов в каноническую информационную модель

В качестве канонической модели в данной статье рассматривается объектная модель языка СИНТЕЗ [3]. Объектные модели хорошо зарекомендовали себя при унификации различных классов моделей – структурированных, онтологических, сервисных, процессных [29]. Поэтому есть основания выбирать канонические объектные модели при интеграции информационных ресурсов, представленных в моделях различных классов. При этом графовые модели выступают как один из классов моделей ресурсов, подлежащих интеграции.

В качестве канонической модели, кроме языка СИНТЕЗ, могут быть использованы другие существующие объектные или объектно-реляционные модели, например, модель ODMG или SQL:2003. В этом случае также могут быть применены разработанные методы унификации графовых моделей.

4.1. Отображение языка определения данных

Схема в обобщенной графовой модели представляется в языке СИНТЕЗ в виде одноименного модуля (например, *Cinema* – см. пример в разделе 3),

включающего классы, содержащие вершины и ребра графа (например, *vertices* и *edges* соответственно):

```
{ Cinema; in: module;
  { vertices; in: class; … },
  { edges: in: class; … };
   …
}
```

Тип вершины (например, *movie*) представляется в языке СИНТЕЗ одноименным классом (который объявляется подклассом класса всех вершин *vertices*), также входящим в модуль, соответствующий схеме:

```
{ Movie; in: class; superclass: vertices;
  instance_type: {
    id: long;
    title: string;
    year: integer; };
}
```

Атрибуты типа вершины, представляются в языке СИНТЕЗ атрибутами типа экземпляров (*instance_type*) соответствующего класса. Между встроенными типами графовой модели (long, string, int и т.д.) и встроенными типами языка СИНТЕЗ (long, string, integer) устанавливается взаимно-однозначное соответствие.

Тип ребра (например, *directs*) также представляется одноименным классом (который объявляется подклассом класса всех вершин *edges*):

```
{ directs; in: class; superclass: edges;
  instance_type: {
    metaframe
      directed: true;
      restricted: true;
      startVertexType: people;
      endVertexType: movie;
    end
    edgeConstr: {in: invariant;
```

```
        {{ all e/directs.inst (directs(e) ->
              people(e.startVertex) & movie(e.endVertex)) }}
       }; };
  }
```

Атрибуты типа ребра, аналогично типу вершины, представляются в языке СИНТЕЗ атрибутами типа экземпляров соответствующего класса.

Заметим, что информация о направленности (*directed*), определенности ребра (*restricted*), типах его исходящей (*startVertexType*) и входящей (*endVertexType*) вершин представляется специальной конструкцией - метафреймом [36], связанным с типом экземпляра класса. Метафреймы в языке СИНТЕЗ предназначены для выражения дополнительной метаинформации, связанной с такими сущностями, как модули, типы, классы, функции.

Кроме того, ограничение на типы исходящей и входящей вершин представляется инвариантом *edgeConstr*, заданным формулой в типизированной логике первого порядка. Знак *all* означает квантор всеобщности, знак -> - логическую импликацию, & -конъюнкцию, выражение *x/T* – типизацию переменной *x* типом *T*, *C.inst* – тип экземпляров (instance) класса *C*. Предикат *C(x)*, где *C* – имя класса, обращается в истину на экземплярах класса *C*.

Заметим также, что на переменной *e* типа *directs.inst* определены атрибуты исходящей вершины ребра *startVertex* и входящей вершины ребра *endVertex,* хотя их нет непосредственно в типе *directs.inst.* Эти атрибуты являются общими для всех типов вершин и принадлежат типу экземпляров класса *edges*:

```
{ edges; in: class;
  instance_section: {
    startVertex: vertices.inst;
    endVertex: vertices.inst;
    isValidEdge: { in: predicate;
      params: {+stVtx/vertices.inst,
                    +endVtx/vertices.inst
                    returns/Boolean };
      {{ (stVtx = this.startVertex &
```

```
                endVtx = this.endVertex -> returns = true) &
               (stVtx <> this.startVertex |
                endVtx <> this.endVertex -> returns = false) }}
        };
    }
```

Кроме упомянутых атрибутов, тип *edges.inst* включает метод-предикат *isValidEdge*. Предикат *e.isValidEdge(v1, v2)* обращается в истину, если исходящая вершина ребра *e* (*e.startVertex*) совпадает с *v1*, и входящая вершина ребра *e* (*e.endVertex*) совпадает с *v2*. Спецификация метода задается формулой первого порядка, связывающей входные и выходные параметры метода. Знак | означает дизъюнкцию, <> - неравенство, *this* – объект, для которого вызывается метод.

4.2. Отображение языка манипулирования данными

При интеграции неоднородных ресурсов (баз данных, сервисов и т.д.) необходимо отображение ЯОД модели ресурса в каноническую. ЯМД канонической модели, напротив, необходимо отображать в ЯМД модели ресурса, т.к. запросы к посреднику в канонической модели нужно отображать в запросы к ресурсам.

Язык запросов (программ) модели СИНТЕЗ представляет собой Datalog-подобный язык в объектной среде. Программа представляет собой набор конъюнктивных запросов (правил) вида

$q(x/T) :- C_1(x_1/T_1), \ldots, C_n(x_n/T_n), F_1(X_1, Y_1), \ldots F_m(X_m, Y_m), B.$

Тело запроса представляет собой конъюнкцию предикатов-коллекций, функциональных предикатов и ограничения. Здесь C_i - имена коллекций (классов), F_j – имена функций, x_i – имена переменных, значения которых пробегают по классам, T_i – типы переменных, X_j и Y_j – входные и выходные параметры функций, B – ограничение, налагаемое на x_i, X_j, Y_j.

В дальнейшем будет использоваться запись предиката-коллекции вида *movie([title, year])*. Неформально это означает, что нас не интересуют объекты

класса *movie* целиком, а лишь их атрибуты *title, year*. Формально запись означает сокращение от *movie(_/movie.inst[title, year])*. Здесь знак _ обозначает анонимную переменную, *movie.inst* – анонимный тип экземпляров (instance) класса *movie*, *title, year* – необходимые атрибуты типа экземпляров класса.

Будет также использоваться запись *source([i, j, val1/val])*, означающая переименование атрибута *val* в *val1*.

Рассмотрим отображение основных конструкций ЯМД на нескольких примерах.

Пример 1 (Конъюнктивный запрос с использованием предиката смежности вершин и ребер). Рассмотрим запрос, возвращающий имена актеров по фамилии Круз, игравших в фильмах вместе со Скарлетт Йохансон:

```
q([colleague_name]) :-
people(scarlett/[name]),
movies(m),
people(colleague/[collegue_name: name]),
cast(c1), cast(c2),
c1.isValidEdge(m, scarlett),
c2.isValidEdge(m, colleague),
scarlett.name = “Scarlett Johansson”,
colleague.name.like(“*Cruz*”).
```

Запрос вернет непустой результат, если в графе базы данных существуют такие вершины-фильмы *m*, и такие ребра *c1, c2* типа *cast,* что *c1* соединяет *m* с вершиной *scarlett*, и *c2* соединяет *m* с вершиной *colleague.*

В языке Cypher такой запрос имеет вид

```
START scarlett =
    node:node_auto_index(name = 'Scarlett Johansson')
MATCH m-[c1:cast]-scarlett, m-[c2:cast]-collegue
WHERE collegue.name =~ /*Cruz*/
RETURN collegue.name
```

Каждый запрос языка Cypher представляет собой образец (pattern), по которому производится поиск в графе базы данных.

В секции START запроса указываются вершины или ребра, с которых следует начинать поиск. В данном случае это вершина *scarlett,* поскольку для нее указано значение атрибута *name,* а значит, возможен поиск по индексу этого атрибута.

В секции MATCH указывается образец поиска в графе, привязанный к стартовым вершинам. В данном случае это указание, что следует искать фильмы, в которых играла (*Cast*) *scarlett*, а также других актеров, играющих в том же фильме.

В секции WHERE указывается фильтр поиска. В данном случае это фамилия коллеги-актрисы.

В секции RESULT указываются возвращаемые значения. В данном случае это полное имя коллеги-актера.

Основные принципы отображения конъюнктивных запросов объектной модели в язык Cypher, проиллюстрированные на данном примере, состоят в следующем:

- конъюнктивный запрос представляется в языке Cypher запросом, возвращающим результат (секция RETURN);
- предикаты-коллекции и предикат смежности вершин и ребер представляются образцами секции MATCH. Каждому предикату смежности соответствует свой образец. Переменные, типизированные в предикатах-коллекциях, представляются одноименными переменными, использующимися в образцах;
- предикаты-условия представляются соответствующими предикатами секции WHERE или START;
- атрибуты результирующего предиката конъюнктивного запроса представляются одноименными атрибутами в секции RETURN.

Пример 2 (Удаление вершин). Рассмотрим запрос, удаляющий из базы данных фильм «Отчаянный»:

```
-movie(m) :- movie(m), m.year = “Desperado”.
```

В правилах со знаком «–» в голове осуществляется удаление объектов из коллекции.

В языке Cypher такой запрос представляется запросом с секцией DELETE:

```
START m = node:node_auto_index(title = 'Desperado')
DELETE m
```

Пример 3 (Обновление значения атрибута). Рассмотрим запрос, устанавливающий год создания фильма «Васаби»:

```
movie(m/[year]) :-
movie(m/[title, year1/year]), m.title = “Vasabi”,
year = 2001.
```

В языке Cypher такой запрос представляется запросом с секцией SET:

```
START m = node:node_auto_index(title = 'Vasabi')
SET m.year = 2001
RETURN year
```

5. Сохранение информации и семантики операций ЯМД при отображении

В данном разделе рассматриваются вопросы доказательства сохранения информации и семантики операций при отображении графовых моделей в объектные с использованием формального языка спецификаций AMN [30][31]. Применяется метод, предложенный и опробованный при унификации модели, основанной на многомерных массивах, в работе [32].

Язык AMN основан на теории множеств и типизированном языке первого порядка. Спецификации AMN называются *абстрактными машинами* и сочетают в себе пространства состояний и поведения машины, определенного

операциями на состояниях. В языке AMN формализуется специальное отношение между спецификациями – *уточнение*.

Идея метода заключается в следующем. Рассмотрим исходную модель S и целевую модель T. Построим отображение θ модели S в модель T. Выразим семантику моделей в виде абстрактных машин AMN, построив при этом машины M_S и M_T соответственно. При этом структуры данных моделей представляются переменными машин, свойства структур данных представляются инвариантами машин, характерные операции моделей данных представляются операциями машин. *Операциями* в данном случае называются характерные родовые запросы в языках СИНТЕЗ и Cypher соответственно.

Рассматриваемые операции исходной и целевой модели должны быть связаны отображением ЯМД. Отображение ЯОД представляется в виде специального *склеивающего инварианта* – замкнутой формулы, связывающей состояния машин M_S и M_T.

Отображение θ считается *сохраняющим информацию и семантику операций*, если машина M_S, соответствующая исходной модели, уточняет машину M_T, соответствующую целевой модели [32]. Уточнение доказывается интерактивно при помощи специальных программных средств [31].

В качестве иллюстрации основных принципов выражения семантики синтетической графовой модели и языка СИНТЕЗ в AMN рассмотрим частичные (в связи с ограниченным объемом статьи) AMN-спецификации, соответствующие данным моделям. Нижеследующий текст организован следующим образом: приводятся последовательные части спецификации на языке AMN и сопровождаются комментариями.

Основные идеи представления семантики объектной модели языка СИНТЕЗ в языке AMN изложены в работе [32]. В настоящей статье рассматривается семантика специфических конструкций, необходимых для унификации графовых моделей.

Итак, *спецификация*, выражающая семантику объектной модели языка СИНТЕЗ, представляется в языке AMN конструкцией REFINEMENT:

```
REFINEMENT ObjectDM
```

Константы, необходимые для унификации графовой модели, объявлены в разделе CONSTANTS машины *ObjectDM* и типизируются в разделе PROPERTIES:

```
CONSTANTS
c_edges, c_vertices,
a_startVertex,  a_endVertex,
c_edges_instance_type
PROPERTIES …
```

Раздел PROPERTIES содержит формулу, которая состоит из предикатов, типизирующих константы Предикаты соединяются операцией конъюнкции. Так, имена классов ребер и вершин представлены константами *c_edges* и *c_vertices,* тип которых – подмножество множества строк (*STRING_Type*):

```
c_edges: STRING_Type &
c_vertices: STRING_Type
```

Знак типизации «:» формально означает принадлежность элемента множеству.

Имя типа экземпляров класса ребер представлено константой *c_edges_instance_type*:

```
c_edges_instance_type: STRING_Type
```

Идентификаторы атрибутов этого типа, соответствующих исходящей и входящей вершинам ребра, представляются константами *a_startVertex, a_endVertex*, тип которых – натуральное число (*NAT*):

```
a_startVertex: NAT &
a_endVertex: NAT
```

Переменные, составляющие пространство состояний объектной модели, объявлены в разделе ABSTRACT_VARIABLES машины *ObjectDM* и типизируются в разделе INVARIANT:

```
ABSTRACT_VARIABLES
```

```
    m_directed, m_restricted,
    m_startVertexType,   m_endVertexType,
    isValidEdge
  INVARIANT …
```

Раздел INVARIANT содержит формулу, которая состоит из предикатов, типизирующих переменные состояния, и налагающих различные совместные ограничения на переменные и константы. Предикаты соединяются операцией конъюнкции.

Так, декларируется, что *c_edges* и *c_vertices* действительно являются именами классов:

```
c_edges: classNames &
c_vertices: classNames
```

Здесь *classNames* – множество, содержащее имена всех классов базы данных [32].

Метаинформация, связанная с типом экземпляров класса ребер, представлена переменными *m_directed* (направленность ребра), *m_restricted* (определенность типов вершин ребра), *m_startVertexType* (тип исходящей вершины), *m_endVertexType* (тип входящей вершины):

```
m_directed: subclasses(c_edges) --> BOOL &
m_restricted: subclasses(c_edges) --> BOOL &
m_startVertexType:
   subclasses(c_edges) --> subclasses(c_vertices) &
m_endVertexType:
   subclasses(c_edges) --> subclasses(c_vertices)
```

Переменные типизированы полными функциями (знак -->), определенными на множестве всех классов ребер (которые являются подклассами класса всех вершин *c_edges*). Функция *subclasses* ставит в соответствие классу множество его подклассов.

Декларируется, что *c_edges_instance_type* действительно является именем типа, а атрибуты *a_startVertex* и *a_endVertex* являются атрибутами этого типа. Декларируется также, что тип значений данных атрибутов – абстрактный тип данных (*ADT*):

```
c_edges_instance_type: typeNames &
a_startVertex: typeAttributes(c_edges_instance_type) &
a_endVertex: typeAttributes(c_edges_instance_type) &
attributeType(a_startVertex) = ADT &
attributeType(a_endVertex) = ADT
```

Здесь функция *typeAttributes* возвращает множество атрибутов типа, функция *attributeType* – тип значений атрибута [32].

Предикат смежности вершин и ребер представляется функцией *isValidEdge*, сопоставляющей ребру *edg* и двум вершинам v_1, v_2 значение *истина*, если вершины v_1, v_2 соединены ребром *edg*:

```
isValidEdge: objectsOfClass(c_edges)*
   objectsOfClass(c_vertices) *
   objectsOfClass(c_vertices) --> BOOL
!(edg, v1, v2).(edg: objectsOfClass(c_edges) &
   v1: objectsOfClass(c_vertices) &
   v2: objectsOfClass(c_vertices) =>
   ((isValidEdge(edg, v1, v2) = TRUE) <=>
   (adtAttributeValue(a_startVertex)(edg) = v1 &
   adtAttributeValue(a_endVertex)(edg) = v2) ) )
```

Здесь * - знак декартова произведения множеств. Функция *adtAttributeValue(a)(o)* возвращает значение атрибута *a* объекта *o* [32].

Дополнительные необходимые свойства переменных состояния представлены конъюнктивными компонентами инварианта. Так, ребро обязательно связывает два объекта из класса *c_vertices* (вершины):

```
!edg.(edg: objectsOfClass(c_edges) =>
   adtAttributeValue(a_startVertex)(edg):
         objectsOfClass(c_vertices) &
   adtAttributeValue(a_endVertex)(edg):
         objectsOfClass(c_vertices) )
```

Здесь «!» – знак квантора всеобщности, «=>» – логическая импликация. Функция *objectsOfClass* возвращает множество объектов – экземпляров класса [32].

Если типы вершин, соединяемых ребром, определены, то они должны принадлежать классам, задаваемым метаатрибутами *startVertexType* и *endVertexType* класса ребра:

```
!(cls, edg).(cls: subclasses(c_edges) &
  edg: objectsOfClass(cls) =>
  (m_restricted(cls) = TRUE =>
  adtAttributeValue(a_startVertex)(edg):
        objectsOfClass(m_startVertexType(cls)) &
  adtAttributeValue(a_endVertex)(edg):
        objectsOfClass(m_endVertexType(cls)) ) )
```

Из всего ЯМД в спецификации рассмотрена единственная операция *deleteVertex* удаления вершины:

```
OPERATIONS
deleteVertex(attr, cond) =
PRE attr : dom(attributeNames) &
cond : INT --> BOOL &
attributeType(attr) = Integer
THEN
objectsOfClass(c_vertices) :=
objectsOfClass(c_vertices) -
{ vert | vert: objectsOfClass(c_vertices) &
vert: dom(adtAttributeValue(attr)) &
cond(integerAttributeValue(attr)(vert)) = TRUE }
END
```

Параметрами операции являются идентификатор целочисленного атрибута *attr* и функция *cond*, отвечающая условию на значение атрибута. Операция *deleteVertex* удаляет из класса *c_vertices* все такие вершины *vert*, что на *vert* определен атрибут *attr*, и для значения этого атрибута выполнено условие *cond*. Здесь знак «:=» означает присваивание, знак «-» - разность множеств; конструкция *{v | F(v)}* – выделение множества таких значений *v*, что предикат *F(v)* обращается в истину, функция *integerAttributeValue(a)(o)* возвращает значение целочисленного атрибута *a* объекта *o*.

Спецификация, выражающая семантику синтетической графовой модели, представляется в языке AMN конструкцией

```
REFINEMENT GraphDM
```

Переменные, составляющие пространство состояний объектной модели, объявлены в разделе ABSTRACT_VARIABLES машины *GraphDM*:

```
ABSTRACT_VARIABLES
vertexTypeIDs, edgeTypeIDs, attributeIDs,
typeName, attributes, attributeName, attributeTyping,
directed, restricted, headType, tailType,
vertices, vertixType, edges, edgeType,
headVertix, tailVertix,
g_integerAttributeValue
```

Идентификаторы типов вершин представлены переменной *vertexTypeIDs*; идентификаторы типов ребер - переменной *edgeTypeIDs*; идентификаторы атрибутов – переменной *attributeIDs*; имена типов - переменной *typeName*; принадлежность атрибутов типам – переменной *attributes*; имена атрибутов – переменной *attributeName*; типы значений атрибутов – переменной *attributeTyping*; направленность ребер - переменной *directed*; определенность типов исходящей и входящей вершиной ребра - переменными *restricted*, *headType, tailType*; вершины и ребра, составляющие базу данных - переменными *vertices, edges*; типы конкретных вершин и ребер - переменными *vertixType, edgeType;*исходящая и входящая вершины конкретных ребер - переменными *headVertix, tailVertix;* значения целочисленных атрибутов - переменной *g_integerAttributeValue.* Функции, представляющие значения атрибутов других типов (например, *BOOL* или *STRING*), определяются аналогично.

Переменные типизируются в разделе INVARIANT при помощи частичных (знак «+->») и тотальных функций аналогично переменным, использующимся для придания семантики объектной модели:

```
INVARIANT
vertexTypeIDs: POW(NAT) &
edgeTypeIDs: POW(NAT) &
attributeIDs: POW(NAT) &
```

```
typeName: vertexTypeIDs \/ edgeTypeIDs -->
  STRING_Type &
attributes: vertexTypeIDs \/ edgeTypeIDs -->
  POW(attributeIDs) &
directed: edgeTypeIDs --> BOOL &
restricted: edgeTypeIDs --> BOOL &
headType: edgeTypeIDs +-> vertexTypeIDs &
tailType: edgeTypeIDs +-> vertexTypeIDs &
attributeName: attributeIDs --> STRING_Type &
attributeTyping: attributeIDs --> BuiltInTypes &
vertices: POW(NAT) &
vertixType: vertices --> vertexTypeIDs &
edges: POW(NAT) &
edgeType: edges --> edgeTypeIDs &
headVertix: edges --> vertices &
tailVertix: edges --> vertices &
g_integerAttributeValue:
  (vertices \/ edges)*attributeIDs +-> INT
```

Здесь знак «\/» означает объединение множеств.

Дополнительные необходимые свойства переменных состояния представлены конъюнктивными компонентами инварианта. Так, для тех типов, метаатрибут *restricted* которых принимает значение *TRUE*, заданы типы исходящей и входящей вершин:

```
!(type).(type: edgeTypeIDs =>
  (restricted(type) = TRUE =>
        type: dom(headType) & type: dom(tailType)) &
  (restricted(type) = FALSE =>
        type /: dom(headType) & type /: dom(tailType)) )
```

Здесь функция *dom* возвращает область определения функции, знак «/:» означает непринадлежность элемента множеству.

Функция *g_integerAttributeValue* определена только для целочисленных атрибутов. Атрибут, для которого определена эта функция, принадлежит типу соответствующей вершины или ребра:

```
!(vert, attr).(vert: vertices & attr: attributeIDs =>
  ((vert |-> attr) : dom(g_integerAttributeValue) =>
        attributeTyping(attr) = Integer) &
        attr: attributes(vertixType(vert))  ) &
!(edg, attr).(edg: edges & attr: attributeIDs =>
  ((edg |-> attr) : dom(g_integerAttributeValue) =>
       attributeTyping(attr) = Integer) &
       attr: attributes(edgeType(edg))  )
```

Если типы вершин, соединяемых ребром, определены (значение метаатрибута *restricted* типа этого ребра принимает значение *TRUE*), то они должны принадлежать типам, задаваемым метаатрибутами *headType* и *tailType* типа ребра:

```
!edg.(edg: edges =>
  (restricted(edg) = TRUE =>
        vertixType(headVertix(edg)) =
          headType(edgeType(edg)) &
        vertixType(tailVertix(edg)) =
          tailType(edgeType(edg))   )   )
```

Аналогично объектной модели рассмотрена единственная операция ЯМД – операция удаления вершины *deleteVertex*:

```
OPERATIONS
deleteVertex(attr, cond) =
PRE attr: attributeIDs & cond: INT --> BOOL &
attributeTyping(attr) = Integer
THEN
vertices := vertices -
{vert | vert: vertices &
  attr: attributes(vertixType(vert)) &
  cond(g_integerAttributeValue(vert, attr)) = TRUE }
END
```

Сигнатура операции совпадает с сигнатурой операции объектной модели. Семантика операции также аналогична: вершина *vert* удаляется из базы

данных (множества *vertices*), если на *vert* определен атрибут *attr*, и для значения этого атрибута выполнено условие *cond.*

Для формального доказательства того, что машина *GraphDM* уточняет машину *ObjectDM,* необходимо построить *инвариант уточнения*, связывающий переменные машин и добавить его к инварианту уточняющей машины.

Инвариант формализует принципы отображения ЯОД, изложенные в разделе 3.1 и объединяет их в одну конъюнкцию.

Множество имен типов графовой модели совпадает с множеством имен классов объектной модели (за исключением предопределенных классов *c_edges, c_vertices*):

```
ran(typeName) = classNames - {c_edges, c_vertices}
```

Множество атрибутов типов графовой модели соответствует множеству атрибутов объектной модели (за исключением предопределенных атрибутов *a_startVertex, a_endVertex*):

```
attributeIDs = dom(attributeNames) - {a_startVertex, a_endVertex}
```

Имена и типы атрибутов графовой и объектной модели совпадают:

```
!attr.(attr: attributeIDs =>
   attributeName(attr) = attributeNames(attr) &
   attributeTyping(attr) = attributeType(attr)  )
```

Вершины и ребра графовой базы данных соответствуют объектам классов *c_vertices* и c_*edges*:

```
vertices = objectsOfClass(c_vertices) &
edges = objectsOfClass(c_edges)
!vert.(vert: vertices =>
 ((vert: objectsOfClass(typeName(vertixType(vert)))) <=>
 (vert: vertices))  ) &
!edg.(edg: edges =>
   ((edg: objectsOfClass(typeName(edgeType(edg)))) <=>
   (edg: edges))  )
```

Значения атрибутов вершин и ребер графовой модели совпадают со значениями соответствующих атрибутов соответствующих объектов:

```
!(vert, attr).(vert: vertices & attr: attributeIDs =>
   ((vert |-> attr) : dom(g_integerAttributeValue) =>
         g_integerAttributeValue(vert, attr) =
           integerAttributeValue(attr)(vert))    )
!(edg, attr).(edg: edges & attr: attributeIDs =>
   ((edg |-> attr) : dom(g_integerAttributeValue) =>
         g_integerAttributeValue(edg, attr) =
           integerAttributeValue(attr)(edg))   )
```

Для указания того, что машина *GraphDM* уточняет машину *ObjectDM*, в машину *GraphDM* была добавлена директива

REFINES ObjectDM

Спецификации *ObjectDM* и *GraphDM* вместе с инвариантом уточнения были загружены в инструментальное средство Atelier B [31]. Автоматически были сгенерированы теоремы, выражающие уточнение спецификаций. В частности, для операции *deleteVertex* были сгенерированы 15 теорем, все они были доказаны также автоматически.

6. Родственные исследования и направления дальнейшей работы

Известно сравнительно небольшое количество работ, в которых исследуются вопросы интеграции или отображения графовых моделей данных. Например, в работе [33] язык запросов над гиперграфами используется для описания взглядов при интеграции графовых баз данных в посредниках. В работе [34] гиперграфовая модель также используется для интеграции графовых баз данных. Предлагается набор операций в рамках гиперграфовой модели для преобразования схемы ресурса в федеративную схему. В данных работах вопрос модельной неоднородности не встает, так как и в качестве канонической модели, и в качестве модели ресурсов выступает гиперграфовая модель. В работе [35] рассматривается отображение реляционной модели в гиперграфовую и императивная реализация операций реляционной алгебры в гиперграфовой

модели. Таким образом, в качестве канонической модели также выступает гиперграфовая модель, а в качестве модели ресурса – реляционная.

В области интеграции графовых баз данных существует еще одна группа работ, в которых рассматриваются вопросы поглощения запросов, ответа на запросы и переписывания запросов с использованием взглядов (представлений). Текущие результаты в данной области изложены в работе [26]. Получены верхние границы сложности ответа на запросы, переписывания запросов с использованием GLAV-взглядов (Global and Local As View) в графовых моделях; доказана разрешимость поглощения запросов.

Основные особенности настоящей работы состоят в следующем. Целью работы является устранение модельной неоднородности современных графовых СУБД для дальнейшей их виртуальной или материализованной интеграции. В качестве исходной модели при отображении используется синтетическая модель, структуры данных которой покрывают возможности современных СУБД, основанных на простых и атрибутированных графах. В качестве целевой модели используется каноническая объектно-фреймовая модель - язык СИНТЕЗ. Для отображения обеспечивается формальное доказательство сохранения информации и семантики операций ЯМД.

Дальнейшая работа включает следующие этапы:

- выбор конкретных графовых моделей, основанных на простых и атрибутированных графах и построение трансформаций, реализующих изложенное отображение;
- расширение инструментальных средств поддержки предметных посредников для виртуальной интеграции графовых баз данных;
- применение технологии предметных посредников для решения научных задач в некоторой предметной области над множеством неоднородных ресурсов, включающим графовые базы данных.

7. Заключение

Новая парадигма, объединяющая теорию, эксперимент и симуляцию в науке и информационных технологиях, связанных с интенсивным использованием данных, требует создания методов и средств оперирования данными, объемы которых выходят за рамки возможностей современных технологий баз данных, а также разработки новых подходов, позволяющих справляться с разнообразием массово и хаотично развивающихся языков и моделей данных. В данной работе рассматривается проблема унификации важного класса нетрадиционных моделей данных – графовых моделей. Унификацией называется отображение модели данных в каноническую информационную модель, служащую общим языком в среде разнообразных моделей ресурсов, сохраняющее информацию и семантику операций языка манипулирования данными. Графовые базы данных используются в различных приложениях: социальных сетях, рекомендательных системах, геопространственных приложених, в системах обнаружения мошенничества. Современные графовые СУБД обычно ориентированы на обработку и анализ больших графов, насчитывающих миллиарды вершин и ребер. Ведущие интернет-компании, такие, как Facebook и Google, интенсивно используют графовые СУБД.

В работе рассмотрены основные свойства графовых моделей, распространенные приложения графовых СУБД. Осуществлен обзор современных моделей и инфраструктур графовых СУБД.

Для обеспечения общности подхода по унификации графовых моделей в работе предложена синтетическая модель данных атрибутированных графов, покрывающая возможности моделей данных таких известных систем, как, например, Neo4j, Dex, InfiniteGraph, OrientDB, VertexDB, Filament, OQGraph, Horton, InfoGrid. В качестве ЯМД синтетической модели рассматривается декларативный язык Cypher, развиваемый в системе Neo4j.

Рассмотрены и проиллюстрированы принципы отображения ЯОД и ЯМД модели атрибутированных графов в каноническую информационную мо-

дель (объектно-фреймовый язык СИНТЕЗ). Рассмотрены также вопросы доказательства сохранения информации и семантики операций при отображении графовых моделей в объектные с использованием формального языка спецификаций AMN. Осуществлен обзор родственных решений по интеграции или отображению графовых моделей данных.

Список литературы

1. The Fourth Paradigm: Data-Intensive Scientific Discovery. Eds. Tony Hey, Stewart Tansley, and Kristin Tolle. – Redmond: Microsoft Research, 2009.
2. Захаров В.Н., Калиниченко Л.А., Соколов И.А., Ступников С.А. Конструирование канонических информационных моделей для интегрированных информационных систем // Информатика и ее применения. – М.: ИПИ РАН, 2007. – Т. 1, Вып. 2. – С. 15-38.
3. Kalinichenko L.A., Stupnikov S.A., Martynov D.O. SYNTHESIS: a Language for Canonical Information Modeling and Mediator Definition for Problem Solving in Heterogeneous Information Resource Environments. – Moscow: IPI RAN, 2007. – 171 p.
4. Ступников С.А., Скворцов Н.А., Будзко В.И., Л. А. Калиниченко. Методы унификации нетрадиционных моделей данных. // Системы высокой доступности. – 2014.
5. R. Angles. A Comparison of Current Graph Database Models. // Proc. of IEEE 28th International Conference on Data Engineering Workshops (ICDEW). – 2012. – P. 171-177.
6. Robinson I., Webber J., Eifrem E. Graph Databases. - O'Reilly Media, 2013. – 212 p.
7. Iordanov B. Hypergraphdb: a generalized graph database. // Proc. 2010 International Conference on Web-age information management (WAIM). – Springer-Verlag, 2010. – P. 25-36.

8. Malewicz G., Austern M.H., Bik A.J.C., Dehnert J.C., Horn I., Leiser N., Czajkowski G. Pregel: A System for Large-Scale Graph Processing. // Proc. of the 2010 ACM SIGMOD International Conference on Management of Data. – 2010. – P. 135-145.
9. Apache Giraph Project. – URL: http://giraph.apache.org/ (дата обращения: 05.02.2014).
10. Hadoop Project. – URL: http://hadoop.apache.org/ (дата обращения: 05.02.2014).
11. Shao B., Wang H., Li Y. Trinity: a distributed graph engine on a memory cloud. // Proc. of the 2013 ACM SIGMOD International Conference on Management of Data. – 2013. – P. 505-516.
12. Kyrola A., Blelloch G., Guestrin C. Graphchi: Large-scale graph computation on just a PC. // 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI). – Berkeley: USENIX, 2012. – P. 31–46. – ISBN: 978-1-931971-96-6.
13. Trinity Manual. – Microsoft Research Asia, 2012. – 68 p. –URL: http://research.microsoft.com/en-us/projects/trinity/trinitymanual.pdf (дата обращения: 05.02.2014).
14. Shao B., Wang H., Xiao Y. Managing and mining large graphs: systems and implementations. // Proc. of the 2012 ACM SIGMOD International Conference on Management of Data. – 2012. – P. 589-592.
15. Sun Z., Wang H. (Hongzhi), Wang H. (Haixun), Shao B., Li J. Efficient Subgraph Matching on Billion Node Graphs. // Proceedings of the VLDB Endowment. – 2012. – Vol. 5, Iss. 9. – P.788-799.
16. Neo4j Graph Database. – URL: http://www.neo4j.org/ (дата обращения: 05.02.2014).
17. The Dex Graph Database Management System. – URL: http://www.sparsity-technologies.com/dex.php (дата обращения: 05.02.2014).

18. Sarwat M., Elnikety S., He Y., Kliot G. Horton: Online Query Execution Engine for Large Distributed Graphs. // Conference: International Conference on Data Engineering (ICDE). – 2012. – P. 1289-1292.
19. Bykov S., Geller A., Kliot G., Larus J., Pandya R., Thelin J. Orleans: Cloud Computing for Everyone. // Proc. of the 2nd ACM Symposium on Cloud Computing (SOCC). – ACM, 2011.
20. Titan Project. – URL: https://github.com/thinkaurelius/titan/wiki (дата обращения: 05.02.2014).
21. Jouili S., Vansteenberghe V. An Empirical Comparison of Graph Databases. // International Conference on Social Computing (SocialCom). – 2013. – P. 708-715.
22. Montag D.. Understanding Neo4j Scalability. – Neo Technology, 2013. – URL: http://info.neotechnology.com/rs/neotechnology/images/Understanding%20Neo4j%20Scalability(2).pdf (дата обращения: 05.02.2014).
23. The Neo4j Manual. – 2013. – URL: http://docs.neo4j.org/ (дата обращения: 05.02.2014).
24. RDF Primer. W3C Recommendation 10 February 2004. Eds. F. Manola, E. Miller. – W3C, 2004. – URL: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/ (дата обращения: 05.02.2014).
25. Скворцов Н. А. Отображение модели данных RDF в каноническую модель предметных посредников // Электронные библиотеки: перспективные методы и технологии, электронные коллекции: Труды XV Всероссийской научной конференции RCDL'2013 (Ярославль, 14-17 октября 2013). – Ярославль: ЯрГУ им. П. Г. Демидова, 2013. – С. 202-209. – ISBN: 978-5-8397-1004-7.
26. Calvanese D., Giacomo G., Lenzerini M., Vardi M. Query Processing under GLAV Mappings for Relational and Graph Databases. // Proc. of the VLDB Endowment. – 2012. – Vol. 6, No. 2. – P.61-72.
27. Angles R., Gutierrez C. Survey of Graph Database Models. // ACM Computing Surveys. – 2008. – Vol. 40, No. 1.
28. Wood P.T. Query languages for graph databases. // ACM SIGMOD Record. – 2012. – Vol. 41, Iss. 1. – P. 50-60.

29. Kalinichenko L.A., Briukhov D.O., Martynov D.O., Skvortsov N.A., Stupnikov S.A. Mediation Framework for Enterprise Information System Infrastructures. // Proc. of the 9th International Conference on Enterprise Information Systems ICEIS 2007. - Funchal, 2007. – Vol.: Databases and Information Systems Integration. - P. 246-251.
30. Abrial J.-R. The B-Book: Assigning Programs to Meanings. – Cambridge: Cambridge University Press, 1996.
31. Atelier B, the industrial tool to efficiently deploy the B Method. – URL: http://www.atelierb.eu/index-en.php (дата обращения: 05.02.2014).
32. Ступников С. А. Унификация модели данных, основанной на многомерных массивах, при интеграции неоднородных информационных ресурсов. // Труды RCDL'2012. – Переславль-Залесский: Университет города Переславля, 2012. – С. 67-77.
33. Theodoratos D. Semantic Integration and Querying of Heterogeneous Data Sources Using a Hypergraph Data Model // BNCOD 2002. Lecture Notes in Computer Science. – Vol. 2405. – Springer, 2002. – P. 166-182.
34. Sundaresan S., Hu G: Schema integration of distributed databases using hypergraph data model. // Proc. of Information Reuse and Integration Conf, IRI 2005. – 2005. – P. 548-553. – ISBN: 0-7803-9093-8.
35. Tahat A., Ling M.H.T. Mapping Relational Operations onto Hypergraph Model // Proc. of CoRR 2011. - The Python Papers, 2011. – Vol. 6 Iss. 1. – P. 1.
36. Kalinichenko L.A., Stupnikov S.A. Heterogeneous information model unification as a pre-requisite to resource schema mapping // A. D'Atri and D. Saccà (eds.), Information Systems: People, Organizations, Institutions, and Technologies (Proc. of the V Conference of the Italian Chapter of Association for Information Systems itAIS). – Berlin-Heidelberg: Springer Physica Verlag, 2010. – P. 373-380.

Реферат

Развитие методов и средств оперирования большими данными требует разработки новых подходов, позволяющих справляться с разнообразием массово и хаотично развивающихся моделей данных и их языков. Для интеграции неоднородных информационных ресурсов необходима унификация их моделей данных - отображение в каноническую информационную модель (служащую общим языком в среде разнообразных моделей ресурсов), сохраняющее информацию и семантику языка определения данных и операций языка манипулирования данными. Данная работа посвящена унификации одного из важных и быстро развивающихся видов моделей, называемых графовыми. Рассматриваются основные современные графовые модели, их черты и области применения, особенности использования графовых моделей при манипулировании большими данными. На основе анализа обзора состояния современных графовых СУБД сделан вывод, что большинство существующих баз данных основаны на простых или атрибутированных графах (attributed graph или property graph), в которых атрибуты (свойства) приписываются ребрам и/или вершинам графа. Именно такие модели и были выбраны в данной статье в качестве исходных, подлежащих унификации моделей. Рассмотрены вопросы доказательства сохранения информации и семантики операций при отображении графовых моделей в объектно-фреймовую каноническую модель с использованием формального языка спецификаций AMN.

Mapping of Graph Data Models into a Canonical Model for the Development of Data Intensive Systems

Abstract

To integrate the heterogeneous information resources containing Big Data it is required to *unify* their data models by mapping them into the canonical information

model preserving information and semantics of their data description language and semantics of the operations of their data manipulation language. This research is devoted to the unification of the *graph* models – the important kind of the existing various data models. The essential modern graph models are discussed including their features, application domains, peculiarities of their use under big data manipulation. The issues of proof of information and operation semantics preserving by the mapping of the graph models into the object-frame canonical model applying the formal specification language AMN are considered.

Summary

The development of methods and facilities for operating of Big Data requires new approaches providing for coping with the diversity of various chaotically developed data models and their languages. For integration of the heterogeneous information resources the unification of their data models is required by their mapping into the canonical information model (serving as the generalized language in the environment of various resource models). Such mappings should preserve information and semantics of their data description language and semantics of the operations of their data manipulation language. This research is devoted to the unification of the *graph* models – the important and fastly developing kind of the existing data models. The essential modern graph models are discussed including their features, application domains, peculiarities of their use under big data manipulation. On the basis of the survey of the existing graph DBMSs the paper concludes that the majority of the existing graph databases are based on the property or attributed graphs in which attributes are assigned to the vertices and edges of the graph. The generalization of such models has been selected in this paper as the source data models to be unified. The issues of proof of information and operation semantics preserving by the mapping of such graph models into the object-frame canonical model applying the formal specification language AMN are considered.