Методы унификации нетрадиционных моделей данных[1]

УДК 004.652

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

Н. А. Скворцов – н.с. ИПИ РАН, E-mail: nskv@ipi.ac.ru

В. И. Будзко – д.т.н., профессор, зам. директора по научной работе ИПИ РАН, E-mail: vbudzko@ipiran.ru

В.Н.Захаров – д.т.н., ученый секретарь ИПИ РАН, E-mail: VZakharov@ipiran.ru

Л. А. Калиниченко – д.ф.-м.н., профессор, зав. лабораторией, ИПИ РАН, E-mail: leonidandk@gmail.com

Одной из нерешенных проблем манипулирования большими данными является проблема интеграции различных нетрадиционных моделей данных (отличающихся от реляционной), для решения которой необходимо прежде всего создание унифицированного представления различных видов нетрадиционных моделей в канонической информационной модели. Такое представление требует построения отображения моделей данных, сохраняющего семантику их языков определения данных и манипулирования данными, в каноническую модель. В статье рассматриваются принципы отображения четырех видов нетрадиционных моделей данных в объектно-фреймовую каноническую модель: модели данных, основанные на многомерных массивах; графовые модели данных; модели данных NoSQL; триплетная модель данных (RDF). Унификация нетрадиционных моделей служит базой для материализованной или виртуальной интеграции соответствующих коллекций данных.

[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».

Ключевые слова: нетрадиционные модели данных, большие данные, отображения моделей данных, интеграция информационных ресурсов

Keywords: non-traditional data models, big data, data model mappings, information resource integration

1. Введение

В настоящий период развития ИТ во главу угла поставлено создание средств анализа данных, накапливаемых в Вебе, в социальных сетях, в виде машинных и сенсорных данных, в виде логов серверов, текстовых данных, изображений, данных, полученных в результате наблюдений и измерений разнообразными высокотехнологичными инструментами. Данные таких масштабов (часто измеряемых уже в петабайтах) относят к категории «больших данных» (Big Data) [1], они плохо поддаются обработке и анализу в рамках хорошо известных технологий баз данных, опирающихся в основном на реляционную модель данных или ее производные (объектно-реляционная модель).

Для представления и манипулирования коллекциями подобных данных созданы новые модели данных, отличающиеся от традиционных (реляционных). Современные платформы ИТ включают специальные средства управления большими данными в кластерных и параллельных инфраструктурах. Одной из нерешенных проблем манипулирования большими данными является проблема интеграции данных различных моделей, решение которой требует прежде всего создания унифицированного представления различных видов нетрадиционных моделей в *канонической информационной модели* (общем языке, унифицирующем языки разнообразных моделей данных) [2]. Такое представление требует построения отображения моделей данных, сохраняющего семантику их языков определения данных и манипулирования данными, в каноническую модель. Наличие отображения необходимо как для материализованной

интеграции (создание хранилища данных), так и для виртуальной интеграции (посредством предметных посредников).

При материализованной интеграции предполагается создание хранилища данных (warehouse), в которое загружаются коллекции данных, подлежащие интеграции. В процессе загрузки происходит преобразование данных из схемы коллекции в общую схему хранилища.

Виртуальная же интеграция рассматривается в статье в терминах сред предметных посредников [3]. Среды предметных посредников образуют промежуточный слой между пользователем (приложением) и неоднородными информационными ресурсами. При этом данные из ресурсов не материализуются в посреднике. Федеративная схема посредника, описывающая некоторую предметную область, создается независимо от существующих ресурсов в терминах задач конкретной предметной области. Ресурсы, релевантные предметной области, затем регистрируются в посреднике – их схемы связываются специальными семантическими отображениями с федеративной схемой. Исполнительная среда посредников предоставляет возможность пользователям (приложениям) задавать запросы (программы) к посреднику в терминах федеративной схемы. Эти запросы переписываются средой посредников в частичные запросы над информационными ресурсами, затем исполняются на ресурсах. Результаты частичных запросов объединяются и выдаются пользователю также в терминах федеративной схемы.

Необходимым предусловием интеграции ресурсов, основанных на нетрадиционных моделях данных, является построение отображений соответствующих моделей в каноническую модель данных, сохраняющих информацию и семантику операций языка манипулирования данными (ЯМД) [2]. Это обусловлено тем, что семантические отображения, связывающие федеративную схему и схемы ресурсов, нужно проводить в единой (канонической модели) [4]. Отображение должно быть верифицируемым – доказуемо правильным.

Заметим, что для установления семантических соотношений между схемами ресурсов и федеративной схемой необходимо отображение языка опреде-

ления данных (ЯОД) исходной модели в каноническую, но ЯМД канонической модели, напротив, необходимо отображать в ЯМД исходной модели. Это связано с тем, что запросы к посреднику в канонической модели необходимо отображать в запросы к ресурсам.

Отметим отличие виртуальной и материализованной интеграции. При виртуальной интеграции отображение ЯМД обеспечивает возможность трансляции программ на языке посредника в запросы на языке ресурсов.

В случае материализованной интеграции данные извлекаются из ресурса и представляются в хранилище в канонической модели. При этом программы на ЯМД канонической модели исполняются непосредственно на данных в хранилище. Отображение ЯМД нужно лишь для того, чтобы убедиться, что отображение моделей сохраняет информацию и семантику операций, т.е. является семантически правильным. Отображение моделей служит базой для процесса Извлечения-Преобразования-Загрузки (ETL), формирующего из данных ресурса данные хранилища: ETL-процесс может быть выражен только в терминах канонической модели.

В качестве канонической модели в данной работе рассматривается язык СИНТЕЗ [5], поддерживающий комбинированную слабоструктурированную (фреймовую) и объектную модель данных, нацеленную на разработку предметных посредников или хранилищ данных для решения задач в средах неоднородных ресурсов. Объектные модели хорошо зарекомендовали себя при унификации различных классов моделей – структурированных, онтологических, сервисных, процессных [3]. Разработан прототип программных средств для поддержки среды предметных посредников с языком СИНТЕЗ в роли канонической модели [6]. С точки зрения предметных посредников СУБД, основанные на нетрадиционных моделях данных, поддерживают новые виды ресурсов, подлежащих интеграции в посредниках вместе с привычными ресурсами – реляционными и объектными базами данных, веб-сервисами и т.д.

В статье рассматриваются принципы и методы отображения следующих видов нетрадиционных моделей данных в каноническую модель:

- модели данных, основанные на многомерных массивах (array-based data models);
- графовые модели данных;
- модели данных NoSQL;
- триплетная модель данных (RDF).

На примерах иллюстрируется метод верификации отображений – доказательства сохранения информации и семантики операций при отображении.

Целью работы является определение хорошо обоснованных унифицирующих отображений нетрадиционных моделей данных в каноническую, представление возможности унифицированного представления столь различных моделей для материализованной или виртуальной интеграции соответствующих коллекций данных. Интеграция данных занимает важное место в стеках программных решений всех ведущих вендоров – IBM, Oracle, SAP, Informatica, SAS и других. Стеки обычно включают:

- решения по сбору, распределенному хранению и обработке больших данных (в инфраструктуре Hadoop или ей подобных);
- средства организации хранилищ данных (data warehouses) и витрин данных (data marts), в том числе средства ETL; средства виртуальной интеграции данных;
- средства аналитики на различных уровнях стека, в том числе и средства самого высокого уровня, называемые обычно *бизнес-аналитикой* (business intelligence).

Методы унификации нетрадиционных моделей данных необходимы в тех случаях и на тех уровнях стека, где происходит интеграция разномодельных данных. Такая интеграция может осуществляться как на уровне хранилищ данных и систем виртуальной интеграции, так и на уровне распределенных инфраструктур хранения и обработки. Однако, в существующих промышленных ре-

шениях осуществляется интеграция данных, представленных преимущественно в различных вариантах реляционной модели данных. Унификация нетрадиционных моделей данных представляется задачей систем интеграции данных следующих поколений. В частности, разработанные методы будут применяться в инфраструктуре предметных посредников, разрабатываемой в ИПИ РАН [3].

2. Нетрадиционные модели данных и их унификация

2.1. Модели данных, основанные на многомерных массивах

Одним из важных видов моделей данных, нацеленных на параллельную обработку и анализ данных в распределенных средах – гридах и облаках, являются модели данных, основанные на многомерных массивах (array-based data models или array data models) и называемые далее ММ-моделями. Родственными данным моделям являются так называемые «кубы данных», используемые в OLAP технологии [7][8]. Исследования ММ-моделей начались достаточно давно [9][10] и продолжают развиваться. В данном разделе рассматривается конкретная модель, а именно – модель, используемая в СУБД SciDB [11].

История SciDB начинается с 2007 года, когда на симпозиуме по Экстремально большим базам данных (XLDB) представителями науки и промышленности был сделан вывод, что существующие СУБД не в состоянии манипулировать объемами данных, которые появятся в ближайшем будущем [12]. Одним из примеров провайдеров таких данных является строящийся телескоп LSST (Large Synoptic Survey Telescope) [13]. Был также сделан вывод о необходимости разработки СУБД нового поколения, которая должна удовлетворять, в частности, следующим требованиям [12]:

- модель данных основывается на многомерных массивах, а не на кортежах отношений;
- модель хранения базируется на версионности, а не на обновлении значений;

- обеспечивается масштабируемость до сотен петабайт и высокая отказоустойчивость;
- СУБД является свободно распространяемым программным продуктом.

Некоторое время спустя был организован международный проект под руководством Майкла Стоунбрейкера, целью которого являлось создание новой системы управления базами данных, получившей название SciDB. В настоящее время свободно распространяется очередная версия этой системы для ОС Ubuntu и RedHat.

Нужно отметить, что модель данных SciDB подвергается некоторой критике со стороны исследователей, продолжающих развитие моделей, основанных на многомерных массивах. Так, авторы языка SciQL [14] отмечают, что язык ADM является смесью SQL и деревьев алгебраических операций. По их мнению, язык для СУБД, основанных на многомерных массивах, должен быть интегрирован с синтаксисом и семантикой SQL:2003. Несмотря на эти замечания, модель ADM представляет несомненный практический интерес для интеграции баз данных. SciDB используется как в научных проектах, связанных с LSST (предполагается после запуска телескопа), и физикой высоких энергий, так и в коммерческих, ориентированных на генетику, страхование, финансы. Сравнительное тестирование SciDB с СУБД Postgres и статистическим программным обеспечением R показало преимущества SciDB по производительности и масштабируемости.

2.1.1. Отображение модели ADM в каноническую информационную модель

SciDB поддерживает два языка для работы с массивами: AQL (Array Query Language) и AFL (Array Functional Language). AQL является SQL-подобным декларативным языком, включающим как операции ЯОД, так и операции ЯМД. AFL представляет собой функциональный язык манипулирования массивами, операции которого можно объединять в композиции. Допускается использование операций AFL в запросах AQL.

Операции языков и их представление в канонической модели будут иллюстрироваться на примерах из сценария применения SciDB в области оптической астрономии [15], а также на простых примерах из документации SciDB [16].

Основной единицей определения данных в модели ADM является массив, имеющий конечное количество *измерений* $d_1, d_2, ..., d_n$ [16]. Длиной измерения называется количество упорядоченных значений в этом измерении. По умолчанию типом измерения являются целые числа. Каждая комбинация значений измерений соответствует ячейке массива, которая может содержать конечное количество значений, называемых *атрибутами.* Типом атрибута может быть один из встроенных типов ADM [16].

Основная операция ЯОД ADM – создание массива - выглядит следующим образом:

```
CREATE ARRAY source
< ampExposureId: int64,
   filterId: int8,
   apMag: double >
[ ra(double), de(double), objectId ];
```

Создается массив оптических источников *source,* измерениями которого являются координаты *ra* и *de* типа *double* и целочисленный идентификатор объекта *objectId*. Ячейка массива состоит из трех атрибутов: идентификатора устройства, проводящего измерение светимости источника (*ampExposureId*), идентификатора фильтра, использованного при измерении (*filterId*), апертурной звездной величины (*apMag*).

В языке СИНТЕЗ создание массива представляется определением одноименного класса:

```
{ source; in: class;
  instance_type:{
  double ra;
  double de;
```

```
    long objectId;
    long ampExposureId;
    short filterId;
    double apMag;
    key: { unique; { ra, de, objectId } };
    definiteness: {obligatory;
      { ra, de, objectId, filterId, apMag } };
    };
  }
```

Как измерения, так и атрибуты, составляющие ячейку, представляются в языке СИНТЕЗ атрибутами типа экземпляров (*instance_type*) класса. Между встроенными типами ADM (*int8, int64, double* и т.д.) и встроенными типами языка СИНТЕЗ (*short, long, double*) устанавливается взаимно-однозначное соответствие. Совокупность атрибутов, соответствующих измерениям, объявляется уникальной (инвариант *key,* выражаемый встроенным утверждением *unique*). Объявляется также, что атрибуты, соответствующие измерениям и атрибутам ADM, должны быть определены у всех экземпляров класса (инвариант *definiteness,* выражаемый встроенным утверждением *obligatory*).

Таким образом обеспечивается сохранение отличительных свойств многомерных массивов («кубов данных»), существенным образом различающих измерения и атрибуты, составляющие ячейку:

- по набору значений измерений однозначно определяется набор значений атрибутов ячейки (уникальность измерений);
- ячейка массива всегда определяется полным набором значений измерений (определенность измерений).

Изложенный подход к отображению ЯОД может быть обобщен на случай, когда канонической является объектная или объектно-реляционная модель, отличная от языка СИНТЕЗ. Также, непринципиальным является выбор модели данных, основанной на многомерных массивах. Аналогичный подход может

быть применен, например, для моделей систем SciQL [14] или SciHadoop [17]. В общем виде, принципы отображения ЯОД выглядят следующим образом:

- массив отображается в коллекцию типизированных объектов (класс) объектной модели;
- измерения и атрибуты, составляющие ячейку массива, отображаются в атрибуты типа экземпляров класса;
- между встроенными типами модели, основанной на многомерных массивах, и встроенными типами объектной модели устанавливается взаимнооднозначное соответствие;
- совокупность атрибутов, соответствующих измерениям, объявляется уникальной (при помощи механизма ключей, утверждений или инвариантов);
- атрибуты, соответствующие измерениям и атрибутам ячейки массива, объявляются определенными (при помощи утверждений или инвариантов).

2.1.2. Отображение языка манипулирования данными

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

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

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. Предикаты, находящиеся в голове правил, могут быть использованы в телах других правил в качестве предикатов-коллекций.

В дальнейшем будет часто использоваться запись предиката-коллекции вида *source([ra, de])*. Неформально это означает, что нас не интересуют объек-

ты класса *source* целиком, а лишь их атрибуты *ra, de.* Формально запись означает сокращение от *source(_/source.inst[ra, de])* – конструкции языка СИНТЕЗ. Здесь знак _ обозначает анонимную переменную, *source.inst* – анонимный тип экземпляров (instance) класса *source*, *ra, de* – необходимые атрибуты типа экземпляров класса.

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

Подробно отображение ЯМД SciDB рассмотрено в работе [18]. В данном разделе мы ограничимся рассмотрением лишь нескольких конструкций языка.

Рассмотрим программу на языке AQL, извлекающую координаты (*ra, de*) и апертурную звездную величину (*apMag*) астрономических источников из класса *source* с условием на используемый фильтр (*filterId*) и апертурную звездную величину, причем запрос *q* использует результаты запроса *r.*

```
q([ra, de, apMag]) :- r([ra, de, apMag]), filterId= #filterId.
r([ra, de, apMag]) :- source([ra, de, apMag]), apMag >= #apMag.
```

Здесь *#filterId* и *#apMag* – константы соответствующих типов.

Такая программа представляется в AQL следующим запросом:

```
SELECT apMag FROM
  ( SELECT apMag FROM source WHERE apMag >= #apMag )
WHERE filterId = #filterId;
```

Простые условия отображаются в AQL без изменений, рекурсивные запросы представляются вложенными запросами. Заметим, что координаты *ra, de* не указываются в секции SELECT – они являются измерениями и извлекаются по умолчанию.

Рассмотрим запрос, изменяющий значения в квадратной матрице *matrix* на значения с обратным знаком в том случае, если модуль значения больше 5.

```
matrix(x/[i, j, val]) :-
```

```
  matrix(x/[i, j, val1/val]), abs(val) > 5, val = -val1.
```

В AQL данный запрос представляется следующим образом:

```
UPDATE source SET val = -val WHERE abs(val) > 5;
```

Рассмотрим принципы отображения конструкций языка СИНТЕЗ, соответствующих конструкциям AFL, на примере *расширения элементов массива в подмассивы.* Каждый элемент массива расширяется в подмассив определенного размера. Значения всех ячеек подмассива дублируют значение оригинальной ячейки. Пример программы, расширяющей каждую ячейку матрицы 3x3 в подматрицу 2x2:

```
q([i, j, val]) :- {x/[i, j, val] | exists y (
  source(y/[i1/i, j1/j, val]) &
  ( i = i1*2 & j = j1*2 | i = i1*2 +1 & j = j1*2 |
  i= i1*2 & j= j1*2 + 1 | i= i1*2 +1 & j= j1*2 +1) ) }.
```

Здесь выражение *{x/T | F(x)}*, где *F* – формула со свободной переменной *x*, обозначает конструкцию выделения множества; *exists* обозначает квантор существования.

В ADM запрос представляется с использованием операции *xgrid*:

```
SELECT * FROM xgrid(source, 2, 2);
```

Можно заметить, что операция AFL *xgrid* имеет достаточно сложно устроенный прообраз в канонической модели (это справедливо и для многих других операций). Между тем, эти операции являются естественными для массивов. Поэтому, при интеграции ресурсов, основанных на многомерных массивах, в канонической модели возможно использование специального класса *array*, инкапсулирующего специфические операции, характерные для многомерных массивов:

```
{ array; in: class;
  instance_type: {
  xgrid: { in: function;
    params: {
     +dimensions/{sequence; type_of_element: string;},
```

```
      +scales/{sequence; type_of_element: integer;}};
  }; };
}
```

В приведенном примере рассмотрена сигнатура единственной операции *xgrid*, параметрами которой являются последовательность имен измерений *dimensions* и последовательность масштабов увеличения по каждому из измерений *scales*. Параметром операции по умолчанию также считается класс *array* как коллекция объектов. При отображении ЯОД каждый класс - образ массива (например, класс *source* из раздела 2.1.1) становится подклассом класса *array*:

```
{ source; in: class; superclass: array;
  instance_type: { … };
}
```

Аналогично *xgrid,* операциями класса *array* могут быть представлены такие операции AFL, как *substitute, sort, multiply* и т.д.

Заметим, что решение о представлении операций, характерных для многомерных массивов, в рамках специального класса канонической модели допускает обобщение на объектные канонические модели, отличные от языка СИНТЕЗ, и модели, основанные на многомерных массивах, отличные от ADM (например, модели данных систем SciQL [14] или SciHadoop [17]).

2.2. Графовые модели данных

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») [19]. Целью графовых моделей было преодоление ограничений, налагаемых традиционными моделями данных, связанных с представлением исходных графовых структур данных.

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 графо-

вых 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.

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. Одни инфраструктуры основаны на вершинно-центрированной вычислительной модели [20][21], другие используют в качестве графового хранилища различные NoSQL базы данных [22], третьи используют собственную кластерную архитектуру [23].

Обзоры состояния современных графовых СУБД [19] 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 работе [53] в качестве исходных, подлежащих унификации моделей. Подробно описание отображения графовых моделей в каноническую модель данных выделено в отдельную статью, опубликованную в настоящем выпуске журнала [53]. Это связано с все возрастающим масштабом и спектром применений анализа больших графов, насчитывающих миллиарды вершин и ребер, во многих предметных областях - начиная с социальных сетей и до обнаружения мошенничества.

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 [24], развиваемый в системе Neo4j. Основные конструкции и ключевые слова имеют сходство с такими широко распространенными языками, как SQL и SPARQL.

Рассмотрены и проиллюстрированы принципы отображения ЯОД и ЯМД модели атрибутированных графов в каноническую информационную модель. Рассмотрены также вопросы доказательства сохранения информации и семантики операций при отображении графовых моделей в объектные с использованием формального языка спецификаций AMN [25].

2.3. Модели данных NoSQL

Появление класса систем баз данных, объединяемого термином NoSQL, продиктовано необходимостью работы с большими данными, требующими масштабируемости и отказоустойчивости при работе на кластерах. Технологии NoSQL направлены на обеспечение горизонтального масштабирования и доступа к данным сверхбольших объёмов или работу с высоконагруженными по чтению или обновлению данными, они часто обеспечивают более приемлемую производительность простых операций, чем реляционные системы баз данных. В период своего становления они выступили альтернативой традиционным реляционным базам данных для обеспечения производительности таких нагруженных проектов, как крупные социальные сети и поисковые системы в Веб. Системы баз данных NoSQL [26], в основном, основаны на следующих принципах работы с данными.

В NoSQL произошёл отказ от табличной организации данных, хранимых по кортежам, что позволяет избежать обращений к целым кортежам, если необ-

ходимо прочитать только некоторые их компоненты. Структура данных в таких моделях не связана жёсткими схемами. Они ориентированы на слабоструктурированные данные, динамическую модификацию их структуры, отсутствие ограничений предопределёнными типами и связями, произвольные, в том числе, неструктурированные значения. Для повышения производительности поиска необходимых данных они организованы в виде пар «ключ-значение», для доступа к значениям по ключам используется принцип хеш-таблиц. Для надёжности и физического распределения по узлам данные могут быть связаны с определённым узлом или реплицироваться на несколько узлов с помощью таких механизмов, как распределённые хеш-таблицы (DHT) [27].

Как и платформа управления большими данными Hadoop, системы больших баз данных NoSQL предполагают использование параллельных вычислений в кластерах. Некоторые системы баз данных используют реализации модели распределённых вычислений MapReduce. В частности, в системе MongoDB [29] есть собственная реализация операций MapReduce. Для обработки данных в системе HBase [28] может использоваться Hadoop MapReduce.

Так как обращения производятся только по ключам, то упрощается и язык манипулирования данными. Обычно используется набор операций, подобный CRUD (Create/Read/Update/Delete) [30]. В зависимости от организации ключей и значений на основе операций могут составляться SQL-подобные запросы, однако и они должны укладываться в обращение по ключам.

Обращение по ключам вносит существенные ограничения в возможностях запросов, которые, в результате, влияют на организацию данных. В большинстве систем возможные разновидности запросов, используемых при решении задач, должны быть известны заранее. Те данные, по которым необходимо организовать поиск, должны оказаться ключами в некоторых парах «ключ-значение», а искомые данные – значениями. Для этого формируются вспомогательные пары, дублирующие данные в разной организации ключей и значений. Фактически они реализуют над данными в основных парах материализованные взгляды, которые должны формироваться в соответствии с данными в основ-

ных парах при каждом их обновлении. Однако задача формирования таких взглядов и обеспечение их целостности возлагается не на систему баз данных, а на само приложение, обновляющее данные в базе. Таким образом, происходит сдвиг обработки запросов на этап обновления данных, что усложняет обновление, но позволяет быстро выполнять чтение данных. Используется журнальная структура обновлений, при которой необходимые обновления буферизуются и распространяются на необходимые узлы по мере возможности. Операции чтения используют уже имеющиеся данные, даже если эти данные ожидают обновления. Непротиворечивость обновляемых данных гарантируется только конечная. Другими словами, системы баз данных NoSQL, в большинстве своём, пользуются моделью транзакций, отличной от традиционной модели ACID [31], поддерживаемой большинством реляционных СУБД, требующей затрат по времени доступа к данным и основанной на блокировках при обеспечении совместной работы. Описанные возможности соответствуют семантике транзакционной модели BASE [32], декларирующей постоянную доступность данных (отсутствие блокировок даже для обновляемых данных), гибкое состояние (не гарантирующее актуальность считываемых данных) и конечную целостность (в конечном счёте, данные в базе гарантированно будут обновлены).

Ключи и значения могут иметь различную структуру и организовываться в различные структуры в разных системах. С точки зрения моделей данных системы баз данных NoSQL можно разделить на следующие основные классы:

- базы данных *«ключ-значение»* (например, Oracle NoSQL [33]), основанные напрямую на парах ключей и значений, в которых, однако, и ключи, и значения, возможно, будут иметь определённую структуру или быть слабоструктурированными;
- базы данных *с колоночным хранением* (например, Cassandra [34]), в которых с одним ключом связан набор колонок значений, имеющих имена, и таким образом, имитируется табличная организация, однако же набор колонок обычно не фиксируется какой-либо схемой, и может динамически изменяться;

- *документные базы данных* (например, MongoDB [29]), в которых разрешены иерархические структуры, основанные на вложенных парах «ключ-значение», то есть, значением любой пары может становиться последовательность пар ключ значение.

Как в традиционных системах баз данных, так и в системах баз данных NoSQL при решении задачи отображения моделей строится отображение языков определения и манипулирования данными. В случае моделей NoSQL структура данных не определяется схемой данных в явном виде, и отображение на уровне языка определения данных сталкивается с отсутствием её описания. Интерфейс приложения, работающего над базой, обеспечивает более высокий уровень семантики данных, и семантическое отображение ресурсов в целевую модель может быть организовано на уровне использования интерфейса приложения. Однако приложения создаются для решения определённых задач, одна база данных может использоваться в ряде приложений. Не всегда на уровне приложений предоставляются достаточные средства доступа к данным для отображения схем и трансляции данных. Тогда отображение ресурсов и доступ к данным необходимо строить напрямую через интерфейс языков баз данных NoSQL. Этот подход может быть связан с решением проблем слабой структурированности данных, недостаточной информации о структуре и семантике данных из-за отсутствия определения схем данных. В таких случаях представляется оправданным выявление структуры данных из доступных источников: кода приложения, документации, знаний создателя базы и приложения, анализа самих данных.

В нижеследующем подразделе рассматривается отображение одного из классов NoSQL моделей в каноническую модель – модели данных «ключ-значение». Более подробно отображения всех трех классов моделей рассмотрены в работе [35].

2.3.1. Отображение модели баз данных «ключ-значение» в каноническую модель

В данном разделе рассматривается конкретная модель баз данных «ключ-значение», а именно – модель системы баз данных Oracle NoSQL [33].

Хранимыми элементами в данной системе являются ключи и значения. Ключи являются составными и разделяются на старший ключ (major key) и младший ключ (minor key). *Старший ключ* представляет собой последовательность из одного или более компонентов, являющихся строками. По старшему ключу, помимо прочего, производится выбор узла в кластере, так что пары, у которых старшие ключи совпадают, гарантированно будут находиться на одном и том же узле. *Младший ключ* также является последовательностью строк, однако он может быть и пустым. Вместе со старшим ключом младший ключ определяет уникальное значение ключа, с которым в базе может быть связано одно значение. Значения в парах Oracle NoSQL могут быть произвольными. Они рассматриваются как строки байтов, и в них могут быть записаны данные с любой семантикой, структурированные или неструктурированные.

Рассмотрим следующий пример структуры данных «ключ-значение» на языке JSON [36] :

```
{"Smith/Bob/-/phonenumber/home": "555 5555"}
{"Smith/Bob/-/phonenumber/mobile": "333 3333"}
```

Старший ключ (например, “Smith/Bob”) определяет имя и фамилию человека, к которому относятся данные, а младший ключ (например, “phonenumber/home”) – путь в структуре, описывающей вид данных об этом человеке. В значении, связанном с ключом, записан телефон человека.

Для отображения структур данных, хранящихся в базе данных модели «ключ-значение», важно понять, какие из компонентов ключей являются для объектной модели экземплярами (“Smith”), а какие определяют атрибуты структур (“phonenumber”). Если компонент ключа имеет некоторый набор *фиксированных* значений, есть вероятность, что они играют роль определения атрибутов. *Нефиксированные* значения вероятнее всего соответствуют экземпля-

рам. Однако указать точно, какие из компонентов ключа определяют структуру, а какие являются экземплярами, может эксперт на основании документации, анализа кода приложения или собственных знаний о структуре базы. Для фиксированных значений компонентов также необходимо определить, какие из них являются дочерними для других, то есть связанными с определённым значением родительского компонента.

После того, как выяснена природа фиксированных значений компонентов ключей, можно переходить к отображению структуры базы данных. Можно выделить следующие правила отображения:

1. Связанные по структуре ключей пары отображаются в классы (*class1, class2, …*) и типы их экземпляров. Если какой-либо компонент ключа фиксирован и имеет единственное значение в подобных ключах, то он становится именем класса.

2. В соответствие компонентам ключей, имеющим нефиксированные значения, ставятся атрибуты типа экземпляров класса (*majorKey1, majorKey2, ..., minorKey1, minorKey2, ...*). Значения компонентов станут при этом строковыми значениями атрибутов. Однако при известной семантике компонентов ключей можно задавать другие типы атрибутов и преобразование строковых значений компонентов ключей к ним. Атрибуты могут иметь пустое значение, если компонент ключа не определён в некоторых парах.

3. Если в качестве компонента ключа используется фиксированное значение, его следует сделать атрибутом с именем, соответствующим значению компонента.

4. Если для компонента с фиксированным значением существуют дочерние фиксированные компоненты, то есть, фиксированные значения связаны с единственным определённым значением данного родительского компонента, то создаётся одноимённый со значением родительского компонента абстрактный тип данных с атрибутами, соответствующими фиксированным значениям дочернего компонента. Атрибут, порождённый родительским компонентом, приобретает в качестве типа значений созданный абстрактный тип данных.

5. Значение пары отображается в значение атрибута, соответствующего последнему фиксированному значению, у которого нет дочерних компонентов. Либо, если такового не нет, то значение пары отображается в атрибут *value*. Типом значения атрибута по умолчанию является фрейм языка СИНТЕЗ. Необходимо знать способ преобразования значений во фреймы. Если выявить такой способ невозможно, значения будут иметь тип *bitstring* языка СИНТЕЗ. Значения могут приводиться к любым другим типам в зависимости от их известной семантики и структуры.

6. Набор всех атрибутов типа, образованных компонентами полных ключей, указывается как уникальный. Если с атрибутом, соответствующим фиксированному значению компонента ключа, связано значение, соответствующее значению в паре, то включать его в набор излишне.

Рассмотрим схему данных, соответствующую структуре данных из вышеприведенного примера, построенную по приведенным правилам:

```
{ class1; in: class;
   instance_section: {
      majorKey1: String;
      majorKey2: String;
      phonenumber: Phonenumber;
      key: { unique; { majorKey1, majorKey2 } };
   };
}
{ Phonenumber; in: type;
    home: String;
    mobile: String;
}
```

Принципы отображения ЯМД подробно рассмотрены в работе [35]. В данном разделе иллюстрируется представление запросов с условиями канонической модели в ЯМД Oracle NoSQL.

Рассмотрим запрос с условиями на компоненты старшего ключа:

```
q(x) :- class1(x) & x.majorKey1="Smith" & x.majorKey2="Bob".
```

В ЯМД Oracle NoSQL такой запрос представляется при помощи операции множественного чтения *multiGet(k, r)*. Операция читает пары «ключ-значение», соответствующие неполному ключу *k*, задающему только часть первых компонентов полных ключей. В *r* налагаются условия на значения компонента ключа, следующего за компонентами, заданными в *k*:

```
SortedMap<Key, ValueVersion> x = kvstore.multiGet(
Key.createKey((new ArrayList<String>()).add("Smith").add("Bob"));
```

2.4. Триплетная модель RDF

Модель RDF [37] используется для описания информационных или неинформационных сущностей произвольного вида в открытой информационной среде, а также в качестве базовой модели в проектах Семантического Веба. В основе модели лежат триплеты «субъект-предикат-объект», посредством которых описываются сущности, называемые в этой модели ресурсами.

RDF является разновидностью графовой модели данных, в которой субъекты и объекты являются узлами, а предикаты – направленными именованными рёбрами. Однако относительно графов модель RDF привносит свою специфику. Во-первых, посредством использования URI она определяет глобальную идентификацию ресурсов в открытом информационном пространстве. Во-вторых, модель является расширяемой. За рёбрами RDF-графов в зависимости от используемых словарей может скрываться определённая семантика, фактически определяющая новые информационные модели. В частности, RDF-модель используется в качестве базиса расширяющих её моделей, в которых предопределены предикаты, несущие конкретную семантику элементов моделей, и существует определённый список моделей, неразрывно связанный с RDF. В-третьих, в модели RDF предусмотрены правила вывода неявных связей. В отличие от графовых моделей, имеющих реализации алгоритмов для решения определённых задач, в RDF набор правил закреплён в определении самой модели, и фактически является частью определения семантики элементов модели. В зависимости от словарей, определяющих семантику описаний, набор правил

вывода также может расширяться. Все эти особенности отражаются на подходах к отображению данной модели в каноническую модель предметных посредников. Таким образом, модель данных RDF заслуживает отдельного рассмотрения.

С моделью RDF неразрывно связаны модель RDF-Schema, предоставляющая средства определения схем данных для RDF-описаний, а также язык запросов SPARQL [39], основанный на использовании триплетов в запросах. Фактически они вместе определяют обогащённую модель данных, включающую языки определения данных и манипулирования данными. Помимо этого, модель RDF является базисом для других моделей данных. В частности, язык OWL [38] с определёнными семантикой и набором правил вывода в одном из своих синтаксисов основывается на RDF, и данные, представленные в модели OWL, обретают вид RDF-ресурсов.

Модель RDF определяется посредством абстрактного синтаксиса и формальной семантики. Предложено несколько конкретных синтаксисов для сериализации RDF-данных. Данные RDF могут быть распределены в Вебе и представляться в открытой среде в виде документов, сгенерированных в соответствии с одним из синтаксисов. Для поиска данных служат специализированные поисковые механизмы, обращение к которым возможно с помощью языка SPARQL [39]. С другой стороны, для хранения RDF-описаний и манипулирования ими разработан ряд специализированных хранилищ. Большинство из них использовали реляционные структуры для представления графов и триплетов в них. Сегодня RDF-хранилища используют колоночную организацию хранения триплетов и горизонтальное масштабирование [40]. Помимо этого, множество проектов в Вебе предоставляют API-интерфейс к своим данным в виде RDF в соответствии с определённой схемой и точек доступа SPARQL.

Таким образом, информационными ресурсами с данными RDF, интегрируемыми в предметные посредники, могут становиться поисковые механизмы с точками доступа SPARQL общего назначения, RDF-хранилища или Веб-проекты с API, представляющим данные в виде RDF.

В Вебе успешно развивается проект Linked Open Data (LOD) [41], ставящий задачей накопление открытых взаимосвязанных данных в Вебе в виде триплетов. Он использует в своей основе стек приведённый выше языков, для работы с данными в модели RDF. Открытость данных обеспечивает доступ к ним в машинно-читаемом и понятном человеку формате. Взаимосвязанность данных является залогом доступности данных в проекте. Данные, отвечающие набору определённых требований проекта [42], доступ к которым осуществляется через RDF, по заявке авторов данных могут быть включены в состав проекта. Требования касаются открытого лицензирования, уникальной идентификации данных в вебе, связанности данных с другими данными, состоящими в проекте, описание метаданных к ним, анонсирование появления новых данных и других аспектов.

В проекте участвуют коллекции связанных данных на веб-серверах, в базах данных, в RDF-хранилищах. Над данными могут разрабатываться бизнес- или научные приложения. Созданы поисковые системы, предназначенные для поиска связанных данных в вебе и выявления необходимой информации по связям между данными. Для создания сервисов, визуализации, связывания новой информаци разрабатываются "мэшапы" – комбинированные множества данных.

Обычно в проектах накапливается большое количество триплетов, относящихся к всевозможным областям знаний. Внешними проектами LOD используется для идентификации сущностей в вебе, обогащения имеющейся информации о сущностях, обеспечения данных необходимыми метаданными и решения других задач. Поэтому накопленные в проекте данные являются одним из подтверждений актуальности использования информационных ресурсов, представленных в модели RDF, при решении задач с использованием доступа к сверхбольшим объёмам данных.

2.4.1. Отображение триплетной модели данных в каноническую модель

Тривиальным отображением модели данных RDF в каноническую модель могло бы стать отображение отношений, хранящих триплеты, в абстрактный тип с атрибутами, соответствующими субъектам, предикатам и объектам. Однако такой подход не приемлем в среде предметных посредников. Для них важна семантическая интеграция информационных ресурсов, при которой все обнаруживаемые связи данных с семантикой предметной области должны отображаться в понятия предметной области посредника и в соответствующие структуры концептуальной схемы посредника. Поэтому выявленную семантику предметной области следует применить для создания соответствующих типов в спецификациях информационных ресурсов, интегрируемых в посреднике.

Отображать данные в базовой модели RDF в каноническую модель необходимо на уровне фреймов языка СИНТЕЗ. Фреймы, как и графы RDF, призваны описывать ресурсы произвольной природы. И при отображении RDF триплеты с общим субъектом как описывающие один и тот же ресурс должны ассоциироваться с определённым фреймом. Далее в примерах RDF-спецификаций используется синтаксис Turtle [43]:

```
<http://somewhere/MattJones/>
vCard:FN "Matt Jones" ;
vCard:N [ vCard:Family "Jones"; vCard:Given "Matthew" ].
```

Пространства имён (например, *vCard*), определяемые в RDF как словари, отображаются в миры фреймов. Мир фреймов в языке СИНТЕЗ представляется с помощью фрейма с указанием принадлежности его метаклассу world, определяющему семантику этого фрейма как мира. Внутри мира располагаются спецификации входящих в него фреймов:

```
{ vCard; in: world;  … }
```

С определённым субъектом триплетов связывается фрейм с соответствующим именем. Набор триплетов с общим субъектом отображается в набор слотов одного фрейма. Предикатам, связанным с этим субъектом, соответствуют слоты фрейма. Объектам – значения слотов. Если тип значения не опреде-

лён, он назначается как тип *frame*, так как фреймы формируют произвольные значения. Набор триплетов, различающихся объектами, отображается в слот фрейма с несколькими значениями:

```
{ MattJones; in: frame;
  FN: 'Matt Jones';
  N: {in: frame; Family: 'Jones';},
     { in: frame; Given: 'Matthew';};
}
```

Описания в модели RDF Schema отображаются в объектную модель языка СИНТЕЗ. Сложности отображения RDF в объектную модель (как и реляционную) связаны, в основном, с конфликтом парадигм открытого мира в случае RDF и закрытого мира в случае объектной модели [44]. Во-первых, описываемые ресурсы могут принадлежать нескольким RDF-классам одновременно. Большинство объектных моделей исключает такую возможность, определяя принадлежность объекта строго одному типу (классу). Во-вторых, свойства в RDF являются самостоятельными сущностями, которые могут использоваться в разных классах и ресурсах. Их связь с классами определяется только указанием классов домена и области значений свойств. Тем временем, атрибуты типов в объектных моделях обычно жёстко связаны с единственным типом (классом), а также наследуются от супертипов. В-третьих, структура экземпляров в RDF не ограничивается спецификациями классов. В большинстве объектных моделей объект жёстко соответствует структуре, описанной в типе, которому он принадлежит. Средства языка СИНТЕЗ позволяют избежать указанных проблем, благодаря разделению в языке на классы как множества и абстрактные типы данных как интенсиональные описания, наличию языка фреймов, не ограничивающего данные определёнными типами, а также возможности использовать типовые выражения для их комбинации.

Определения RDF Schema определяют схему RDF-данных посредством задания ресурсов, классов, свойств и их ограничений:

```
vCard:VCard rdf:type rdf:Class.
```

```
vCard:Name rdf:type rdf:Class.
vCard:fn rdf:type rdf:Property; rdfs:domain vCard:VCard;
  rdfs:range xsd:string.
vCard:n rdf:type rdf:Property; rdfs:domain vCard:VCard;
  rdfs:range vCard:Name.
vCard:family-name rdf:type rdf:Property; rdfs:domain vCard:Name;
  rdfs:range xsd:string.
vCard:Given rdf:type rdf:Property; rdfs:domain vCard:Name;
  rdfs:range xsd:string.
```

Ресурс RDF Schema отображается во фрейм языка СИНТЕЗ. Класс RDF Schema отображается в класс языка СИНТЕЗ с типом экземпляров:

```
{ VCard; in: class;
   instance_section: {
      fn: {set; type_of_element: string;};
      n: { set; type_of_element: Name;};
   };
}
{ Name; in: class;
  instance_section: {
    family_name: string;
    given_name: string;
  };
}
```

Собственного ЯМД в модели RDF нет. Для запросов над базами RDF-документов обычно используется язык SPARQL [39]. Запросы на языке SPARQL представляют собой конъюнкции триплетов, на месте субъектов, предикатов и объектов в которых могут быть определённые значения или переменные.

Например, запрос канонической модели, возвращающий имена и фамилии лиц, хранящихся в базе данных:

```
q([fname, gname]) :-
  VCard(x), is_in(vc, x.n),
  fname = vc.family_name, gname = vc.given_name.
```

отображается в следующий запрос на SPARQL:

```
SELECT ?x ?fname ?gname {
   ?x  vcard:n  ?vc.
   ?vc vcard:family-name ?fname.
   ?vc vcard:given-name ?gname.
}
```

Более подробно отображение модели RDF в каноническую модель рассмотрено в работе [45].

3. Сохранение информации и семантики операций ЯМД при унификации нетрадиционных моделей данных

Метод доказательства сохранения информации и семантики операций при отображении моделей данных [46] основывается на представлении семантики моделей в формальном языке спецификаций AMN [25].

Язык AMN представляет собой теоретико-модельную нотацию, основанную на теории множеств и типизированном языке первого порядка. Спецификации AMN называются абстрактными машинами. AMN позволяет интегрированно рассматривать спецификацию пространства состояний и поведения машины (определенного операциями на состояниях). В AMN формализуется специальное отношение между спецификациями – *уточнение*. Неформально, спецификация B уточняет спецификацию A, если пользователь может использовать B вместо A, не замечая факта замены A на B.

Идея метода заключается в следующем. Рассмотрим исходную модель данных S и целевую модель T. Построим отображение θ модели S в модель T (подобно изложенным отображениям в предыдущем разделе). Выразим семантику моделей в виде абстрактных машин AMN, построив при этом машины M_S и M_T соответственно. При этом структуры данных моделей – классы, массивы представляются переменными машин, различные свойства структур данных представляются инвариантами машин, характерные операции моделей данных представляются операциями машин. 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.

Будем считать отображение θ сохраняющим информацию и семантику операций, если машина M_S, соответствующая исходной модели, уточняет машину M_T, соответствующую целевой модели. Уточнение доказывается интерактивно при помощи специальных программных средств [47].

В качестве иллюстрации основных принципов выражения семантики моделей данных в AMN рассмотрим частичные AMN-спецификации, соответствующие моделям ADM (раздел 2.1) и СИНТЕЗ.

Спецификация, выражающая семантику объектной модели языка СИНТЕЗ, представляется в языке AMN конструкцией REFINEMENT:

```
REFINEMENT ObjectDM
```

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

```
ABSTRACT_VARIABLES
    typeNames, classNames, instanceType, typeAttributes,
    attributeNames, attributeType,
    objectIDs, objectType, integerAttributeValue, …
INVARIANT …
```

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

Так, имена типов и классов представлены переменными *typeNames* и *classNames*, тип которых – подмножество множества строк (*STRING_Type*):

```
typeNames: POW(STRING_Type) &
classNames: POW(STRING_Type)
```

Здесь *POW* – конструктор множества подмножеств.

Атрибуты (переменная *attributeNames*) представлены частичной функцией (знак +->), ставящей в соответствие уникальному идентификатору атрибута (натуральному числу из множества *NAT*) имя атрибута (строку):

```
attributeNames: NAT +-> STRING_Type
```

Типы экземпляров классов (переменная *instanceType*) представлены тотальной функцией (знак -->) из множества имен классов в множество имен типов:

```
instanceType: classNames --> typeNames
```

Принадлежность атрибутов типам (переменная *typeAttributes*) выражена тотальной функцией из множества имен типов в множество подмножеств атрибутов:

```
typeAttributes: typeNames --> POW(dom(attributeNames))
```

Здесь *dom* – операция, возвращающая область определения функции, а *dom(attributeNames)* – множество имен атрибутов.

Типы значений атрибутов (переменная *attributeType*) представлены функцией из множества атрибутов в множество идентификаторов встроенных типов данных *BuiltInTypes*:

```
attributeType: dom(attributeNames) +-> BuiltInTypes
```

Значения атрибутов объектов (например, переменная *integerAttributeValue*) представлены функциями из множества атрибутов в функции из множества идентификаторов объектов в множества значений атрибутов. Для простоты рассмотрена лишь функция для целочисленных атрибутов:

```
integerAttributeValue:
    dom(attributeNames) +-> (objectIDs +-> INT) &
```

Из всего ЯМД рассмотрим единственную операцию *update* обновления значений атрибута в объектах класса:

```
OPERATIONS
update(cls, attr, exp, cond) =
PRE cls: classNames &
    attr: typeAttributes(instanceType(cls)) &
```

```
    attributeType(attr) = Integer &
    exp: INT --> INT & cond: NAT --> BOOL
THEN
  integerAttributeValue :=
  integerAttributeValue <+
  { xx | xx: (NAT*(NAT<->INT)) &
     #(oo, val).( oo: objectsOfClass(cls) & val: INT &
         xx = attr |-> ({oo |-> val}) &
     (cond(integerAttributeValue(attr)(oo)) = TRUE =>
         val = exp(integerAttributeValue(attr)(oo))) &
     (cond(integerAttributeValue(attr)(oo)) = FALSE =>
          val = integerAttributeValue(attr)(oo))  ) }
END
```

Параметрами операции являются имя класса *cls*, имя целочисленного атрибута *attr* типа экземпляров класса, функция *exp*, отвечающая за преобразование атрибута и функция *cond*, отвечающая условию на значение атрибута. Пусть *o* – некоторый объект класса *cls*, для которого определено значение атрибута *attr* и это значение есть *v*. Тогда операция *update* изменяет значение атрибута на *exp(v)* в случае, если выражение *cond(v)* принимает значение истина и оставляет значение атрибута без изменений в противном случае. Очевидно, такая операция *update* есть обобщение примера обновления, рассмотренного в разделе 2.1.2. Действительно, для рассмотренного примера *cls* есть *source*, *attr* есть *val*, *exp(v) = -v*, *cond(v) = abs(v)*.

Заметим, что в рассмотренной спецификации для простоты не рассмотрены некоторые черты объектной модели, например отношения тип-подтип и класс-подкласс.

Спецификация, выражающая семантику модели ADM, представляется в языке AMN конструкцией

```
REFINEMENT ArrayDM
```

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

```
ABSTRACT_VARIABLES
```

```
    arrayNames, arrayCellAttributes, cellAtrributeType,
    integerCellAttributeValue,…
```

Имена массивов представлены переменной *arrayNames*; принадлежность атрибутов ячеек – переменной *arrayCellAttributes*; тип атрибута ячейки – переменной *cellAtrributeType*; значения атрибутов ячеек – переменной *integerCellAttributeValue*. Переменные типизируются в разделе INVARIANT при помощи частичных и тотальных функций аналогично переменным, использующимся для придания семантики объектной модели:

```
INVARIANT
    arrayNames: POW(STRING_Type) &
    arrayCellAttributes: arrayNames -->
        POW(dom(cellAttributeNames)) &
    cellAtrributeType:
       dom(cellAttributeNames) --> BuiltInTypes &
    integerCellAttributeValue:
        NAT*dom(cellAttributeNames) +-> INT
```

Здесь * - знак декартова произведения множеств.

Аналогично объектной модели рассмотрена единственная операция ЯМД – операция обновления *update*:

```
OPERATIONS
update(cls, attr, exp, cond) =
PRE cls: arrayNames & attr: arrayCellAttributes(cls) &
  cellAtrributeType(attr) = Integer &
  exp: INT --> INT & cond: NAT --> BOOL
THEN
  integerCellAttributeValue :=
  integerCellAttributeValue <+
  { yy | yy: (NAT*NAT)*INT &
    #(cell, val).(cell: cells(cls) & val: INT &
    yy = ((cell |-> attr)|-> val) &
    (cond(integerCellAttributeValue(cell, attr)) = TRUE =>
     val = exp(integerCellAttributeValue(cell, attr))) &
    (cond(integerCellAttributeValue(cell, attr))= FALSE =>
```

```
      val = integerCellAttributeValue(cell, attr)) ) }
END
END
```

Сигнатура операции совпадает с сигнатурой операции объектной модели. Семантика операции также аналогична: значение *v* атрибута *attr* массива *cls* заменяется на *exp(v)*, если значение *cond(v)* есть истина и не изменяется в противном случае.

Заметим, что в данной спецификации для простоты не рассмотрены некоторые черты ADM, например, нецелочисленные измерения.

Для формального доказательства того, что машина *ArrayDM* 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 2.1, и объединяет отдельные формулы, соответствующие этим принципам, в одну конъюнкцию.

Например, для любой ячейки значения ее атрибутов совпадают со значениями соответствующих атрибутов объекта, соответствующего ячейке:

```
!(cell, cattr).(cell: NAT & cattr: NAT &
    (cell |-> cattr): dom(integerCellAttributeValue) =>
    cell: dom(integerAttributeValue(cattr)) &
    integerCellAttributeValue(cell, cattr) =
      integerAttributeValue(cattr)(cell) )
```

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

```
REFINES ObjectDM
```

Спецификации *ObjectDM* и *ArrayDM* вместе с инвариантом уточнения были загружены в инструментальное средство Atelier B [47]. Автоматически были сгенерированы теоремы, выражающие уточнение спецификаций. В частности, для операции *update* были сгенерированы 10 теорем. 3 из них были дока-

заны автоматически, для доказательства остальных необходимо применять интерактивные средства доказательства.

4. Родственные исследования

Модели, основанные на многомерных массивах. Существует ряд работ, связанных с отображением моделей, основанных на многомерных массивах, в реляционную модель данных. Обычно эти работы нацелены на реализацию многомерных массивов при помощи реляционных СУБД. Такие работы появились одновременно с исследованиями моделей, основанных на многомерных массивах [10], и развиваются в настоящее время [48].

Основные особенности результатов, изложенных в данной статье, состоят в следующем. В качестве исходной модели при отображении используется специфическая модель, основанная на многомерных массивах, СУБД SciDB, язык которой представляет собой комбинацию декларативного SQL-подобного языка и функционального языка, включающего специфические операции над многомерными массивами. Результаты отображения этой конкретной модели в каноническую могут быть использованы для интеграции ресурсов, представленных в моделях, основанных на многомерных массивах, но отличных от модели данных SciDB.

Графовые модели. Известно сравнительно небольшое количество работ, в которых исследуются вопросы интеграции или унификации графовых моделей данных. Например, в работе [49] язык запросов над гиперграфами используется для описания взглядов при интеграции графовых баз данных в посредниках. В работе [50] гиперграфовая модель также используется для интеграции графовых баз данных. Предлагается набор операций в рамках гиперграфовой модели для преобразования схемы ресурса в федеративную схему. В подобных работах вопрос модельной неоднородности не возникает, так как и в качестве канонической модели, и в качестве модели ресурсов выступает гиперграфовая

модель. В работе [51] 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 [52]. 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.

Основной особенностью результатов, изложенных в работе [53], является то, что в качестве исходной модели при отображении используется синтетическая модель, структуры данных которой покрывают возможности современных графовых СУБД, основанных на простых и атрибутированных графах.

Модели NoSQL. Отображение моделей с участием моделей NoSQL востребовано скорее в обратном направлении: из произвольных моделей в NoSQL. При отображении реляционной модели в NoSQL разрабатываются способы представления в NoSQL данных, ранее представленных в реляционной модели, и имитация операций реляционной алгебры в моделях NoSQL [54]. Отображение объектной модели в NoSQL имеет целью исследовать представление объектов при использовании языков программирования с базами данных NoSQL [55]. При этом подчёркивается отсутствие дорогостоящего объектно-реляционного отображения, с каким приходится сталкиваться при работе в языках программирования с реляционными системами баз данных.

Известные промышленные решения по интеграции NoSQL моделей – хранилище Hive [56] и СУБД Oracle – устроены довольно просто. Целью интеграции является импорт данных, представленных в нетрадиционных моделях, в реляционное хранилище. Для Hive такой моделью является формат обмена

данными JSON [57], для Oracle – СУБД Oracle NoSQL. Предполагается, что исходные данные, хотя и представлены в слабоструктурированной модели, но фактически имеют однородную структуру. Для обеспечения импорта такая структура должна быть явным образом определена средствами ЯОД реляционной модели [58][59].

Представление произвольных вложенных структур данных JSON в языке системы Hive возможно благодаря тому, что реляционная модель в Hive расширена *сложными типами данных* – массивами, отображениями (map), структурами (struct), объединениями (union). Аналогично, реляционная модель в СУБД Oracle расширена объектными типами, массивами (varrays) и вложенными таблицами.

Триплетная модель RDF. Отображение и трансляция данных между моделью RDF и другими моделями данных не редкость. Большинство данных хранится и обрабатывается в моделях, отличных от RDF, а трансляция их в RDF необходима для обмена информацией между проектами, между различными представлениями данных, для публикации данных в семантическом вебе в таких проектах как Linked Open Data [60], и других. В основном, разрабатываемые отображения имеют следующие направления.

Большинство инструментов для работы с RDF используют исключительно триплеты в интерфейсе для всех предикатов, включая определённые в самих моделях RDF и RDF Schema. При этом сами инструменты могут быть предназначены для использования в других моделях данных, например, с объектно-ориентированными языками программирования. Отображение RDF в другие модели производится без ухода от триплетной формы [61], а основные преобразования из моделей, используемых в инструменте, в RDF приходится совершать на уровне прикладных программ.

В реляционных хранилищах триплетов интересно рассмотреть подходы к их хранению. Первый из них заключается в хранении триплетов в единственном отношении с тремя основными атрибутами, соответствующими субъекту,

предикату и объекту. Помимо этого в нём же могут храниться идентификаторы графов и другие необходимые элементы. Чтобы не хранить все данные в одном большом отношении, его разбивают по некоторому принципу, например, по принадлежности субъектов определённым классам [62]. Но для этого над RDF-данными должна быть определена некоторая схема. Предлагаемые представления продолжают разрабатываться [63], мотивируемые поиском эффективного хранения и доступа к RDF-графам.

Отображения, учитывающие семантику данных, обычно разрабатывается для обратной задачи: трансляции данных из реляционной или объектной модели в RDF [64]. В этом случае обратная трансляция должна обеспечивать восстановление исходных спецификаций.

Работы, связанные с отображением языка RDF с учётом семантики данных в высокоуровневые модели данных, разрабатываются для обработки RDF-данных из языков программирования [44] или языков декларативных запросов [65].

Заключение

Нетрадиционные (не-реляционные) модели данных ориентированы на представление и манипулирование коллекциями больших данных, накапливаемых в Вебе, в социальных сетях, в виде машинных и сенсорных данных и т.д. При создании систем интеграции больших данных одной из важнейших проблем является унификация нетрадиционных моделей данных - отображение исходных моделей данных в каноническую информационную модель, сохраняющее семантику их языков определения данных и манипулирования данными. Интеграция данных занимает важное место в стеках программных решений всех ведущих вендоров. Стеки обычно включают решения по сбору, распределенному хранению и обработке больших данных; средства организации хранилищ данных; средства виртуальной интеграции данных; средства аналитики на различных уровнях стека. Методы унификации нетрадиционных моделей дан-

ных необходимы в тех случаях и на тех уровнях стека, где происходит интеграция разномодельных данных. Такая интеграция может осуществляться как на уровне хранилищ данных и систем виртуальной интеграции, так и на уровне распределенных инфраструктур хранения и обработки. В существующих промышленных решениях осуществляется интеграция данных, представленных преимущественно в различных вариантах реляционной модели данных. Унификация нетрадиционных моделей данных представляется задачей систем интеграции данных следующих поколений.

В работе рассмотрены принципы и методы отображения четырех видов нетрадиционных моделей данных в каноническую модель: моделей данных, основанные на многомерных массивах; графовых моделей данных; моделей данных NoSQL; триплетной модели данных RDF. В качестве канонической модели используется объектная-фреймовая модель с Datalog-подобным языком запросов (программ) – язык СИНТЕЗ. Для отображения обеспечивается формальное доказательство сохранения информации и семантики операций ЯМД. Результаты могут быть с легкостью обобщены и использованы при интеграции в системах, использующих каноническую модель, отличную от языка СИНТЕЗ – например, другую объектную (ODMG) или объектно-реляционную модель (SQL:2003). Произведен обзор родственных работ по интеграции и отображению нетрадиционных моделей данных.

Разработанные методы будут применяться в инфраструктуре предметных посредников, разрабатываемой в ИПИ РАН.

Список литературы

1. Challenges and Opportunities with Big Data. – Computing Community Consortium, 2012. – URL: http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf (дата обращения: 05.02.2014).
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., 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.
4. Kalinichenko L.A., Stupnikov S.A. Heterogeneous information model unification as a prerequisite 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.
5. 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.
6. Брюхов Д.О., Вовченко А. Е., Захаров В.Н., Желенкова О.П., Калиниченко Л.А., Мартынов Д.О., Скворцов Н.А., Ступников С.А. Архитектура промежуточного слоя предметных посредников для решения задач над множеством интегрируемых неоднородных распределенных информационных ресурсов в гибридной грид-инфраструктуре виртуальных обсерваторий // Информатика и ее применения. – М.: ИПИ РАН, 2008. – Т. 2, Вып. 1. – С. 2-34.
7. Vassiliadis P., Sellis T.K. A Survey of Logical Models for OLAP Databases. // SIGMOD Record. – 1999. – Vol 28, No. 4. – P. 64-69.
8. Pedersen T.B., Jensen C.S. Multidimensional Database Technology. // IEEE Computer. – 2001. – Vol. 34, No. 12. – P. 40-46.
9. Libkin L., Machlin R., Wong L. A Query Language for Multidimensional Arrays: Design, Implementation, and Optimization Techniques. // Proc. of the 1996 ACM SIGMOD International Conference on Management of Data. – 1996. – P. 228-239.
10. P. Baumann. A Database Array Algebra for Spatio-Temporal Data and Beyond. // Next Generation Information Technologies and Systems. – 1999. – P. 76–93.
11. Brown P.G. Overview of SciDB: Large Scale Array Storage, Processing and Analysis. // Proc. of the 2010 ACM SIGMOD International Conference on Management of Data. – 2010. – P. 963-968.
12. Besla J., Kim K.T. Report from the First Workshop on Extremely Large Databases. // Data Science Journal. – 2008. – Vol. 7.
13. Large Synoptic Survey Telescope. – URL: http://www.lsst.org/ (дата обращения: 05.02.2014).

14. Kersten M.L., Zhang. Y., Ivanova M., Nes N.J. SciQL, a query language for science applications. // EDBT/ICDT Workshop on Array Databases. – Uppsala, 2011. – P. 1-12.
15. Astronomy in ArrayDB. – URL: http://scidb.org/UseCases/ Astronomy%20in%20ArrayDB.pdf (дата обращения: 05.02.2014).
16. SciDB User's Guide. Version 13.12. – 2014. – URL: http://www.scidb.org/HTMLmanual/13.12/scidb_ug/index.html (дата обращения: 05.02.2014).
17. Buck J.B., Watkins N., LeFevre J., Ioannidou K., Maltzahn C., Polyzotis N., Brandt S.A. SciHadoop: array-based query processing in Hadoop. // Proc. of 2011 International Conference for High Performance Computing, Networking, Storage and AnalysisSuper Computing. – ACM, 2011. – ISBN: 978-1-4503-0771-0.
18. Ступников С.А. Верифицируемое отображение модели данных, основанной на многомерных массивах, в объектную модель данных // Информатика и ее применения. – М.: ИПИ РАН, 2013. – Т. 7, вып. 3. – С. 22-34. – ISSN: 1992-2264.
19. Angles R. A Comparison of Current Graph Database Models. // Proc. of IEEE 28th International Conference on Data Engineering Workshops (ICDEW). – 2012. – P. 171-177.
20. 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.
21. 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.
22. Titan Project. – URL: https://github.com/thinkaurelius/titan/wiki (дата обращения: 05.02.2014).
23. Neo4j Graph Database. – URL: http://www.neo4j.org/ (дата обращения: 05.02.2014).
24. The Neo4j Manual. 2013. – URL: http://docs.neo4j.org/ (дата обращения: 05.02.2014).
25. Abrial J.-R. The B-Book: Assigning Programs to Meanings. – Cambridge: Cambridge University Press, 1996. – ISBN:0-521-49619-5.
26. Cattell R. Scalable SQL and NoSQL Data Stores. // ACM SIGMOD Record. – NY: ACM New York, 2010. – Vol. 39, Iss. 4. – P. 12-27.
27. Stoica I., Morris R., Liben-Nowell D., Karger D.R., Kaashoek M.F., Dabek F., Balakrishnan H. Chord: a scalable peer-to-peer lookup protocol for internet applications. – IEEE/ACM Transactions on Networking (TON). – 2003. – Vol. 11, Iss. 1. – P. 17-32.
28. Apache HBase Home. – URL: http://hbase.apache.org/ (дата обращения: 05.02.2014).
29. MongoDB. – URL: http://www.mongodb.org/ (дата обращения: 05.02.2014).

30. Martin J. Managing the Data-base Environment. – New Jersey: Prentice-Hall. – 381 p. – ISBN 0-13-550582-8.
31. Gray J. The Transaction Concept: Virtues and Limitations. // Proceedings of the 7th International Conference on Very Large Databases. – Cupertino: Tandem Computers, 1981. – P. 144–154.
32. Pritchett D. Base: An Acid Alternative. // ACM Queue. – 2008. – Vol 6, Iss 3. – P. 48-55.
33. Getting Started with NoSQL Database 11g Release 2. – Oracle. – 2011. – URL: http://docs.oracle.com/cd/NOSQL/html/GettingStartedGuide/Oracle-NoSQLDB-GSG.pdf (дата обращения: 05.02.2014).
34. The Apache Cassandra Project. – URL: http://cassandra.apache.org/ (дата обращения: 05.02.2014).
35. Скворцов Н. А. Отображение моделей данных NoSQL в объектные спецификации. Труды 14-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» RCDL 2012. – Переславль-Залесский: Университет города Переславля, 2012. – С. 78-87.
36. D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627. – The Internet Engineering Task Force (IETF) – 2006. – URL: http://tools.ietf.org/html/rfc4627 (дата обращения: 05.02.2014).
37. RDF vocabulary description language 1.0: RDF schema. D. Brickley, R.V. Guha (Eds.), W3C Recommendation. – W3C, 2004. – URL: http://www.w3.org/TR/2004/REC-rdf-schema-20040210 (дата обращения: 05.02.2014).
38. OWL Web Ontology Language Reference. M. Dean, G. Schreiber (Eds.), W3C Recommendation. – W3C, 2004. – URL: http://www.w3.org/TR/2004/REC-owl-ref-20040210/ (дата обращения: 05.02.2014).
39. SPARQL Query Language for RDF. E. Prud'hommeaux, A. Seaborne (eds.), W3C Recommendation. – W3C, 2008. – URL: http://www.w3.org/TR/rdf-sparql-query/ (дата обращения: 05.02.2014).
40. Erling O. Virtuoso, a Hybrid RDBMS/Graph Column Store. // IEEE Data Engineering Bulletin. – 2012. – Vol. 35, No. 1. – P. 3-8.
41. Linked Open Data. W3C SWEO Community Project. – URL: http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData (дата обращения: 05.02.2014).
42. Datahub. The easy way to gat, use and share data. – URL: http://datahub.io/ (дата обращения: 05.02.2014).

43. Turtle – Terse RDF Triple Language. – W3C, 2011. – URL: http://www.w3.org/TeamSubmission/turtle/ (дата обращения: 05.02.2014).
44. Oren E. et al. ActiveRDF: Object-oriented semantic web programming. // Proc. of the 16th international conference on World Wide Web. – ACM, 2007. – С. 817-824.
45. 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.
46. Kalinichenko L.A. Method for Data Models Integration in the Common Paradigm // Proc. of the First East-European Symposium on Advances in Databases and Information Systems ADBIS 97. – St.-Petersburg: Nevsky Dialect, 1997. – Vol. 1: Regular Papers. – P. 275-284.
47. Atelier B, the industrial tool to efficiently deploy the B Method. – URL: http://www.atelierb.eu/index-en.php (дата обращения: 05.02.2014).
48. Ballegooij A.R. RAM: Array Database Management through Relational Mapping. – SIKS Dissertation Series. – SIKS, 2009. – 180 p. http://oai.cwi.nl/oai/asset/14074/14074D.pdf (дата обращения: 05.02.2014).
49. 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.
50. Sundaresan S., Hu G: Schema integration of distributed databases using hyper-graph data model. // Proc. of Information Reuse and Integration Conf, IRI 2005. – 2005. – P. 548-553. – ISBN: 0-7803-9093-8.
51. 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.
52. 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.
53. Ступников С. А. Отображение графовых моделей данных в каноническую модель в системах с интенсивным использованием данных. // Системы высокой доступности. – 2014.
54. Merriman D. SQL to Mongo Mapping Chart. – 2011. – URL: http://docs.mongodb.org/manual/reference/sql-comparison/ (дата обращения: 05.02.2014).
55. Meijer H.J.M. Object model to key-value data model mapping – US Patent App. 12/938,168, 2010. – Google Patents, 2013.

56. The Apache Hive data warehouse software. – URL: http://hive.apache.org/ (дата обращения: 05.02.2014).
57. JavaScript Object Notation. – URL: http://www.json.org/ (дата обращения: 05.02.2014).
58. JsonSerde - a read/write SerDe for JSON Data. – URL: https://github.com/rcongiu/Hive-JSON-Serde (дата обращения: 05.02.2014).
59. Using Oracle External Tables To Access Oracle NoSQL Database Data. Oracle Technology Network. Documentation. – URL: http://docs.oracle.com/cd/E26161_02/html/examples/ externaltables/cookbook.html (дата обращения: 05.02.2014).
60. Yu L. Linked open data // A Developer's Guide to the Semantic Web. – Springer Berlin Heidelberg, 2011. – C. 409-466. – ISBN 978-3-642-15969-5.
61. Beckett D., Grant J. SWAD-Europe Deliverable 10.2: Mapping Semantic Web data with RDBMSes. – W3C, 2003. – URL: http://www.w3.org/2001/sw/Europe/reports/ scalable_rdbms_mapping_report. (дата обращения: 05.02.2014).
62. Wilkinson K., Sayers C., Kuno H.A., Reynolds D. Efficient RDF Storage and Retrieval in Jena2.// In Semantic Web and Databases Workshop. – 2003. – P. 131–150.
63. Bornea M.A., et al. Building an Efficient RDF Store Over a Relational Database. // Proc. of the 2013 ACM SIGMOD International Conference on Management of Data. – New York: ACM, 2013. – P. 121-132.
64. Sahoo S.S. et al. A survey of current approaches for mapping of relational databases to RDF. – W3C RDB2RDF Incubator Group Report. – 2009.
65. Chebotko A., Lu S., Fotouhi F. Semantics preserving SPARQL-to-SQL translation // Data & Knowledge Engineering. – 2009. – Т. 68. – №. 10. – С. 973-1000.

Реферат

В настоящий период развития ИТ во главу угла поставлено создание средств анализа данных, накапливаемых в Вебе, в социальных сетях, в виде машинных и сенсорных данных и т.д.. Данные таких масштабов (часто измеряемых уже в петабайтах) относятся к категории «больших данных». Для представления и манипулирования коллекциями подобных данных созданы новые модели данных, отличающиеся от традиционных (реляционных) моделей данных. Одной из нерешенных проблем манипулирования большими данными является проблема интеграции данных различных моделей, решение которой тре-

бует прежде всего создания унифицированного представления различных видов нетрадиционных моделей в канонической информационной модели (общем языке, унифицирующем языки разнообразных моделей данных). Такое представление требует построения отображения моделей данных, сохраняющего семантику их языков определения данных и манипулирования данными, в каноническую модель. Наличие отображения необходимо как для материализованной интеграции (создание хранилища данных), так и для виртуальной интеграции (посредством предметных посредников). В статье рассматриваются принципы отображения четырех видов нетрадиционных моделей данных в каноническую модель (в качестве которой для определенности используется язык СИНТЕЗ, реализующий комбинированную объектно-фреймовую модель данных): модели данных, основанные на многомерных массивах; графовые модели данных; модели данных NoSQL; триплетная модель данных (RDF). На примерах иллюстрируется метод верификации отображений – доказательства сохранения информации и семантики операций при отображении. Целью работы является определение хорошо обоснованных унифицирующих отображений нетрадиционных моделей данных, представление возможности унифицированного представления столь различных моделей для материализованной или виртуальной интеграции соответствующих коллекций данных.

Methods for Unification of Non-traditional Data Models

Abstract

One of the not yet solved problems of Big Data manipulation is the problem of integration of various non-traditional data models (differing from the relational one). For solving of such problem it is uppermost required to create the unified representation of various kinds of non-traditional data models in the canonical information model. For such representation it is required to build the data model mapping preserving semantics of its data description and data manipulation languages into the ca-

nonical one. In the paper the principles of mapping of four kinds of non-traditional data models into the object-frame canonical model are considered: the data models based on the multidimensional arrays; the graph-based data models; the NoSQL data models; the triple-based RDF data model. The unification of the non-traditional data model serves as a base for the materialized or virtual integration of the respective collections of data.

Summary

In the current period of IT development the creation of data manipulation and analysis facilities aimed at Web, social media, machine and sensor data, etc., is regarded as of paramount importance. The data of such scale (frequently measured in petabytes) are related to the category of the Big Data. To represent and manipulate collections of such data the new data models were created that differ of the traditional (relational) data models. One of the not yet solved problems of Big Data manipulation is the problem of integration of various non-traditional data models. For solving of such problem first of all it is required to create the unified representation of various kinds of non-traditional data models in the canonical information model (the generalized language unifying the languages of various data models). For such representation it is required to construct the data model mapping preserving semantics of its data description and data manipulation languages in the canonical one. Such mapping is required for the materialized integration (creation of a data warehouse) as well as for the virtual integration (by means of the subject mediators) of the respective collections of data. In the paper the principles of mapping of four kinds of non-traditional data models into the canonical model (for which the SYNTHESIS language is used representing the object-frame composed data model) are considered: the data models based on the multidimensional arrays; the graph-based data models; the NoSQL data models; the triple-based RDF data model. The method of data model mapping verification applied for the proof of information and operation preserving under the mapping is illustrated by examples. The objective of this research is the definition of well

founded unifying mappings of the non-traditional data models to present the possibility of the unified representation of so different data models for the materialized or virtual integration of the respective collections of data.