ВЕРИФИЦИРУЕМОЕ ОТОБРАЖЕНИЕ МОДЕЛИ ДАННЫХ, ОСНОВАННОЙ НА МНОГОМЕРНЫХ МАССИВАХ, В ОБЪЕКТНУЮ МОДЕЛЬ ДАННЫХ[1]

С. А. СТУПНИКОВ[2]

Аннотация. В работе рассматривается отображение модели данных, основанной на многомерных массивах (ММ-модели), в объектную модель данных. Изложены общие принципы отображения ММ-моделей в объектные модели данных. Рассмотрено отображение конкретных моделей – Array Data Model, использующейся в СУБД SciDB, в язык СИНТЕЗ, использующийся в качестве канонической модели данных в технологии предметных посредников. Проиллюстрирован метод верификации отображения – доказательства сохранения информации и семантики операций при отображении. Верификация осуществляется при помощи формального языка спецификаций AMN. Практической целью работы является создание базы для виртуальной или материализованной интеграции ресурсов, основанных на многомерных массивах.

Ключевые слова: многомерные массивы, объектная модель данных, отображение моделей данных, интеграция баз данных

1. Введение

Развитие науки и промышленности, широкое распространение информационных технологий ведет к накоплению огромных объемов данных как в науке, так и в бизнесе. Данные могут быть как наблюдательными, экспериментальными, так и полученными в ходе компьютерного моделирования. Данные таких масштабов (часто измеряемых уже в петабайтах) называются «большими данными» (Big Data) [1], плохо поддаются обработке и анализу в рамках хоро-

[1] Работа выполнена при поддержке РФФИ (грант 11-07-00402-а).

[2] Институт проблем информатики РАН, ssa@ipi.ac.ru

шо известных технологий баз данных, опирающихся в основном на реляционную модель данных.

Именно поэтому развиваются различные модели данных, нацеленные на параллельную обработку и анализ данных в распределенных средах – гридах и облаках. Одним из важных видов таких моделей 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 [2, 3]. Исследования ММ-моделей начались достаточно давно [4, 5] и продолжают развиваться. В данной статье рассматривается конкретная модель, а именно – модель, используемая в СУБД SciDB [6].

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. Одним из примеров поставщиков таких данных является строящийся телескоп LSST (Large Synoptic Survey Telescope) [7]. Был также сделан вывод о необходимости разработки СУБД нового поколения, которая должна удовлетворять, в частности, следующим требованиям [8]:

- модель данных основывается на многомерных массивах, а не на кортежах;
- модель хранения базируется на версионности, а не на обновлении значений;
- масштабируемость до сотен петабайт и высокая отказоустойчивость;
- СУБД является свободно распространяемым программным обеспечением.

Некоторое время спустя был запущен 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 системы для ОС Ubuntu и RedHat.

Целью данной статьи является исследование вопроса о верифицируемом отображении ММ-моделей, и в частности, Array Data Model (ADM) [9], использующейся в системе SciDB, в объектные модели данных для виртуальной или материализованной интеграции ресурсов при создании федеративных баз данных или хранилищ данных.

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

Виртуальная же интеграция рассматривается в статье применительно к предметным посредникам [10]. Предметные посредники представляют собой специальный вид программного обеспечения, образующий 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. 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 в частичные запросы над информационными ресурсами, затем исполняются на ресурсах. Результаты частичных запросов объединяются и выдаются пользователю также в терминах федеративной схемы.

Важным понятием технологии систем интеграции баз данных является каноническая модель, служащая общим языком, унифицирующим разнообразные модели ресурсов.

Необходимым предусловием интеграции ресурсов, основанных на многомерных массивах, является построение отображения соответствующей ММ-модели в каноническую модель данных, сохраняющего информацию и семантику операций языка манипулирования данными (ЯМД) [11]. Это обусловлено тем, что семантические отображения, связывающие федеративную схему и схемы ресурсов, нужно проводить в единой (канонической модели) [12]. Отображение должно быть верифицируемым – доказуемо правильным.

В качестве канонической модели в данной работе рассматривается язык СИНТЕЗ [13] – комбинированная слабоструктурированная и объектная модель данных, нацеленная на разработку предметных посредников для решения задач в средах неоднородных ресурсов. Разработан прототип программных средств для поддержки среды предметных посредников с языком СИНТЕЗ в роли канонической модели [14].

С точки зрения предметных посредников СУБД, основанные на многомерных массивах, представляют собой новый вид ресурсов, подлежащих интеграции в посредниках вместе с привычными ресурсами – реляционными и объектными СУБД, веб-сервисами и т.д.

Нужно отметить, что ADM подвергается некоторой критике со стороны исследователей, продолжающих развитие моделей, основанных на многомерных массивах. Так, авторы языка SciQL [15] 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, связанных с генетикой, страхованием, финансами. Сравнительное тестирование SciDB с СУБД Postgres и статистическим ПО R показало преимущества SciDB по производительности и масштабируемости.

Статья организована следующим образом. В разделе 2 рассмотрены и проиллюстрированы основные принципы отображения модели данных ADM в язык СИНТЕЗ. Принципы обобщены на случай моделей, отличных от ADM и СИНТЕЗ. В разделе 3 рассмотрен метод доказательства сохранения информации и семантики операций при отображении моделей с использованием формального языка спецификаций AMN [16]. Метод проиллюстрирован на структурах данных и операциях ЯМД моделей SciDB и СИНТЕЗ. В разделе 5 рассмотрены некоторые родственные исследования и направления дальнейшей работы.

2. Отображение модели ADM в язык СИНТЕЗ

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 отображение будут иллюстрироваться на адаптированных примерах из сценария применения SciDB в области оптической астрономии [17], а также на простых примерах из документации SciDB [9].

2.1 Отображение языка определения данных

Отображение ЯОД в данном разделе описывается независимо от вида интеграции – виртуальной или материализованной.

Основной единицей определения данных в модели ADM является массив, имеющий конечное количество *измерений* d_1, d_2, ..., d_n [9]. Длиной измерения называется количество упорядоченных значений в этом измерении. По умолчанию типом измерения являются 64-битные целые числа. Поддерживаются также нецелочисленные измерения, например строки или числа с плава-

ющей точкой. Каждая комбинация значений измерений соответствует ячейке массива, которая может содержать конечное количество значений, называемых *атрибутами.* Типом атрибута может быть один из встроенных типов ADM [9].

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

```
CREATE ARRAY source
< ampExposureId: int64 NULL,
   filterId: int8,
   apMag: double >
[ ra(double), de(double), objectId=0:*];
```

Создается массив оптических источников *source*, измерениями которого являются координаты *ra* и *de* типа *double* и целочисленный идентификатор объекта. Для целочисленного измерения указаны его нижняя (0) и верхняя (* - обозначающая бесконечность) границы. Ячейка массива состоит из трех атрибутов: *ampExposureId, filterId, apMag*. Указано, что атрибут *ampExposureId* может принимать неопределенное значение NULL. В данном примере приведены только некоторые из реально используемых атрибутов и измерений.

В языке СИНТЕЗ создание массива представляется определением одноименного класса:

```
{ source; in: class;
  instance_type:{
  double ra;
  ra2long: {in: function; params: {-ret/long}; };
  double de;
  de2long: {in: function; params: {-ret/long}; };
  long objectId; metaslot lower: 0; higher: inf; end
  objectIdBounds: {in: invariant;
    {{ all s (source(s) -> s.objectId >= 0) }}
  };
  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 не-NULL 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).

Заметим также, что отсутствие в коллекции объекта с некоторым набором значений измерений означает *пустую ячейку* в массиве.

Для нецелочисленных измерений *ra* и *de* кроме атрибутов в языке СИНТЕЗ определяются функции *ra2long, de2long*, преобразующие нецелочисленные значения в целочисленные. Необходимость привнесения этих функций вызвана следующим. При попытке описать операции, характерные для ММ-моделей, в объектной модели (в частности, в языке СИНТЕЗ), выясняется необходимость применения существенно различных механизмов работы с целочисленными и нецелочисленными измерениями. Это вызвано различием типов из-

мерений, возможной неравномерностью шага измерения и т.д. Для того, чтобы обеспечить возможность единообразного описания операций над целочисленными и нецелочисленными измерениями и необходимы функции, приводящие нецелочисленные измерения к целочисленным.

Ограничения, связанные с нижними и верхними границами целочисленных измерений, представляются в языке СИНТЕЗ во-первых, метаслотом соответствующего атрибута (например, *objectId*). В метаслоте хранится метаинформация, связанная с атрибутом, как с отдельной сущностью языка. В данном случае метаслот включает два слота *lower* и *higher*, отвечающих, соответственно, верхней и нижней границе измерения. Во-вторых, создается инвариант (например, *objectIdBounds*) предикативная спецификация которого устанавливает ограничения на значения измерения для каждого из объектов класса, отвечающего массиву. Спецификация инварианта имеет вид формулы первого порядка, где *all* – квантор существования, -> – импликация.

Необходимо отметить, что массив представляется в объектной модели множеством объектов класса (фактически, кортежей значений атрибутов). При этом наблюдается некоторое противоречие со стремлением создателей ММ-моделей отойти от моделей, основанных на кортежах. Однако, в контексте интеграции ресурсов ММ-модели - лишь один класс из большого множества разнообразных классов моделей данных. Представление специфических ММ-моделей в объектной модели является методологически и технически гораздо более простым и естественным, нежели использование многомерных массивов в качестве канонической модели.

Изложенные принципы отображения ЯОД могут быть обобщены на случай, когда канонической является объектная или объектно-реляционная модель, отличная от языка СИНТЕЗ. Также, непринципиальным является выбор модели данных, основанной на многомерных массивах. 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 не-NULL атрибутам ячейки массива, объявляются определенными (при помощи утверждений или инвариантов);
- для нецелочисленных измерений в типе экземпляров дополнительно определяются методы, преобразующие нецелочисленные значения в целочисленные;
- ограничения, связанные с нижними и верхними границами целочисленных измерений, отображаются при помощи инвариантов или встроенных утверждений о кардинальности соответствующих атрибутов. В случае использования инвариантов при отображении, границы измерений представляются также метаданными атрибута.

2.2 Отображение языка манипулирования данными

При интеграции баз данных для установления семантических соотношений между схемами ресурсов и федеративной схемой необходимо отображение ЯОД исходной модели в каноническую. 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 языке канонической модели исполняются непосредственно на данных. Отображение ЯМД нужно лишь для того, чтобы убедиться, что отображение моделей сохраняет информацию и семантику операций. Семантически правильное отображение служит базой для процесса Извлечения-Преобразования-Загрузки (ETL), формирующего из данных ресурса данные хранилища: ETL-процесс может быть выражен только в терминах канонической модели.

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 объекты класса *source* целиком, а лишь их атрибуты *ra, de.* Формально запись означает сокращение от *source(_/source.inst[ra, de]).* 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*.

При отображении ЯМД будут сначала рассмотрены основные конструкции языка программ СИНТЕЗ, соответствующие конструкциям языка AQL. Затем будут рассмотрены конструкции СИНТЕЗ, соответствующие конструкциям языка AFL.

Начнем рассмотрение с конструкций языка СИНТЕЗ, соответствующих конструкциям языка AQL, связанных с *извлечением* данных.

Предикаты-классы, условия, подзапросы. Рассмотрим программу, извлекающую координаты (*ra*, *de*) и апертурную звездную величину (*apMag*) астрономических источников из класса *source* с условием на 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* – некоторые 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.

Соединение классов. Соединение по определенным атрибутам (например, *objectId*)

```
q2([ra, de, filterId, uMag]) :-
```

```
    source([ra, de, objectId, fliterId]),
    objectSummary([objectId, uMag]).
```

представляется в AQL конструкцией JOIN-ON:

```
SELECT filterId, uMag INTO q2
FROM source
JOIN objectSummary
ON source.objectId = objectSummary.objectId;
```

где массив *objectSummary* имеет вид

```
CREATE ARRAY objectSummary
<uMag: float NULL,  gMag: float NULL>
[ objectId=0:* ];
```

Агрегация. Рассмотрим запрос, возвращающий объекты с минимальной звездной величиной *uMag*:

```
q([objectId, uMag]) :-
    objectSummary(obj/[objectId, uMag]), uMag = min(obj.uMag).
```

Запрос представляется в AQL с использованием агрегирующей функции того же рода:

```
SELECT uMag
FROM source, (SELECT min(uMag) AS min FROM Source)
WHERE uMag = min;
```

Группирование. Рассмотрим запрос, возвращающий среднее значение звездной величины *uMag*, вычисленное на группе по идентификатору объекта *filterId:*

```
q([objectId, avgMag]) :-
    group_by({objectId}, q2(obj/[ra,de,filterId, uMag])),
    avgMag = average(obj.uMag).
```

Здесь коллекция *q2*, на которой производится группирование по атрибуту *objectId* – результат соединения классов *source* и *objectSummary*, рассмотренный выше.

Очевидно, в AQL запрос представляется при помощи конструкции GROUP BY:

```
SELECT avg(uMag) AS avgMag
FROM q2 GROUP BY objectId;
```

Рассмотрим конструкции языка СИНТЕЗ, соответствующие конструкциям языка AQL, и связанные с *изменением* данных.

Обновление. Рассмотрим запрос, изменяющий значения в квадратной матрице (см. предыдущий пример) на значения с обратным знаком в том случае, если модуль значения больше 5.

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

Удаление. Рассмотрим программу, удаляющую из базы данных класс и все его содержимое:

```
-source(x) :- source(x).;
delete_frame(source).
```

В правилах со знаком – в голове осуществляется удаление объектов из коллекции. В данном случае из коллекции удаляются все объекты. Функция *delete_frame* удаляет коллекцию как объект из базы данных. Операция ; обозначает последовательную композицию программ. В AQL данный запрос представляется при помощи операции *DROP*:

```
DROP ARRAY source;
```

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) 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.

Разработанные отображения ЯОД и ЯМД были частично реализованы на языке ATL (ATLAS Transformation Language) [18]. ATL-программы представляют собой декларативно-императивные трансформации, реализующие отображения произвольных исходных моделей уровня M1 (согласно классификации MOF [19]), конформных исходной метамодели уровня M2, в целевые модели уровня M1, конформные целевой метамодели уровня M2. Модели уровня M1 являются схемами, представленными в канонической модели данных или модели ADM; модели уровня M2 есть описание абстрактного синтаксиса канонической модели или модели ADM. В качестве метамодели уровня M3, которой конформны метамодели уровня M2, рассматривается модель Ecore [20]. Синтаксис ЯОД и ЯМД ядра канонической информационной модели (языка СИНТЕЗ) и модели ADM были представлены в метамодели Ecore.

Было осуществлено построение ATL-трансформаций, реализующих отображения подмножества ЯОД модели ADM в ЯОД канонической модели и

подмножества ЯМД канонической модели в ЯМД модели ADM. Подмножества ЯМД определялись конструкциями ЯОД и ЯМД канонической модели, поддерживаемыми в настоящее время в исполнительной среде предметных посредников. Поддерживаемый язык запросов канонической модели включает правила, в голове которых могут быть предикаты-коллекции, а в теле – конъюнкция предикатов-коллекций, условий соединения коллекций и других условий на значения атрибутов типов экземпляров коллекций. Условием соединения может быть только равенство атрибутов. Поддерживаются основные арифметические предикаты и функции, а также термы - вызовы функций.

3 Сохранение информации и семантики операций ЯМД при отображении

Метод доказательства сохранения информации и семантики операций при отображении моделей данных [21] основывается на представлении семантики моделей в формальном языке спецификаций AMN [16].

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 *S* и целевую модель *T*. Построим отображение θ модели *S* в модель *T* (подобно изложенному 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 опе-

рациями 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 [22].

В качестве иллюстрации основных принципов выражения семантики моделей ADM и СИНТЕЗ в AMN рассмотрим частичные AMN-спецификации, соответствующие данным моделям.

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, attributeNames,
    instanceType, typeAttributes, attributeType,
    unique, obligatory,
    intAttributeLowerBound, intAttributeHigherBound,
    objectIDs, objectType, objectsOfClass,
    integerAttributeValue,
    adtAttributeValue
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
```

Множества уникальных атрибутов типов *unique* и множества определенных атрибутов типов *obligatory* представлены тотальными функциями из множества имен типов в множество подмножеств атрибутов:

```
unique: typeNames --> POW(dom(attributeNames))&
obligatory: typeNames --> POW(dom(attributeNames))
```

Нижние границы целочисленных атрибутов (переменная *intAttributeLowerBound*) представлены частичной функцией из множества атрибутов в множество целых чисел:

```
intAttributeLowerBound: dom(attributeNames) +-> INT
```

Аналогично представляются верхние границы.

Идентификаторы объектов (переменная *objectIDs*) представлены подмножеством натуральных чисел:

```
objectIDs: POW(NAT)
```

Типы объектов (переменная *objectType*) представлены тотальной функцией из множества объектных идентификаторов в множество имен типов:

```
objectType: objectIDs --> typeNames
```

Состав классов (переменная *objectsOfClass*) представлен тотальной функцией из множества имен классов в множество подмножеств идентификаторов объектов:

```
objectsOfClass: classNames --> POW(objectIDs)
```

Значения атрибутов объектов (переменные *integerAttributeValue, adtAttributeValue* и т.д.) представлены функциями из множества атрибутов в функции из множества идентификаторов объектов в множества значений атрибутов. Для простоты рассмотрены лишь функции для целочисленных атрибутов и атрибутов со значениями АТД (объектами):

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

Дополнительные необходимые свойства переменных состояния представлены конъюнктивными компонентами инварианта. Так, каждый атрибут является атрибутом некоторого типа:

```
UNION(tt).(tt: typeNames | typeAttributes(tt)) =
    dom(attributeNames)
```

Здесь *UNION* – родовая операция объединения, в данном случае объединяются множества атрибутов *typeAttributes(tt)* по всем именам типов *tt* из множества *typeNames*.

Никакой атрибут не принадлежит двум типам одновременно:

```
!(t1, t2).(t1: typeNames & t2: typeNames =>
    (typeAttributes(t1) /\ typeAttributes(t2) = {}))
```

Здесь ! – знак квантора всеобщности, => – логическая импликация, /\ – символ пересечения множеств, {} – пустое множество.

Уникальные и определенные атрибуты типа выбираются из множества атрибутов типа:

```
!(tt).(tt: dom(unique) => unique(tt) <: typeAttributes(tt)) &
!(tt).(tt: dom(obligatory) =>
    obligatory(tt) <: typeAttributes(tt))
```

Здесь <: – символ отношения множество-подмножество.

Нижние и верхние границы могут быть определены только для целочисленных атрибутов:

```
!(attr).(attr: dom(intAttributeLowerBound) =>
    attributeType(attr) = Integer) &
```

Тип объекта – экземпляра класса есть тип экземпляров этого класса:

```
!(cc).(cc: classNames =>
    !(oo).(oo: objectsOfClass(cc) =>
       objectType(oo) = instanceType(cc))) &
```

Для каждого атрибута определена ровно одна функция значений:

```
dom(adtAttributeValue) /\ dom(integerAttributeValue) = {} &
dom(adtAttributeValue) \/ dom(integerAttributeValue) =
   dom(attributeNames)
```

Здесь \/ - символ объединения множеств.

Для любого объекта и любого определенного атрибута типа этого объекта функция значений атрибута определена на объекте:

```
!(oo, aa).(oo: dom(objectType) &
```

```
    aa: typeAttributes(objectType(oo)) &
    aa: obligatory(objectType(oo)) =>
        (attributeType(aa) = Integer =>
            oo: dom(integerAttributeValue(aa))) &
        (attributeType(aa) = ADT =>
            oo: dom(adtAttributeValue(aa))  ) &
```

Для любого объекта и любого целочисленного атрибута типа объекта, определенного на объекте и для которого определена нижняя (верхняя) граница, значение атрибута не меньше (не больше) нижней (верхней) границы:

```
!(oo, aa).(oo: objectIDs &
    aa: typeAttributes(objectType(oo)) &
    oo: dom(integerAttributeValue(aa) =>
    (aa: dom(intAttributeLowerBound) =>
        (integerAttributeValue(aa)(oo) >=
         intAttributeLowerBound(aa))) ) &
```

Объект однозначно идентифицируется набором своих уникальных атрибутов:

```
!(oo1, oo2).(oo1: objectIDs & oo2: objectIDs &
    objectType(oo1) = objectType(oo2) &
    unique(objectType(oo1)) /= {} &
    !(aa).(aa: unique(objectType(oo1)) =>
        (attributeType(aa) = Integer =>
              integerAttributeValue(aa)(oo1) =
                   integerAttributeValue(aa)(oo2)) &
        (attributeType(aa) = ADT =>
               adtAttributeValue(aa)(oo1) =
                    adtAttributeValue(aa)(oo2)) ) =>
    oo1 = oo2 )
```

Из всего ЯМД в спецификации рассмотрена единственная операция *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 2.2. 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, dimensionNames, cellAttributeNames,
    arrayDimensions, arrayCellAttributes,
    cellAtrributeType, nullable,
    dimLowerBound, dimHigherBound,
    cells, dimensionValue, integerCellAttributeValue
```

Имена массивов представлены переменной *arrayNames*; имена измерений – переменной *dimensionNames*; имена атрибутов ячеек массива – переменной *cellAttributeNames*; принадлежность измерений массивам – переменной *arrayDimensions*; принадлежность атрибутов ячеек – переменной *arrayCellAttributes*; тип атрибута ячейки – переменной *cellAtrributeType*; атрибуты ячеек массивов, которые могут принимать неопределенные значения – переменной *nullable*; верхние (нижние) границы измерений – переменной *dimLowerBound* (*dimHigherBound*); множества идентификаторов ячеек массивов – переменной *cells*, значения измерений в ячейках – переменной *dimensionValue*; 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) &
    dimensionNames: NAT +-> STRING_Type &
    cellAttributeNames: NAT +-> STRING_Type &
    arrayDimensions: arrayNames --> POW(dom(dimensionNames)) &
    arrayCellAttributes: arrayNames -->
        POW(dom(cellAttributeNames)) &
    cellAtrributeType:
       dom(cellAttributeNames) --> BuiltInTypes &
```

```
    nullable: dom(cellAttributeNames) --> BOOL &
    dimLowerBound: dom(dimensionNames) --> INT &
    dimHigherBound: dom(dimensionNames) +-> INT &
    cells: arrayNames --> POW(NAT) &
    dimensionValue: NAT*dom(dimensionNames) +-> INT  &
    integerCellAttributeValue:
        NAT*dom(cellAttributeNames) +-> INT &
```

Здесь * - знак декартова произведения множеств.

Дополнительные необходимые свойства переменных состояния представлены конъюнктивными компонентами инварианта. Так, любая ячейка любого массива однозначно идентифицируется набором значений измерений:

```
!(arr, cell1, cell2).(arr: arrayNames &
  cell1: cells(arr) &  cell2: cells(arr) &
  !(dim).(dim: arrayDimensions(arr) =>
    dimensionValue(cell1, dim) = dimensionValue(cell2, dim)) =>
    cell1 = cell2)
```

Для любой ячейки любого массива определены значения всех измерений и значение по крайней мере одного атрибута:

```
!(arr, cell).(arr: arrayNames & cell: cells(arr) =>
  !(dim).(dim: arrayDimensions(arr) =>
    (cell |-> dim): dom(dimensionValue)) &
    #(attr).(attr: arrayCellAttributes(arr) &
      cellAtrributeType(attr) = Integer &
      (cell, attr): dom(integerCellAttributeValue)) )
```

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 их в одну конъюнкцию.

Так, множество имен массив совпадает с множеством имен классов:

```
classNames = arrayNames
```

Множество идентификаторов и имен измерений и атрибутов ячеек совпадает с множеством идентификаторов и имен атрибутов типов экземпляров классов:

```
attributeNames = dimensionNames \/ cellAttributeNames
```

Любому измерению любого массива соответствует атрибут типа экземпляра класса, соответствующего этому массиву:

```
!(arr, dim).(arr: arrayNames & dim: arrayDimensions(arr) =>
    #(attr).(attr: typeAttributes(instanceType(arr)) &
             attr = dim & attributeType(attr) = Integer) )s
```

Любому атрибуту ячейки любого массива соответствует атрибут типа экземпляра класса, соответствующего этому массиву и типы атрибутов совпадают:

```
!(arr, cattr).(arr: arrayNames &
   cattr: arrayCellAttributes(arr) =>
    #(attr).(attr: typeAttributes(instanceType(arr)) &
         attr = cattr &
         attributeType(attr) = attributeType(cattr)))
```

Атрибут ячейки массива, который может принимать неопределенные значения, соответствует определенному (*obligatory*) атрибуту типа:

```
!(arr, cattr).(arr: arrayNames & cattr /: dom(nullable) &
    cattr: arrayCellAttributes(arr) =>
    cattr: obligatory(instanceType(arr)) )
```

Здесь знак /: обозначает отношение непринадлежности элемента множеству.

Измерения соответствуют уникальным атрибутам типов:

```
!(arr, dim).(arr: arrayNames &
    dim: arrayDimensions(arr) => dim: unique(instanceType(arr)) )
```

Верхние (нижние) границы измерений равны верхним (нижним) границам соответствующих атрибутов типов:

```
!(dim).(dim: dom(dimLowerBound) =>
    dim: dom(intAttributeLowerBound) &
    dimLowerBound(dim) = intAttributeLowerBound(dim))
```

Непустые ячейки массивов соответствуют объектам классов:

```
cells = objectsOfClass
```

Для любой ячейки значения ее измерений и определенных атрибутов совпадают со значениями соответствующих атрибутов объекта, соответствующего ячейке:

```
!(cell, dim).(cell: NAT & dim: NAT &
  (cell |-> dim): dom(dimensionValue) =>
  cell: dom(integerAttributeValue(dim)) &
  dimensionValue(cell, dim) = integerAttributeValue(dim)(cell))
&
!(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 [22]. Автоматически были сгенерированы теоремы, выражающие уточнение спецификаций. В частности, для операции *update* были сгенерированы 10 теорем. 3 из них были доказаны автоматически, для доказательства остальных необходимо применять интерактивные средства доказательства.

4 Родственные исследования и направления дальнейшей работы

Родственными данной работе следует считать работы, связанные 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 начались одновременно с

исследованиями моделей, основанных на многомерных массивах [5], и продолжаются в настоящее время [23].

Основные особенности данной работы 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. В качестве целевой модели используется объектная модель с Datalog-подобным языком запросов (программ) – язык СИНТЕЗ. Для отображения обеспечивается формальное доказательство сохранения информации и семантики операций ЯМД.

Отметим, что результаты работы 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). Результаты также могут быть использованы для интеграции ресурсов, представленных в модели, основанной на многомерных массивах, но отличной от ADM.

Некоторые вопросы отображения требуют дальнейших исследований, например, следует ли иметь в канонической модели при интеграции массив-ориентированных моделей данных операции, связанные с размером порции (chunk size) данных в БД [9].

Дальнейшую работу можно разбить на два этапа:

- расширение инструментальных средств поддержки предметных посредников для виртуальной интеграции SciDB-ресурсов: (i) расширение средств регистрации ресурсов в посреднике [10] трансформацией ЯОД ADM в каноническую модель; (ii) создание SciDB-адаптера - специального ПО, связывающего исполнительную среду посредников с SciDB-ресурсами (составной частью адаптера является разработанная трансформация ЯМД);

- применение технологии предметных посредников для решения научных задач в некоторой предметной области над множеством неоднородных ресурсов, включающим SciDB-ресурсы.

Благодарности

Автор выражает благодарность Л. А. Калиниченко, П. Е. Велихову и А. Е. Вовченко за полезные замечания, высказанные в ходе обсуждения данной работы на семинарах ИПИ РАН.

Список литературы

1. Challenges and Opportunities with Big Data // A community white paper developed by leading researchers across the United States. – 2012. http://cra.org/ccc/docs/init/bigdatawhitepaper.pdf
2. P. Vassiliadis and T. K. Sellis. A Survey of Logical Models for OLAP Databases. SIGMOD Record, 28(4):64–69, 1999.Abrial J.-R. The B-Book: Assigning Programs to Meanings. Cambridge: Cambridge University Press, 1996.
3. Torben Bach Pedersen, Christian S. Jensen. Multidimensional Database Technology. IEEE Computer, vol. 34, no. 12, pp. 40-46, 2001.
4. Leonid Libkin, Rona Machlin, Limsoon Wong. A Query Language for Multidimensional Arrays: Design, Implementation, and Optimization Techniques. SIGMOD, 1996. – P.228-239.
5. P. Baumann. A Database Array Algebra for Spatio-Temporal Data and Beyond. In Next Generation Information Technologies and Systems, pages 76–93, 1999.
6. Overview of SciDB: Large Scale Array Storage, Processing and Analysis. The SciDB Development team. SIGMOD, 2010.
7. Large Synoptic Survey Telescope. http://www.lsst.org
8. J. Becla, K.-T. Lim. Report form the First Workshop on Extremely Large Databases. Data Science Journal, Volume 7, 2008.
9. SciDB User's Guide. Version 12.3. – 2012. http://www.scidb.org/
10. 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. - Volume Databases and Information Systems Integration. - P. 246-251.

11. 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., 2007. – Т. 1, Вып. 2. – С. 15-38.
12. 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.
13. 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.
14. 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., 2008. – Т. 2, Вып. 1. – С. 2-34.
15. Martin L. Kersten, Ying Zhang, Milena Ivanova, Niels Nes: SciQL, a query language for science applications. EDBT/ICDT Array Databases Workshop 2011:1-12
16. Abrial J.-R. The B-Book: Assigning Programs to Meanings. Cambridge: Cambridge University Press, 1996.
17. Astronomy in ArrayDB. http://trac.scidb.org/raw-attachment/wiki/UseCases/Astronomy%20in%20ArrayDB.pdf
18. ATL Project, http://www.eclipse.org/m2m/atl/
19. Budinsky, F., Steinberg, D., Ellersick, R., Grose, T. Eclipse Modeling Framework, Chapter 5 Ecore Modeling Concepts. – Addison Wesley Professional, (2004)
20. Meta Object Facility (MOF) 2.0 Core Specification (2003), http://www.omg.org/cgi-bin/apps/doc?ptc/03-10-04.pdf
21. 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. - V. 1: Regular Papers. - P. 275-284.
22. Atelier B, the industrial tool to efficiently deploy the B Method. http://www.atelierb.eu/index-en.php
23. Alex van Ballegooij. RAM: Array Database Management through Relational Mapping. SIKS Dissertation Series, N 2009-25. – 2009. http://oai.cwi.nl/oai/asset/14074/14074D.pdf