Унификация модели данных, основанной на многомерных массивах, при интеграции неоднородных информационных ресурсов

© С. А. Ступников
ИПИ РАН,
Москва
ssa@ipi.ac.ru

Аннотация

В работе рассматривается отображение одной из моделей данных, основанной на многомерных массивах – модели СУБД SciDB в каноническую модель систем виртуальной или материализованной интеграции баз данных. Для определенности в статье рассматривается отображение в объектную модель данных – язык СИНТЕЗ, использующийся в качестве канонической модели данных в технологии предметных посредников. Проиллюстрирован метод доказательства сохранения информации и семантики операций при отображении моделей данных с использованием формального языка спецификаций. Целью работы является создание обоснованной теоретической базы для интеграции ресурсов, основанных на многомерных массивах.

Работа выполнена при поддержке РФФИ (гранты 10-07-00342-а, 11-07-00402-а) и Президиума РАН (программа 16П, проект 4.2).

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) [7], плохо поддаются обработке и анализу в рамках хорошо 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 [18, 16]. Исследования ММ-моделей начались достаточно давно [14, 5] и продолжают развиваться. В данной статье рассматривается конкретная модель, а именно – модель, используемая в СУБД SciDB [15].

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) [13]. Был также сделан вывод о необходимости разработки СУБД нового поколения, которая должна удовлетворять, в частности, следующим требованиям [6]:

- 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 об *унификации* модели данных системы SciDB (будем называть ее в дальнейшем ADM – Array Data Model [17]) для виртуальной или материализованной интеграции ресурсов при создании федеративных баз данных или хранилищ данных.

Труды 14-й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» — RCDL-2012, Переславль-Залесский, Россия, 15-18 октября 2012 г.

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 [9]. 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.

Унификацией некоторой исходной модели данных называется ее отображение в каноническую модель, сохраняющее информацию и семантику операций языка манипулирования данными (ЯМД) [20]. Унификация должна быть доказуемо правильной. Очевидно, что унификация моделей ресурсов является необходимым предусловием для регистрации ресурсов в посреднике – семантические отображения, связывающие федеративную схему и схемы ресурсов, нужно проводить в единой (канонической модели) [10].

В качестве канонической модели в данной работе рассматривается язык СИНТЕЗ [11] - 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 [19].

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 [12] 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. В разделе 3 рассмотрен метод доказательства сохранения информации и семантики операций при отображении моделей с использованием формального языка спецификаций AMN [1]. Метод проиллюстрирован на структурах данных и операциях ЯМД моделей SciDB и СИНТЕЗ.

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 [2], а также на простых примерах из документации SciDB [17].

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* $d_1, d_2, \ldots, d_n$ [17]. 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 плавающей 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 [17].

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 измерений, 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 взаимно-однозначное 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

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

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.

Функциональные предикаты. Рассмотрим запрос, конструирующий из двух звездных величин *uMag, rMag* в разных фильтрах цветовую разность:

```
q([objectId, ugColor]) :-
        objectSummary([objectId, uMag, rMag]),
        diff(uMag, rMag, ugColor).
```

Запрос включает функциональный предикат, семантика которого определяется функцией *diff*:

```
{ diff; in: function;
  params: {+min/float, +sub/float, -res/float};
  {{  res = min - sub }}
}
```

В двойных фигурных скобках заключена предикативная спецификация функции, связывающая значения входных и выходных параметров в формуле типизированного исчисления первого порядка.

Для простоты изложения рассматривается функция, возвращающая разность параметров.

В AQL запрос представляется следующим образом:

```
SELECT uMag – rMag AS ugColor
FROM objectSummary;
```

где массив *objectSummary* имеет вид

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

Таким образом, если результат функции описывается выражением, составленным из операций над простыми встроенными типами, то в AQL подобное выражение может быть в явном виде вынесено в оператор SELECT.

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;
```

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;
```

Агрегация в бегущем окне (Window Aggregation). Такой вид агрегации является специфическим для моделей, основанных на многомерных массивах. *Окно* определяется своим размером в каждом измерении массива. Центроид окна располагается в каждом из элементов массива и для каждого такого окна вычисляется некоторая агрегирующая функция. Например, для вычисления среднего значения в бегущем окне размера 3x3 в квадратной матрице 4x4, определяемой в AQL следующим образом:

```
CREATE ARRAY source  <val: double>[i = 0:3, j = 0:3];
```

необходим запрос

```
q([i, j, avgVal]) :- source(x/[i, j]),
        windowAggr(i, j, avgVal).
```

где семантика предиката *windowAggr* определяется функцией

```
{ windowAggr; in: function;
  params: {+i0/long, +j0/long, -res/double};
  {{  source(x/[i, j, val]) & i >= i0 - 1 & i <= i0 + 1 &
      j >= j0 - 1 & j <= j0 + 1 & res = average(x.val)  }}
}
```

вычисляющей среднее значение в окне с заданным центроидом в классе *source*.

AQL предлагает для агрегации в бегущем окне специальную конструкцию WINDOW:

```
SELECT avg(val) AS avgVal
FROM source WINDOW 3, 3;
```

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;
```

Продолжим описание отображения для конструкций языка СИНТЕЗ, соответствующих конструкциям AFL. Для экономии места будет рассмотрено лишь ограниченное количество конструкций. Рассматриваемые конструкции связаны только с *извлечением* данных.

Вычисление значений новых атрибутов. Рассмотрим запрос, вычисляющий значение нового атрибута – расстояние в километрах по имеющемуся расстоянию в милях:

```
q([i, miles, kilometers]) :-
      distance([i, miles]), kilometers = 1.6*miles.
```

В ADM данный запрос представляется с использованием операции *apply* языка AFL:

```
SELECT *
FROM apply(distance, kilometers, 1.6*miles);
```

Увеличение размерности массива и конкатенация массивов. Под *массивом* имеется ввиду структура данных, полученная в результате отображения массива модели ADM в язык СИНТЕЗ (раздел 2.1). Рассмотрим программу, соединяющую в двумерный массив два вектора:

```
q([x, y, val]) :- vector1([x, val]), y =1.
q([x, y, val]) :- vector2([x, val]), y = 2.
```

В ADM запрос представляется с использованием операций *adddim* и *substitute*:

```
SELECT *
FROM concat(adddim(vector1, y), adddim(vector2, y));
```

Поиск (lookup). При особом виде соединения - поиске - наборы атрибутов из ячеек одного массива (*pattern*) используются как координаты ячеек другого массива (*source*). Результирующий массив имеет ту же форму, что и первый массив, и содержит значения из ячеек второго массива:

```
q([k, val]) :- pattern([k, i, j]), source([i, j, val]).
```

В ADM запрос представляется с использованием операции *lookup*:

```
SELECT * FROM lookup(pattern, source);
```

Расширение 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);
```

Умножение матриц. Умножение матриц в объектной модели (как и в реляционной) требует соединения матриц по разным индексам и дальнейшей группировки с суммированием произведений элементов матриц:

```
r([i, j, val1, val2]) :-  left([i, j1/j, val1/val]),
      right([i1/i, j, val2/val]), j1 = i1.
q([i, j, val]) :- group_by({i, j}, r(x/[i, j, val1, val2])),
      val = sum(x.val1*x.val2).
```

В ADM запрос представляется с использованием операции *multiply*:

```
SELECT * FROM multiply(left, right);
```

На основании рассмотренного отображения ЯМД можно заметить, что многие из операций AFL (например, *xgrid*) имеют достаточно сложно устроенный прообраз в канонической модели. 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.

В данной статье не рассматриваются операции AFL, имеющие альтернативное представление конструкциями AQL - соединение, агрегация, фильтрация и т.д. Не рассматриваются также операции, тесно связанные с реализацией, а не с моделью данных – чтение и запись файлов, загрузка подключаемых модулей, вывод служебной информации и т.д.

3 Сохранение информации и семантики операций ЯМД при отображении

Метод доказательства сохранения информации и семантики операций при отображении моделей данных [8] основывается на представлении семантики моделей в формальном языке спецификаций AMN [1].

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 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 [3].

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

Часть спецификации, соответствующая объектной модели языка СИНТЕЗ, выглядит следующим образом:

```
1   REFINEMENT  ObjectDM
2   ABSTRACT_VARIABLES
3     typeNames, classNames, attributeNames,
4     instanceType, typeAttributes, attributeType,
5     unique, obligatory,
6     intAttributeLowerBound, intAttributeHigherBound,
7     objectIDs, objectType, objectsOfClass,
8     integerAttributeValue
9   INVARIANT
10    typeNames: POW(STRING_Type) &
11    classNames: POW(STRING_Type) &
12    attributeNames: NAT +-> STRING_Type &
13    instanceType: classNames --> typeNames &
14    typeAttributes: typeNames -->
15       POW(dom(attributeNames)) &
16   attributeType: dom(attributeNames) +->BuiltInTypes&
17    unique: typeNames --> POW(dom(attributeNames))&
18    obligatory: typeNames -->
19       POW(dom(attributeNames)) &
20    intAttributeLowerBound:
21       dom(attributeNames) +-> INT &
22    objectIDs: POW(NAT) &
23    objectType: objectIDs --> typeNames &
24    objectsOfClass: classNames --> POW(objectIDs) &
25    !(cc).(cc: classNames =>
26      !(oo).(oo: objectsOfClass(cc) =>
27      objectType(oo) = instanceType(cc))) &
28    integerAttributeValue:
29      dom(attributeNames) +-> (objectIDs +-> INT) &
30    !(oo, aa).(oo: dom(objectType) &
31       aa: typeAttributes(objectType(oo)) &
32       aa: obligatory(objectType(oo)) =>
33          (attributeType(aa) = Integer =>
34             oo: dom(integerAttributeValue(aa)))) &
35    !(oo, aa).(oo: objectIDs &
36       aa: typeAttributes(objectType(oo)) &
37       oo: dom(integerAttributeValue(aa) =>
38       (aa: dom(intAttributeLowerBound) =>
39          (integerAttributeValue(aa)(oo) >=
40           intAttributeLowerBound(aa))) ) &
41    !(oo1, oo2).(oo1: objectIDs & oo2: objectIDs &
42       objectType(oo1) = objectType(oo2) &
43       unique(objectType(oo1)) /= {} &
44       !(aa).(aa: unique(objectType(oo1)) =>
45          (attributeType(aa) = Integer =>
46             integerAttributeValue(aa)(oo1) =
47                integerAttributeValue(aa)(oo2)) ) =>
48       oo1 = oo2 )
49  OPERATIONS
```

```
50      update(cls, attr, exp, cond) =
51      PRE cls: classNames &
52            attr: typeAttributes(instanceType(cls)) &
53            attributeType(attr) = Integer &
54            exp: INT --> INT & cond: NAT --> BOOL
55      THEN
56        integerAttributeValue :=
57        integerAttributeValue <+
58        { xx | xx: (NAT*(NAT<->INT)) &
59          #(oo, val).( oo: objectsOfClass(cls) & val: INT &
60          xx = attr |-> ({oo |-> val}) &
61          (cond(integerAttributeValue(attr)(oo)) = TRUE =>
62              val = exp(integerAttributeValue(attr)(oo))) &
63          (cond(integerAttributeValue(attr)(oo)) = FALSE
64              => val = integerAttributeValue(attr)(oo))  ) }
65      END
66    END
```

Переменные, составляющие пространство состояний объектной модели, объявлены в разделе ABSTRACT_VARIABLES машины *ObjectDM* и типизируются в разделе INVARIANT. Так, имена типов и классов представлены переменными *typeNames, classNames,* тип которых – подмножество множества строк (строки спецификации 10-11). Атрибуты (переменная *attributeNames*) представлены частичной функцией, ставящей в соответствие уникальному идентификатору атрибута (натуральному числу) имя атрибута (строка 12). Типы экземпляров классов (переменная *instanceType*) представлены тотальной функцией из множества имен классов в множество имен типов (13). Принадлежность атрибутов типам (переменная *typeAttributes*) выражена тотальной функцией из множества имен типов в множество подмножеств атрибутов (14-15). Типы значений атрибутов (переменная *attributeType*) представлены функцией из множества атрибутов в множество встроенных типов данных (16). Такими же функциями представлены множества уникальных атрибутов типов *unique* (17) и множества определенных атрибутов *obligatory* (18-19). Нижние границы целочисленных атрибутов (переменная *intAttributeLowerBound*) представлены частичной функцией из множества атрибутов в множество целых чисел (20-21). Аналогично представляются верхние границы. Идентификаторы объектов (переменная *objectIDs*) представлены подмножеством натуральных чисел (22). Типы объектов (переменная *objectType*) представлены тотальной функцией из множества объектных идентификаторов в множество имен типов (23). Состав классов (переменная *objectsOfClass*) представлен тотальной функцией из множества имен классов в множество подмножеств идентификаторов объектов (24). Значения целочисленных атрибутов объектов (переменные *integerAttributeValue*) представлены функциями из множества атрибутов в функции из множества идентификаторов объектов в множества значений атрибутов (28-29). Функции для нецелочисленных атрибутов определяются аналогично.

Некоторые дополнительные необходимые свойства переменных состояния представлены конъюнктивными компонентами инварианта:

- тип объекта - экземпляра класса есть тип экземпляров этого класса (25-27);
- для любого объекта и любого определенного атрибута типа этого объекта функция значений атрибута определена на объекте (30-34);
- 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 (35-40);
- объект однозначно идентифицируется набором своих уникальных атрибутов (41-48).

Из всего ЯМД в спецификации рассмотрена единственная операция *update* обновления значений атрибута в объектах класса. 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* (50-65) 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 данной спецификации для простоты не рассмотрены некоторые черты объектной модели, например отношения тип-подтип и класс-подкласс.

Рассмотрим теперь часть спецификации, соответствующей модели ADM:

```
1     REFINEMENT ArrayDM
2     REFINES ObjectDM
3     ABSTRACT_VARIABLES
4       arrayNames, dimensionNames, cellAttributeNames,
5       arrayDimensions, arrayCellAttributes,
6       cellAtrributeType, nullable,
7       dimLowerBound, dimHigherBound,
8       cells, dimensionValue, integerCellAttributeValue
9     INVARIANT
10      arrayNames: POW(STRING_Type) &
11      dimensionNames: NAT +-> STRING_Type &
12      cellAttributeNames: NAT +-> STRING_Type &
13      arrayDimensions:
14        arrayNames --> POW(dom(dimensionNames)) &
15      arrayCellAttributes:
16        arrayNames --> POW(dom(cellAttributeNames)) &
17      cellAtrributeType:
18        dom(cellAttributeNames) --> BuiltInTypes &
19      nullable: dom(cellAttributeNames) --> BOOL &
20      dimLowerBound: dom(dimensionNames) --> INT &
```

```
21      dimHigherBound: dom(dimensionNames) +-> INT &
22      cells: arrayNames --> POW(NAT) &
23      dimensionValue:
24         NAT*dom(dimensionNames) +-> INT  &
25      integerCellAttributeValue:
26         NAT*dom(cellAttributeNames) +-> INT &
27      !(arr, cell1, cell2).(arr: arrayNames &
28          cell1: cells(arr) &  cell2: cells(arr) &
29          !(dim).(dim: arrayDimensions(arr) =>
30             dimensionValue(cell1, dim) =
31               dimensionValue(cell2, dim))
32         => cell1 = cell2 ) &
33      !(arr, cell).(arr: arrayNames & cell: cells(arr) =>
34         !(dim).(dim: arrayDimensions(arr) =>
35           (cell |-> dim): dom(dimensionValue)) &
36         #(attr).(attr: arrayCellAttributes(arr) &
37            cellAtrributeType(attr) = Integer &
38            (cell, attr): dom(integerCellAttributeValue)) )
39   OPERATIONS
40   update(cls, attr, exp, cond) =
41   PRE cls: arrayNames & attr: arrayCellAttributes(cls) &
42         cellAtrributeType(attr) = Integer &
43         exp: INT --> INT & cond: NAT --> BOOL
44   THEN
45   integerCellAttributeValue :=
46   integerCellAttributeValue <+
47   { yy | yy: (NAT*NAT)*INT &
48    #(cell, val).(cell: cells(cls) & val: INT &
49    yy = ((cell |-> attr)|-> val) &
50    (cond(integerCellAttributeValue(cell, attr)) = TRUE =>
51      val = exp(integerCellAttributeValue(cell, attr))) &
52    (cond(integerCellAttributeValue(cell, attr))= FALSE=>
53      val = integerCellAttributeValue(cell, attr)) ) }
54   END
55   END
```

Имена массивов представлены переменной *arrayNames* (строка спецификации 10); имена измерений – переменной *dimensionNames* (строка 11); имена атрибутов ячеек массива – переменной *cellAttributeNames* (12); принадлежность измерений массивам – переменной *arrayDimensions* (13-14); принадлежность атрибутов ячеек – переменной *arrayCellAttributes* (15-16); тип атрибута ячейки – переменной *cellAtrributeType* (17-18); атрибуты ячеек массивов, которые могут принимать неопределенные значения – переменной *nullable* (19); верхние (нижние) границы измерений – переменной *dimLowerBound* (*dimHigherBound*) (20-21); множества идентификаторов ячеек массивов – переменной *cells* (22), значения измерений в ячейках – переменной *dimensionValue* (23-24); значения атрибутов ячеек – переменной *integerCellAttributeValue* (25-26). Переменные типизируются при помощи частичных и тотальных функций аналогично переменным, использующимся для придания семантики объектной модели.

Дополнительные необходимые свойства переменных состояния представлены конъюнктивными компонентами инварианта:

- любая ячейка любого массива однозначно идентифицируется набором значений измерений (27-32);
- для любой ячейки любого массива определены значения всех измерений и значение по крайней мере одного атрибута (33-38).

Аналогично объектной модели рассмотрена единственная операция ЯМД – операция обновления *update* (40-54). 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 уточнения выглядит следующим образом:

```
1    classNames = arrayNames &
2    attributeNames=dimensionNames\/cellAttributeNames&
3    !(arr, dim).(arr: arrayNames &
4      dim: arrayDimensions(arr) =>
5       #(attr).(attr: typeAttributes(instanceType(arr)) &
6        attr = dim & attributeType(attr) = Integer) ) &
7    !(arr, cattr).(arr: arrayNames &
8      cattr: arrayCellAttributes(arr) =>
9       #(attr).(attr: typeAttributes(instanceType(arr)) &
10     attr=cattr& attributeType(attr)=attributeType(cattr)))&
11   !(arr, cattr).(arr: arrayNames & cattr /: dom(nullable) &
12     cattr: arrayCellAttributes(arr) =>
13     cattr: obligatory(instanceType(arr)) ) &
14   !(arr, dim).(arr: arrayNames &
15     dim: arrayDimensions(arr) =>
16       dim: unique(instanceType(arr)) ) &
17   !(dim).(dim: dom(dimLowerBound) =>
18     dim: dom(intAttributeLowerBound) &
19     dimLowerBound(dim) =
20       intAttributeLowerBound(dim)) &
21   cells = objectsOfClass &
22   !(cell, dim).(cell: NAT & dim: NAT &
23     (cell |-> dim): dom(dimensionValue) =>
24     cell: dom(integerAttributeValue(dim)) &
25      dimensionValue(cell, dim) =
26        integerAttributeValue(dim)(cell) ) &
27   !(cell, cattr).(cell: NAT & cattr: NAT &
28     (cell |-> cattr): dom(integerCellAttributeValue) =>
29     cell: dom(integerAttributeValue(cattr)) &
30     integerCellAttributeValue(cell, cattr) =
31       integerAttributeValue(cattr)(cell) )
```

Инвариант формализует принципы отображения ЯОД, изложенные в разделе 2.1:

- множество имен массивов совпадает с множеством имен классов (строка 1);
- множество идентификаторов и имен измерений и атрибутов ячеек совпадает с множеством идентификаторов и имен атрибутов типов экземпляров классов (2);
- любому измерению массива соответствует атрибут типа экземпляра класса, соответствующего этому массиву (3-6);

- любому атрибуту ячейки любого массива соответствует атрибут типа экземпляра класса, соответствующего этому массиву и типы атрибутов совпадают (7-10);
- атрибут ячейки массива, который может принимать неопределенные значения, соответствует определенному (*obligatory*) атрибуту типа (11-13);
- измерения соответствуют уникальным атрибутам типов (14-16);
- верхние (нижние) границы измерений равны верхним (нижним) границам соответствующих атрибутов типов (17-20);
- непустые ячейки массивов соответствуют объектам классов (21);
- для любой ячейки значения ее измерений и определенных атрибутов совпадают со значениями соответствующих атрибутов объекта, соответствующего ячейке (22-31).

Спецификации *ObjectDM* и *ArrayDM* вместе с инвариантом уточнения были загружены в инструментальное средство Atelier B [3]. 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 [4].

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 [17].

Дальнейшую работу можно разбить на три этапа:

- построение трансформации, реализующей изложенное отображение;
- расширение инструментальных средств поддержки предметных посредников для виртуальной интеграции SciDB-ресурсов: (i) расширение средств регистрации ресурсов в посреднике [19] трансформацией ЯОД ADM в каноническую модель; (ii) создание SciDB-адаптера - специального ПО, связывающего исполнительную среду посредников с SciDB-ресурсами (составной частью адаптера является 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.

Литература

[1] Abrial J.-R. The B-Book: Assigning Programs to Meanings. Cambridge: Cambridge University Press, 1996.

[2] Astronomy in ArrayDB. http://trac.scidb.org/raw-attachment/wiki/UseCases/Astronomy%20in%20ArrayDB.pdf

[3] Atelier B, the industrial tool to efficiently deploy the B Method. http://www.atelierb.eu/index-en.php

[4] 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

[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] J. Becla, K.-T. Lim. Report form the First Workshop on Extremely Large Databases. Data Science Journal, Volume 7, 2008.

[7] 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

[8] 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.

[9] 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.

[10] Kalinichenko L.A., Stupnikov S.A. Heterogeneous information model unification as a pre-requisite to resource schema mapping // A. D'Atri and D. Saccà (eds.), Information Systems: People, Organizations, Institutions, and Technologies (Proc. of the V Conference of the Italian Chapter of Association for Information Systems itAIS). – Berlin-Heidelberg: Springer Physica Verlag, 2010. – P. 373-380.

[11] 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.

[12] Martin L. Kersten, Ying Zhang, Milena Ivanova, Niels Nes: SciQL, a query language for science applications. EDBT/ICDT Array Databases Workshop 2011:1-12

[13] Large Synoptic Survey Telescope. http://www.lsst.org

[14] Leonid Libkin, Rona Machlin, Limsoon Wong. A Query Language for Multidimensional Arrays: Design, Implementation, and Optimization Techniques. SIGMOD, 1996. – P.228-239.

[15] Overview of SciDB: Large Scale Array Storage, Processing and Analysis. The SciDB Development team. SIGMOD, 2010.

[16] Torben Bach Pedersen, Christian S. Jensen. Multidimensional Database Technology. IEEE Computer, vol. 34, no. 12, pp. 40-46, 2001.

[17] SciDB User's Guide. Version 12.3. – 2012. http://www.scidb.org/

[18] P. Vassiliadis and T. K. Sellis. A Survey of Logical Models for OLAP Databases. SIGMOD Record, 28(4):64–69, 1999.

[19] 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.

[20] 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.

Unification of an Array Data Model for the Integration of Heterogeneous Information Resources

Sergey Stupnikov

In the paper a mapping of an array-based data model – the SciDB DBMS model - into a canonical model used for virtual or materialized database integration is presented. To be precise, a mapping into an object model - the SYNTHESIS language is considered. The SYNTHESIS language is used as the canonical model in the subject mediator technology.

A method for proving of preserving the information and the semantics of operations by the mapping is illustrated. The method uses a formal specification language.

An aim of the work is developing of a sound theoretical basis for the integration of array-based resources.