Вестник УГАТУ
Уфа : УГАТУ, 2012 Т. 16, № 6 (51). С. 167–176

УДК 004.65

А. С. Гусаренко, В. В. Миронов

ДИНАМИЧЕСКИЕ DOM-ОБЪЕКТЫ В СИТУАЦИОННО-ОРИЕНТИРОВАННЫХ БАЗАХ ДАННЫХ: ЛИНГВИСТИЧЕСКОЕ И АЛГОРИТМИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ИСТОЧНИКОВ ДАННЫХ

В рамках ранее разработанной концепции динамических DOM-объектов в составе динамической модели ситуационно-ориентированной базы данных предлагаетсялингвистическое обеспечение, предусматривающее графические и текстовые средства спецификации источников данных для DOM-объектов. Описывается рекурсивный алгоритм интерпретации, учитывающий DOM-элементы, ассоциированные с состояниями динамической модели, который обеспечивает заполнение DOM-объекта XML-контентом из специфицированных источников, когда родительское состояние становится текущим. Рассматривается реализация предложенных лингвистических и алгоритмических средств на платформе PHP. *Веб-приложение*; *база данных*; *динамическая модель*; *NoSQL*; *XML*; *DOM*; *PHP*

Современные веб-приложения, предоставляющие пользователям динамический контент на основе информации, хранящейся на веб-серверах, обусловливают необходимость поиска новых подходов к организации баз данных. Как альтернатива традиционным (реляционным) базам данных активно развиваются XML-ориентированные базы данных, в том числе в рамках движения NoSQL [1].

В работах [2–6] была предложена и получила развитие концепция ситуационно-ориентированных баз данных (СОБД), предназначенных для использования в динамических веб-приложениях. В СОБД данные в формате XML [7] ассоциированы с состояниями встроенной динамической модели. Для динамической модели отслеживаются ее текущие состояния, а данные обрабатываются в контексте состояний.

В данной статье рассматриваются вопросы автоматического формирования XML-данных, ассоциированных с состояниями динамической модели, в лингвистическом и алгоритмическом планах.

ВВОДНЫЕ ЗАМЕЧАНИЯ

Данная статья является продолжением работы [8], в которой была разработана концепция динамических DOM-объектов, автоматически создаваемых и заполняемых XML-контентом в процессе интерпретации динамической модели СОБД. В соответствии с этой концепцией DOM-объекты привязываются к состояниям динамической модели; DOM-объекты создаются, загружаются, используются для обработки XML-содержимого, когда соответствующие состояния динамической модели становятся текущими. В отличие от известного подхода, где эта функциональность достигается за счет программирования соответствующих функций в подпрограммах-акциях, ассоциированных с состояниями динамической модели, здесь:

- у элементов-состояний динамической модели в качестве дочерних предусматриваются DOM-элементы, у которых, в свою очередь, дочерние элементы-источники задают загружаемые XML-данные, а дочерние элементы-приемники – сохраняемое XML-содержимое;
- в ходе интерпретации динамической модели интерпретатором выполняется автоматическое создание DOM-объектов для текущих состояний модели и загрузка XML-данных с возможным преобразованием, а также автоматическое удаление DOM-объектов при смене текущих состояний.

В целом концепция направлена на то, чтобы при разработке динамической модели позволять в декларативной форме гибко специфицировать XML-данные, требуемые в тех или иных ситуациях, и способы их получения из различных источников, избавляя от программирования соответствующей функциональности. В данной статье рассматриваются вопросы реализации этой концепции.

Ключевые идеи. На рис. 1 иллюстрируется процесс формирования XML-содержимого DOM-объекта на основе информации из источников данных. Каждый элемент-источник ссылается на определенный XML-документ, *документ источника*, хранящийся в памяти ассоциированных данных (ADM).

Контактная информация: +7 (347)272-89-81.

Работа поддержана грантом РФФИ 10-07-00167-а.

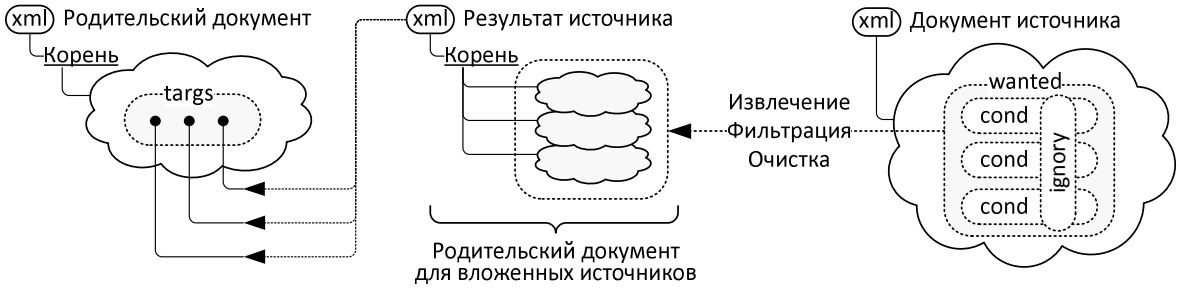

Рис. 1. Формирование XML-содержимого DOM-объекта на основе информации из источников данных

Он используется источником как основа для формирования XML-данных – *результата источника*, предназначенного для загрузки в DOM-объект. Результат источника помещается в *родительский документ*, соответствующий XML-содержимому элемента, родительского по отношению к элементу-источнику.

Источники могут быть вложенными друг в друга. Если элемент-источник непосредственно вложен в DOM-элемент, то его родительским документом является XML-содержимое DOM-объекта, а если элемент-источник сам вложен в другой элемент-источник, то его родительским документом является результат родительского источника.

В простейшем случае результат источника может представлять собой XML-содержимое документа источника. В более сложных ситуациях может потребоваться извлекать из документа источника только определенные, *искомые* XML-элементы (wanted), отфильтровать их в соответствии с заданными *условиями* (cond), очистить их внутреннее содержимое от *игнорируемых* элементов (ignory). В результате искомые XML-элементы могут представлять XML-фрагмент, не имеющий корневого элемента, что может потребовать заключить их внутрьXML-элемента – *корня* (root).

В простейшем случае результат источника помещается в корневой элемент родительского документа в качестве дочернего элемента. В более сложной ситуации может потребоваться отыскать в родительском документе множество *целевых* элементов (targs) и продублировать результат источника внутри каждого из них.

XML-технологии предусматривают два основных подхода к обработке документов:

- **потоковая** обработка – XML-документ обрабатывается потоком по мере чтения его узлов из файла. Документ при этом размещается в оперативной памяти не целиком, а частями. Это позволяет обрабатывать очень большие объемы данных в однонаправленном режиме. Инструменты потоковой обработки: XMLReader и XMLWriter;
- **кэшированная** обработка – перед обработкой XML-документ целиком загружается в структурированном виде (в виде дерева) в оперативную память компьютера. Это дает возможность гибкой обработки документов ограниченного объема. Инструменты кэшированной обработки – технология DOM.

Оба этих подхода могут применяться при обработке узлов в документе источника данных.

Дальнейшее изложение. В процессе реализации динамических DOM-объектов прорабатывалось *лингвистическое обеспечение*, обсуждаемое ниже и включающее языковые средства спецификации источников данных для загрузки, создаваемых DOM-объектов. Такие средства предусматривают возможность задания источников данных как на диаграмме динамической модели, в графическом виде, ориентированном на восприятие человеком, так и в текстовом виде, предназначенном для восприятия интерпретатором.

Исходя из того, что рутина подготовки и загрузки динамических DOM-объектов возлагается на интерпретатор, разработано соответствующее алгоритмическое обеспечение, обсуждаемое ниже и включающее алгоритмы обработки тех элементов динамической модели, которые специфицируют DOM-объекты и источники данных для них. Учитывалось, что необходимые DOM-объекты в модели задаются с помощью DOM-элементов, а элементы – источники данных могут быть вложенными (образовывать иерархию), это обусловило рекурсивный характер алгоритмов.

Проверка работоспособности алгоритмического обеспечения осуществлена путем программирования разработанных алгоритмов на базе языка серверных сценариев PHP с последующим тестированием на примерах.

СРЕДСТВА СПЕЦИФИКАЦИИ ИСТОЧНИКОВ ДАННЫХ

Динамическая модель СОБД задается с помощью формального языка HSM (Hierarchical Situational Models) [7] в виде графической диаграммы, имеющей эквивалентное XML-представление. Динамические DOM-объекты задаются в составе модели с помощью DOM-элементов и вложенных в них элементов – источников и приемников данных. Синтаксис этих элементов поясняется с помощью синтаксических диаграмм, на которых используются следующие обозначения: метасимвол >> означает «по определению»; метасимвол OR – выбор варианта (должен быть выбран один из вариантов); выносные линии, прикрепленные к элементу снизу – атрибуты элемента или его дочерние элементы; темный кружок на выносной линии – обязательный однозначный атрибут; светлый кружок – необязательный однозначный атрибут; светлая стрелка-треугольник – необязательный многозначный атрибут или множественный дочерний элемент; курсивом обозначены значения подстановки.

DOM-элемент. На рис. 2 приведена синтаксическая диаграмма DOM-элемента. По определению этот элемент на графической диаграмме представляется символом, содержащим метку «dom», с атрибутами и вложенными элементами.

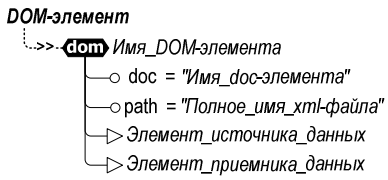

Рис. 2. Синтаксическая диаграмма DOM-элемента

DOM-элемент может содержать следующие необязательные атрибуты, указывающие исходный XML-документ для загрузки в порождаемый DOM-объект:

- doc – задает XML-документ в виде ссылки на doc-элемент (в динамической модели должен быть задан doc-элемент с соответствующим именем, содержащий информацию о нахождении нужного XML-файла);
- path – задает XML-документ путем явного указания пути к XML-файлу.

Если атрибуты отсутствуют, то после порождения DOM-объекта вместо загрузки исходного XML-документа создается корневой XML-элемент с именем DOM-элемента.

DOM-элемент может содержать несколько дочерних элементов – источников и приемников данных:

- элемент источника данных задает XML-результат, который будет помещен в порожденный DOM-объект (в корневой элемент или в указанные элементы предварительно загруженного исходного XML-документа) в качестве дочернего элемента (рассматривается ниже);
- элемент приемника данных задает XML-результат, который будет помещен в документ ADM или в поток выходных данных в качестве ответа на запрос (в данной статье не рассматривается).

Элемент источника данных представляет собой элемент динамической модели, размещаемый в DOM-элементе в качестве потомка (вложенного элемента). Он включает набор атрибутов, задающих свойства источника, иможет, в свою очередь, содержать дочерние элементы-источники (рекурсивное определение).

Синтаксическая диаграмма элемента-источника приведена на рис. 3. Элемент-источник задается символом с меткой «src», справа от которого указывается имя источника.

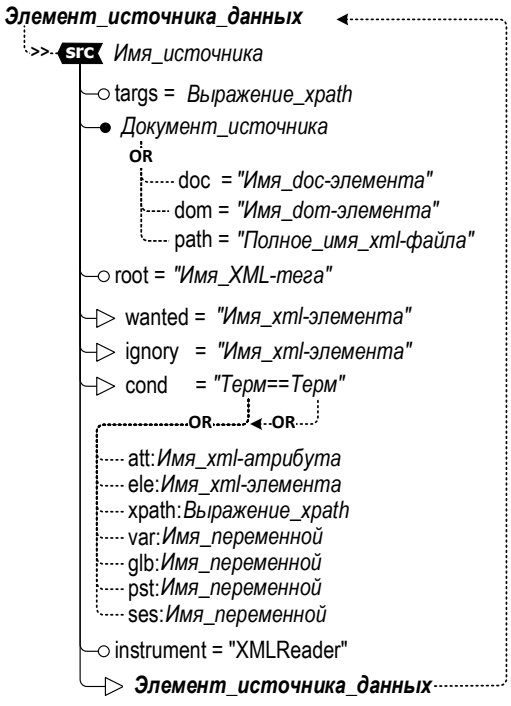

Рис. 3. Синтаксическая диаграмма элемента-источника

Атрибуты элемента-источника:

- targs – необязательный атрибут, указывающий в родительском XML-документе множество целевыхXML-элементов, к каждому

из которых будет прикреплен в качестве дочернего элемента XML-результат источника. При отсутствии атрибута XML-результат прикрепляется к корневому элементу родительского документа;

- *Документ_источника* – обязательная конструкция, задающая документ, из которого берется XML-содержимое. Эта конструкция представляет собой один из трех атрибутов:

– path – задает документ источника путем явного указания пути к XML-файлу;

– doc – задает документ источника в виде ссылки на doc-элемент (в динамической модели должен быть задан doc-элемент с соответствующим именем, содержащий информацию о нахождении нужного XML-файла);

– dom – задает в качестве документа XML-содержимое DOM-объекта, который порожден DOM-элементом с указанным именем;

- root – необязательный атрибут, предписывающий дополнительно заключить XML-результат источника в теги с заданным именем (может быть удобен, когда из документа источника извлекается XML-фрагмент, не имеющий корневого элемента);
- wanted – необязательный многозначный атрибут, задающий список имен XML-элементов, отыскиваемых в документе источника для формирования результата (список значений, разделенных пробелами, как принято в XML). При отсутствии атрибута результат формируется из корневого XML-элемента документа источника;
- ignory – необязательный многозначный атрибут, задающий список имен XML-элементов в документе источника, которые следует игнорировать при формировании результата;
- cond – необязательный многозначный атрибут, задающий список условий, которым должен удовлетворять искомый XML-элемент документа источника (из списка в атрибуте wanted), чтобы попасть в результат. При отсутствии атрибута в результат попадают все найденные XML-элементы из списка wanted. Синтаксис задания условий обсуждается ниже;
- instrument – необязательный атрибут, предписывающий использовать для чтения документа источника XMLReader метод потокового (некэшируемого) доступа. При отсутствии атрибута по умолчанию применяется метод кэшируемого доступа, при котором документ источника целиком помещается в буферный DOM-объект.

Условия фильтрации. Для управления процессом фильтрации XML-элементов источника данных алгоритм предусмотрен многозначный атрибут cond. Он содержит список разделенных пробелами условий в виде

терм == терм,

где левый терм задает проверяемое значение в искомом XML-элементе источника, а правый – в целевом XML-элементе родительского документа. Каждый терм содержит префикс и основу, разделенные двоеточием. Префикс указывает интерпретатору, каким способом отбирать значения при проверке соответствующего условия. Предусмотрены следующие префиксы:

- att: – проверяется значение атрибута XML-элемента, основа задает имя атрибута;
- ele: – проверяется значение дочернего XML-элемента, основа задает имя элемента;
- xpath: – проверяется значение, адресованное XPath-выражением, основа задает выражение;
- var: – проверяется значение переменной динамической модели, значение которой хранится в памяти текущего состояния (CSM), основа задает имя переменной;
- glb: – проверяется значение глобальной переменной среды интерпретации, основа задает имя глобальной переменной;
- pst: – проверяется значение Post-переменной из клиентского запроса, основа задает имя Post-переменной;
- ses: – проверяется значение сессионной (сеансовой) переменной среды интерпретации, основа задает имя сессионной переменной.

Вложенные источники данных. Каждый элемент-источник, в свою очередь, может содержать один или несколько внутренних элементов-источников, формирующих дочерние XML-элементы в составе результата источника. Глубина вложенности не ограничена.

Текстовая форма задания DOM-объекта. Графическое представление динамической модели (и в частности, DOM-объектов) в виде диаграммы используется как средство наглядного проектирования. Для использования в информационной системе на основе диаграммы строится эквивалентное текстовое представление в виде XML-кода.

Фрагмент динамической модели, содержащий DOM-элемент, в текстовом виде имеет следующую структуру:

```
<dom:Имя_DOM-элемента
    doc="Имя_doc-элемента">
    <src:Имя_источника
        doc="Имя_doc-элемента" … >
        …
        <src:Имя_источника
            doc="Имя_doc-элемента" … >
            …
        </src:Имя_источника>
    </src:Имя_источника>
</dom:Имя_DOM-элемента>,
```

где элементы и атрибуты записываются в соответствии с XML-синтаксисом.

АЛГОРИТМИЧЕСКОЕ ОБЕСПЕЧЕНИЕ

Под алгоритмическим обеспечением здесь понимается совокупность алгоритмов, в соответствии с которыми интерпретатор выполняет обработку фрагмента динамической модели, содержащего DOM-элемент. DOM-элементы располагаются в динамической модели как дочерние элементы элементов-состояний, поэтому их обработка выполняется в контексте обработки соответствующих родительских состояний.

Головной алгоритм интерпретации DOM-элемента

Блок-схема головного алгоритма обработки (интерпретации) DOM-элемента – DOMProc – приведена на рис. 4[1].

Вызванный алгоритм начинается (блок 1) с получения имени обрабатываемого DOM-элемента (блок 2) и проверки существования DOM-объекта с таким именем (блок 3). При отсутствии DOM-объект создается с корневым элементом (блок 4). Затем проверяется (блок 5) наличие у DOM-элемента атрибута doc или pathи в этом случае выполняется начальная загрузка в DOM-объект соответствующего документа (блок 6). Для DOM-объекта создается указатель на его корневой элемент (блок 7). Далее в цикле обрабатываются (блок 8) все дочерние элементы DOM-элемента (источники и приемники данных). Для каждого дочернего элемента проверяется, является ли он источником (блок 9), и в этом случае для него вызывается алгоритм обработки источника данных (блок 10), является ли он приемником (блок 11), и в этом случае для него вызывается алгоритм обработки приемника данных (блок 12). При вызове алгоритма обработки источника или приемника передаются указатели на корневой элемент DOM-объекта и на обрабатываемый элемент (источник / приемник). По окончании цикла (блок 8) алгоритм завершается и выполняет возврат в точку вызова (блок 13).

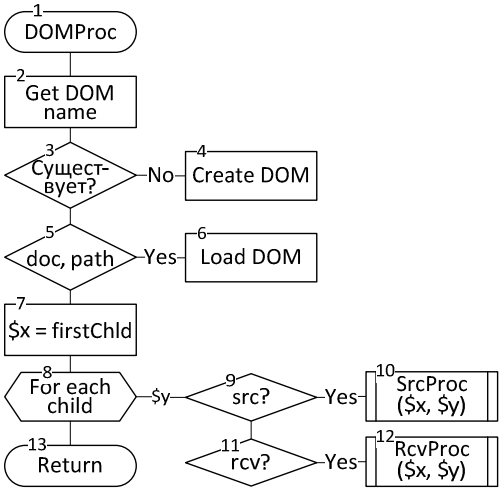

Рис. 4. Головной алгоритм интерпретации DOM-элементов

Алгоритм интерпретации элемента-источника

Блок-схема алгоритма обработки элемента-источника приведена на рис. 5.

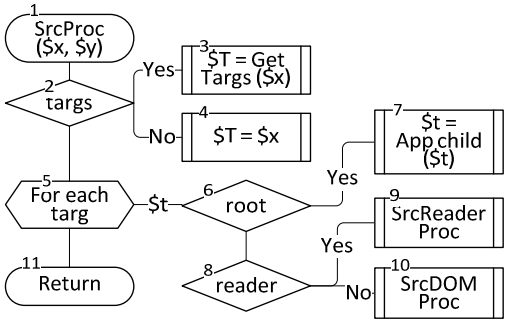

Рис. 5.Алгоритм интерпретации элемента-источника

Алгоритм (блок 1) начинает с проверки наличия атрибута targsy обрабатываемого элемента-источника (блок 2), чтобы определить способ формирования массива целевых элементов родительского документа. При наличии атрибута

[1]Используемый здесь и далее «иерархический» способ представления блок-схемы имеет отличия от традиционного. Вход в очередной блок того же уровня вложенности происходит по соединительной линии сверху или справа; выход всегда снизу; вход в первый дочерний (внутренний) блок из данного блока – справа; если внутренний блок последний в цепочке (не имеет соединительной линии снизу), то по его завершении происходит возврат в родительский блок.

массив целевых элементов формируется в результате поиска в родительском документе множества элементов с именами, заданными в атрибуте targs (блок 3). В противном случае множество целевых элементов включает единственный корневой элемент родительского документа (блок 4).

После этого выполняется циклическая обработка по каждому целевому элементу (блок 5). Проверяется наличие атрибута root у обрабатываемого элемента-источника (блок 6) и в этом случае формируется дочерний элемент у обрабатываемого целевого элемента, который далее будет использоваться вместо целевого элемента (блок 7). Проверяется наличие атрибута instrument = "XMLReader" (блок 8), задающий способ обработки узлов документа источника. В зависимости от этого вызывается алгоритм потоковой обработки – SrcReaderProc (блок 9) или кэшированной обработки – SrcDOMProc (блок 10).

По окончании цикла (блок 5) алгоритм завершается и выполняет возврат в вызывающий алгоритм (блок 11).

Алгоритм потоковой обработки элементов документа источника

Блок-схема алгоритма потоковой обработки элементов документа источника приведена на рис. 6.

Алгоритм (блок 1) начинает работу с инициализации инструмента потокового ввода XMLReader (блок 2), что обеспечивает открытие потока ввода для последовательного считывания тегов документа источника. Далее в цикле (блок 3) выполняется потоковое чтение узла за узлом из документа источника до тех пор, пока документ не закончатся. Для обработки отбираются узлы, являющиеся XML-элементами (блок 4). Проверяется, присутствует ли в элементе-источнике атрибут wanted (блок 5), и в этом случае проверяется, является ли обрабатываемый узел искомым XML-элементом документа источника (блоки 6–11).

Проверка начинается со сброса флага совпадения (блок 6). После этого в цикле (блок 7) перебираются имена из списка, заданного в атрибуте wanted, сравниваются с именем обрабатываемого узла (блок 8) и в случае совпадения устанавливается флаг (блок 9). По завершении цикла (блок 7) проверяется установка флага совпадения (блок 10) и, если флаг не установлен (совпадение не зафиксировано), принудительно завершается обработка данного узла (блок 11) и выполняется переход к обработке следующего (в блоке 3).

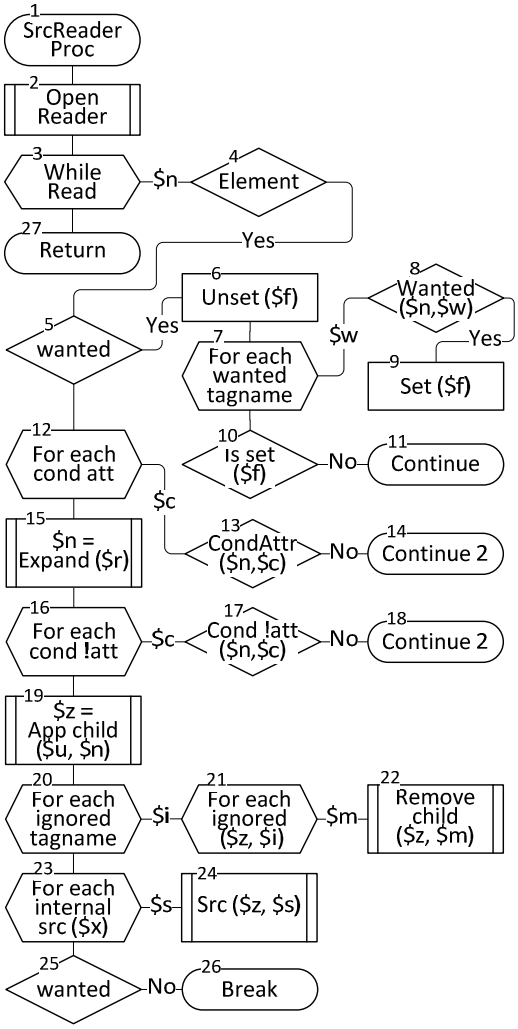

Рис. 6. Алгоритм потоковой обработки элементов документа источника

Если обрабатываемый узел признан искомым, он проверяется на выполнение условий фильтрации, которые заданы в атрибуте cond в виде условий на атрибуты обрабатываемого узла (блоки 12–14). Такие условия могут быть проверены непосредственно, без считывания следующих узлов документа из потока ввода. Для этого в цикле из атрибута cond извлекаются условия этого рода (блок 12) и проверяются для обрабатываемого узла (блок 13). При обнаружении невыполненного условия обработка данного узла принудительно завершается (блок 14) и выполняется переход к обработке следующего (в блоке 3).

Если обрабатываемый узел удовлетворяет условиям на атрибуты, он проверяется на выполнение других условий фильтрации, задан-

ных в атрибуте cond (блоки 15–18). Такие условия требуют обращения к другим узлам документа из потока ввода. Для этого обрабатываемый элемент (вместе со своим внутренним содержимым) извлекается в кэш (блок 15). Далее в цикле из атрибута cond извлекаются условия этого рода (блок 16) и проверяются для обрабатываемого узла, находящегося в кэше (блок 17). При обнаружении невыполненного условия обработка данного узла принудительно завершается (блок 18) и выполняется переход к обработке следующего (в блоке 3).

Если обрабатываемый узел успешно прошел все проверки, он присоединяется в качестве дочернего элемента к обрабатываемому целевому элементу родительского документа (блок 19).

Далее выполняется очистка присоединенного дочернего элемента от внутренних элементов, имена которых заданы в атрибуте ignory (блоки 20–22). Для этого имена из списка ignory просматриваются в цикле (блок 20) и для каждого из них выполняется циклический поиск внутренних элементов (блок 21) с удалением (блок 22).

После того как обрабатываемый узел документа источника проверен, присоединен к целевому элементу и очищен, для него рекурсивно обрабатываются внутренние источники данных (блоки 23, 24). Для этого выполняется цикл по элементам-источникам, дочерним по отношению к обрабатываемому (блок 23). Для каждого внутреннего источника рекурсивно вызывается алгоритм его обработки (блок 24, см. рис. 5), причем в качестве целевого элемента передается обрабатываемый узел.

На последнем шаге обработки узла вновь проверяется наличие атрибута wanted (блок 25) и в случае его отсутствия выполняется (блок 24) принудительное завершение цикла обработки документа. Это делается, чтобы в данной ситуации ограничиться обработкой корневого элемента документа источника.

По окончании основного цикла (блок 3) алгоритм завершается и выполняет возврат в вызывающий алгоритм (блок 27).

Алгоритм кэшированной обработки элементов документа источника

Блок-схема алгоритма кэшированной обработки элементов документа источника приведена на рис. 7. В целом она близка к блок-схеме потоковой обработки, хотя несколько проще ввиду больших возможностей кэшированной обработки XML-документов.

Алгоритм (блок 1) начинает работу с подготовки кэша в виде DOM-объекта с загруженным в него XML-документом источника (блок 2). При этом, если DOM-объект, на который ссылается элемент-источник, уже существует, он используется в качестве кэша; в противном случае создается новый DOM-объект, в который загружается документ источника.

Далее формируется массив ссылок (блок 3) на искомые элементы документа источника, имена которых заданы в атрибуте wanted элемента-источника (при отсутствии у элемента-источника атрибута wanted в массив заносится единственная ссылка на корневой элемент документа источника). Далее путем цикла по массиву ссылок (блок 4) проверяется каждый искомый элемент (блоки 5–13).

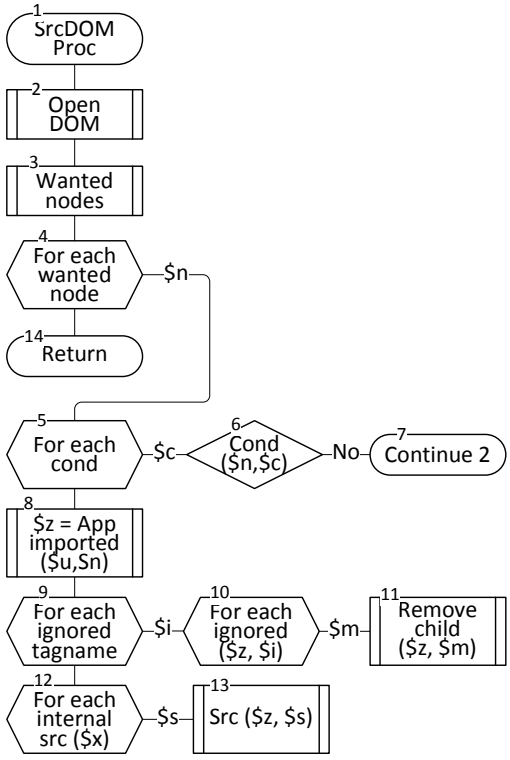

Рис. 7. Алгоритм кэшированной обработки элементов документа источника

Прежде всего проверяется выполнение условий фильтрации, которые заданы в атрибуте cond (блоки 5–7). Для этого условия в цикле извлекаются из атрибута cond (блок 5) и проверяются для обрабатываемого элемента (блок 6). При обнаружении невыполненного условия обработка элемента принудительно завершается (блок 7) и выполняется переход к обработке следующего (в блоке 4).

Если обрабатываемый узел успешно прошел проверки, он присоединяется в качестве дочернего элемента к обрабатываемому целевому элементу родительского документа (блок 8).

Далее выполняется очистка присоединенного дочернего элемента от внутренних элементов, имена которых заданы в атрибуте ignory (блоки 9–11). Для этого имена из списка ignory просматриваются в цикле (блок 9) и для каждого из них выполняется циклический поиск внутренних элементов (блок 10) с удалением (блок 11).

После того как обрабатываемый узел документа источника проверен, присоединен к целевому элементу и очищен, для него рекурсивно обрабатываются внутренние источники данных (блоки 12, 13). Для этого выполняется цикл по элементам-источникам, дочерним по отношению к обрабатываемому (блок 12). Для каждого внутреннего источника рекурсивно вызывается алгоритм его обработки (блок 13, см. рис. 5), причем в качестве целевого элемента передается обрабатываемый узел.

По окончании основного цикла (блок 4) алгоритм завершается и выполняет возврат в вызывающий алгоритм (блок 14).

РЕАЛИЗАЦИЯ НА ПЛАТФОРМЕ PHP

Реализация разработанного алгоритмического обеспечения проводилась в составе интерпретатора динамических моделей СОБД на платформе PHP. Разработана PHP-версия алгоритмов интерпретации DOM-элементов, элементов-источников и элементов-приемников в виде функций PHP, вызываемых друг из друга, а также из других функций интерпретатора. Использовались стандартные возможности потоковой и кэшированной XML-обработки, предоставляемые в PHP 5 (библиотеки libxml иlibxslt, поддерживающие расширенияDOM, XMLReader, XMLWriter).

Модульная структура программного обеспечения в плане функций динамических DOM-объектов поясняется на рис. 8.

Представленный ниже листинг иллюстрирует код динамических моделей, использовавшихся при тестировании разработанного программного обеспечения. Динамическая модель составлена в соответствии с диаграммой, использовавшейся на этапе разработки концепции динамических DOM-объектов [8, с. 168, рис. 2.12]. Она обслуживает веб-приложение, отображающее пользователю сведения о студентах и изучаемых ими предметах, и включает три уровня иерархии, 3 субмодели, 5 состояний. Для формирования выходных страниц используется 4 DOM-объекта (в листинге выделены жирным шрифтом), XML-контент которых динамически формируется из 3 исходных документов в зависимости от текущей ситуации.

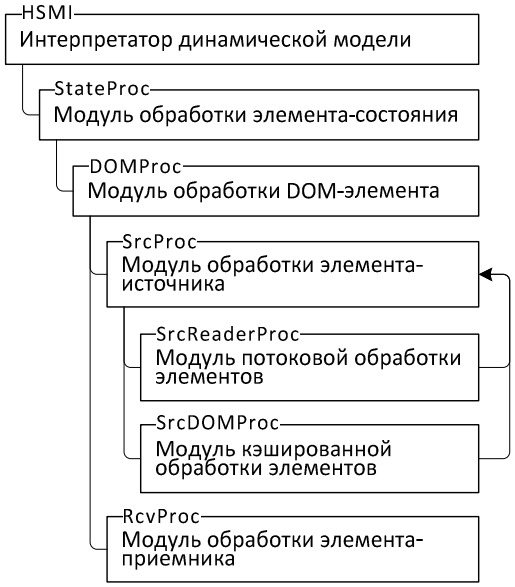

Рис. 8. Модульная структура программного обеспечения динамических DOM-объектов

Листинг. Тестовая динамическая модель с DOM-элементами и элементами-источниками

```
<?xml version="1.0"encoding="UTF-8"?>
<sta:Студенты-Предметы  xmlns:sta = 'HSM' xmlns:jmp = 'HSM' xmlns:sub = 'HSM' xmlns:act = 'HSM' xmlns:mnu = 'HSM'
    xmlns:dom = 'HSM' xmlns:btn = 'HSM' xmlns:use = 'HSM' xmlns:inp = 'HSM' xmlns:div = 'HSM' xmlns:var = 'HSM'
    xmlns:txt = 'HSM' xmlns:sec = 'HSM' xmlns:src = 'HSM' xmlns:rcv = 'HSM' xmlns:doc = 'HSM' xmlns:att = 'HSM'>

    <doc:СписокСтудентов path="XML/stud.xml" />                    <!--  Используемые XML-документы-->
    <doc:СписокПредметов path="XML/predm.xml" />
    <doc:Успеваемость path="XML/sdacha.xml" />
    <doc:СписСтудентов path="XSL/stud.xsl" />
    <doc:УспеваемостьСтудента path="XSL/uspevstud.xsl" />
    <doc:СписПредметов path="XSL/predm.xsl" />
    <doc:УспеваемостьПоПредмету path="XSL/uspevpredm.xsl" />

    <div:Студенты-Предметы section = "sec:СтудентыПредметы"/>      <!--  Погружение в субмодель Студенты-Предметы-->
```

```
    <sec:СтудентыПредметы>
<sub:Студенты-Предметы>                                  <!-- Субмодель Студенты-Предметы -->
    <sta:Студенты>                                       <!-- Состояние Студенты -->
        <act:Студенты pass="2">                          <!-- Акция -->
            <rcv:echomethod="label" value="Студенты" />
        </act:Студенты>
        <jmp:Предметы>                                   <!-- Переход по нажатию кнопки -->
            <btn:Предметы caption="К предметам" />
        </jmp:Предметы>
        <div:Студенты section = "sec:СтудентыПредметы"/> <!-- Погружение в субмодель Студенты-->
    </sta:Студенты>
    <sta:Предметы>                                       <!-- Состояние Предметы -->
        <act:Предметы pass="2">                          <!-- Акция  -->
            <rcv:echomethod="label" value="Предметы" />
        </act:Предметы>
        <jmp:Студенты>                                   <!-- Переход по нажатию кнопки -->
            <btn:Студенты caption="К студентам" />
        </jmp:Студенты>
        <div:Предметы section = "sec:СтудентыПредметы"/> <!-- Погружение в субмодель Предметы-->
    </sta:Предметы>
</sub:Студенты-Предметы>

<sub:Студенты>                                           <!-- Субмодель Студенты -->
    <sta:СписокСтудентов>                                <!-- Состояние СписокСтудентов -->
        <dom:СписокСтудентов>                            <!-- DOM-элемент СписокСтудентов -->
            <src:СписокСтудентов doc="СписокСтудентов" instrument="XMLReader"/>
            <rcv:echopass="2" method="xslt" styleSheet="doc:СписСтудентов"/>
        </dom:СписокСтудентов>
        <jmp:ВыбранСтудент targs="ВыбранСтудент">        <!-- Переход по нажатию кнопки -->
            <btn:ВыбратьСтудента caption="Выбрать студента" />
        </jmp:ВыбранСтудент>
     </sta:СписокСтудентов>

    <sta:ВыбранСтудент>                                  <!-- Состояние ВыбранСтудент -->
        <var:кодСтcash="global">                         <!-- Переменная в памяти текущего состояния-->
            <src:кодСт method="post"/>
        </var:кодСт>
        <dom:УспеваемостьСтудента>                       <!-- DOM-элемент УспеваемостьСтудента -->
            <src:ВыбранныйСтудент doc="СписокСтудентов" wantedElem="студент" cond="att:код==var:кодСт" instrument="XMLReader">
                <src:СдачиСтудента doc="Успеваемость" wantedElem="сдача" cond="att:кодСт==att:код" instrument="XMLReader">
                    <src:СданныйПредмет doc="СписокПредметов" wantedElem="предмет" cond="att:код==att:кодПр"/>
                </src:СдачиСтудента>
            </src:ВыбранныйСтудент>
            <rcv:echo method="xslt" styleSheet="doc:УспеваемостьСтудента" />
        </dom:УспеваемостьСтудента>
        <jmp:СписокСтудентов targs="СписокСтудентов">    <!-- Переход по нажатию кнопки -->
            <btn:КСпискуСтудентов caption="К списку студентов"/>
        </jmp:СписокСтудентов>
    </sta:ВыбранСтудент>
</sub:Студенты>

<sub:Предметы>                                           <!-- Субмодель Предметы -->
    <sta:СписокПредметов>                                <!-- Состояние СписокПредметов -->
        <dom:СписокПредметов>                            <!-- DOM-элемент СписокПредметов -->
            <src:СписокПредметов doc="СписокПредметов" ignory="цикл спец"/>
            <rcv:echo method="xslt" styleSheet="doc:СписПредметов" />
        </dom:СписокПредметов>
        <jmp:ВыбранПредмет targs="ВыбранПредмет">                <!-- Переход по нажатию кнопки -->
            <btn:ВыбратьПредмет caption="Выбрать предмет" />
        </jmp:ВыбранПредмет>
    </sta:СписокПредметов>

    <sta:ВыбранПредмет>                                  <!-- Состояние ВыбранПредмет -->
        <var:кодПр>                                      <!-- Переменная в памяти текущего состояния-->
            <src:кодПр method="post" />
        </var:кодПр>
        <dom:УспеваемостьПоПредмету>                     <!-- DOM-элемент УспеваемостьПоПредмету  -->
            <src:ВыбранныйПредметdoc="СписокПредметов" wantedElem="предмет" cond="att:код==var:кодПр">
                <src:СдачиПоПредметуdoc="Успеваемость" wantedElem="сдача" cond="att:кодПр==xpath:@код">
                    <src:СдавшийСтудентdoc="СписокСтудентов" wantedElem="студент" cond="att:код==att:кодСт"/>
                </src:СдачиПоПредмету>
```

```
            </src:ВыбранныйПредмет>
            <rcv:echomethod="xslt" styleSheet="doc:УспеваемостьПоПредмету" />
        </dom:УспеваемостьПоПредмету>
        <jmp:СписокПредметов targs="СписокПредметов">          <!-- Переход по нажатию кнопки -->
            <btn:КСпискуПредметов caption="К списку предметов"/>
        </jmp:СписокПредметов>
    </sta:ВыбранПредмет>

  </sub:Предметы>
  </sec:СтудентыПредметы>
</sta:Студенты-Предметы>
```

ЗАКЛЮЧЕНИЕ

В данной статье представлены результаты разработки лингвистического и алгоритмического обеспечения источников данных в составе динамических DOM-объектов, задаваемых в динамических моделях ситуационно-ориентированных баз данных.

Разработанное лингвистическое обеспечение в форме синтаксических диаграмм DOM-элементов и вложенных элементов-источников в составе элементов-состояний динамической модели *отличается* наличием атрибутов для задания условий отбора и фильтрации элементов документа-источника, присоединяемых к элементам родительского документа в ходе загрузки в DOM-объект.

Разработанное алгоритмическое обеспечение в форме блок-схем алгоритмов интерпретации DOM-элементов и элементов-источников *отличается* тем, что в зависимости от спецификации динамической модели предусматривается два режима обработки узлов документов источников данных – потоковый и кэшированный.

Алгоритмическое обеспечение реализовано в виде PHP-модулей в составе интерпретатора динамических моделей ситуационно-ориентированной базы данных. Тестовые примеры динамических моделей для веб-приложения продемонстрировали работоспособность предлагаемого подхода к организации динамических DOM-объектов.

Дальнейшие исследования предполагается продолжить в направлении исследования показателей производительности, прежде всего, времени заполнения динамических DOM-объектов XML-контентом источников данных.

СПИСОК ЛИТЕРАТУРЫ

1. **Strauch C., Kriha W.** NoSQL databases. URL: http://www.christof-strauch.de/nosqldbs.pdf (дата обращения 07.11.2012).

2. **Миронов В. В., Юсупова Н. И., Шакирова Г. Р.** Ситуационно-ориентированные базы данных: концепция, архитектура, XML-реализация // Вестник УГАТУ. 2011. Т. 14, № 2 (37). С. 233–244.

3. **Миронов В. В., Юсупова Н. И., Шакирова Г. Р.** Ситуационно-ориентированные базы данных: внешние представления на основе XSL // Там же. 2011 Т. 14, № 4 (39). С. 200–209.

4. **Миронов В. В., Маликова К. Э.** Интернет-приложения на основе встроенных динамических моделей: идея, концепция, безопасность // Там же. 2009 Т. 13, № 2 (35). С. 167–179.

5. **Миронов В. В., Маликова К. Э.** Интернет-приложения на основе встроенных динамических моделей: архитектура, структура данных, интерпретация // Там же. 2010. Т. 14, № 1 (36). С. 154–163.

6. **Миронов В. В., Маликова К. Э.** Интернет-приложения на основе встроенных динамических моделей: элементы управления пользовательского интерфейса // Там же. 2011. Т. 14, № 5 (40). С. 170–175.

7. **Миронов В. В., Юсупова Н. И., Шакирова Г. Р.** Иерархические модели данных: концепции и реализация на основе XML. М.: Машиностроение, 2011. 453 с.

8. **Миронов В. В., ГусаренкоА. С.** Ситуационно-ориентированные базы данных: концепция управления XML-данными на основе динамических DOM-объектов // Вестник УГАТУ. 2012. Т. 16, № 3 (48). С. 159–172.

ОБ АВТОРАХ

Миронов Валерий Викторович, проф. каф. автоматизир. систем упр-я. Дипл. радиофизик (Воронежск. гос. ун-т, 1975). Д-р техн. наук по упр. в техн. системах (УГАТУ, 1995). Иссл. в обл. иерархических моделей и ситуационного управления.

Гусаренко Артем Сергеевич, аспирант той же каф. Дипл. информатик-экономист (УГАТУ, 2010). Готовит дисс. о динамических DOM-объектах в ситуационно-ориентированных базах данных.