Язык модификации данных формата XML функциональными методами

Д.А. Лизоркин
lizorkin@ispras.ru

Аннотация: Задача внесения модификаций в данные формата XML является актуальной по природе слабоструктурированных данных. До настоящего времени не был разработан такой язык модификации XML-данных, который бы выступал в роли общепризнанного мирового стандарта.
В данной статье предлагается язык модификации XML-данных, в основу которого были положены функциональные методы программирования и язык функционального программирования Scheme. Функции, являющиеся в языках функционального программирования объектами первого класса, используются в предлагаемом языке модификации XML-данных в роли обработчиков для операций модификации, что позволяет достичь выразительной мощности и расширяемости набора операций модификации при сохранении синтаксической простоты языка.
Производится анализ алгоритмической сложности выполнения операций предлагаемого языка и рассматриваются детали его реализации функциональными методами.

1. Введение

Популярность языка разметки XML в качестве формата для хранения, передачи и обработки слабоструктурированных данных делает актуальной для практических приложений потребность в языке модификации XML-документов. Несмотря на актуальность данной задачи, доминирующее внимание консорциума W3C, которому принадлежит авторство большинства спецификаций платформы XML [1], в предыдущие годы было направлено на язык запросов из XML-документов [2], и работа над языком модификации XML-данных была начата в Консорциуме сравнительно недавно [3] и в настоящий момент находится в черновом статусе.

В настоящей статье предлагается язык модификации XML-документов, основанный на подходе к обработке XML-данных функциональными методами, в частности, с помощью языка функционального программирования Scheme. Благодаря близкому соответствию иерархической структурой XML-документа вложенным спискам языка функционального

программирования Scheme и свойству функций Scheme быть объектами первого класса обеспечивается выразительная гибкость предлагаемого языка модификаций и минимизируется проблема потери соответствия [4] по сравнению с другими существующими языками модификации XML-документов.

Подход к обработке XML-данных функциональными методами базируется на формате SXML — представлении XML-документов в виде вложенных списков, текстуально записываемых при помощи S-выражений. Языки SXML и XML могут рассматриваться как два синтаксически различных представления Информационного Пространства XML [5]. Для демонстрации соответствия между вложенными тегами XML и вложенными списками SXML, на рис. 1 приведен пример простого XML-документа и его представления на SXML. Каждая из информационных единиц Информационного Пространства XML представляется в виде S-выражения, первым членом которого является либо имя информационной единицы (для типов *элемент* и *атрибут*), либо служебное имя, предусмотренное для информационной единицы данного типа в грамматике SXML [6]. Заметим, что спецификация SXML обеспечивает поддержку пространств имен XML, и существует парсер, позволяющий для заданного XML-документа сконструировать его представление на SXML [7].

```
<?xml version='1.0'?>
<doc>
  <tag attr1="value1"
attr2="value2">
    <nested>Text
node</nested>
  </tag>
  <empty/>
</doc>
```

```
(*TOP* (*PI* xml
"version='1.0'")
 (doc
   (tag (@ (attr1 "value1")
(attr2 "value2"))
     (nested "Text node")
   )
   (empty)
 ))
```

*Рис. 1: XML-документ (слева) и его представление в SXML*Универсальность представления на SXML узлов всех типов обеспечивает простоту обработки документов формата SXML, а функции Scheme, обладающие свойством объектов первого класса, позволяют прозрачным образом интегрировать конструкции предлагаемого в настоящей статье языка модификации с пользовательскими действиями приложения.

Необходимо сразу же отметить, что в функциональной парадигме программирования под модификацией данных понимается нечто отличное от модификации в контексте процедурной и объектно-ориентированной парадигм программирования. Чисто функциональный стиль подразумевает отсутствие в программе побочных эффектов, что исключает такие широко принятые в процедурном и объектно-ориентированном подходе операции как присваивание новых значений глобальным переменным или перестановка

указателей [9]. Модификация в функциональном стиле является операцией, не разрушающей исходные данные, и модификация исходного документа осуществляется за счет конструирования нового документа.

Благодаря отсутствию в программе побочных эффектов, для конструирование модифицированного документа не требуется глубокое копирование исходного документа [8], но, как более подробно обсуждается в разделе 6, неизменные в результате модификации части документа используются в виде единой физической копии для его исходной и результирующей версий [9]. Приложение само решает, каким образом обрабатывать исходный и модифицированный документы. Например, приложение может использовать исходный документ лишь в локальном блоке своего кода, и тогда части, специфичные для исходного документа, будут освобождены автоматическим сборщиком мусора при выходе программы из данного локального блока [10].

Вопрос обеспечения работы нескольких конкурентных приложений над общим документом вкратце затрагивается в разделе будущих исследований, и его детальное исследование находится за пределами данной статьи.

Дальнейшее содержание статьи организовано следующим образом. В разделе 2 дается обзор существующих языков модификации XML-данных и обсуждаются преимущества функционального подхода. В разделе 3 предлагается идея использование функций в качестве обработчиков для операций модификации, иллюстрируется выразительность данного подхода и показывается, как с помощью обработчиков могут быть выражены наиболее распространенные операции существующих языков модификации. В разделе 4 идея обработчиков расширяется возможностью согласованной обработки нескольких узлов документа, что позволяет естественно реализовать такие операции модификации, как перемещение поддеревьев. В разделе 5 вводится сокращенный синтаксис для наиболее употребительных конструкций предлагаемого языка модификаций. Раздел 6 посвящен деталям реализации предлагаемого языка модификаций функциональными методами программирования. Вопрос алгоритмической сложности вычисления запросов на модификацию предлагаемого языка рассматривается в разделе 7. Будущие исследования обсуждаются в разделе 8. Раздел 9 завершает статью.

2. Существующие языки модификации XML-данных

До настоящего времени какого-либо единого стандарта языка модификации XML-данных не существовало. Так, разработанный консорциумом W3C язык запросов к XML-данным XQuery [2] в своей текущей версии является языком, предназначенным лишь для выборки данных и не предоставляющим никаких средств для внесения модификаций в XML-документы. С другой стороны, ввиду большого распространения технологии XML и естественности задачи внесения модификации в XML-данные для практических приложений, имеется ряд разработанных промышленных и академических языков для модификации XML-данных.

При анализе существующих языков модификации XML-данных в первую очередь необходимо отметить работу [11], в которой предлагается детально проработанный язык модификации с описанной формальной семантикой, который является расширением XQuery. По аналогии с понятием запроса в языке XQuery в языке модификаций, предлагаемом в [11], вводится понятие *запроса на модификацию* (update query). Запрос на модификацию состоит из одной или нескольких *операций модификации* (update expression), где каждая операция модификации производит некоторое базовое действие над обрабатываемым XML-документом. В работе [11] вводятся следующие виды операций модификации:

- вставка нового узла в дерево документа;
- удаление узла;
- замена узла новым узлом;
- переименование узла.

Для выбора узлов XML-документа, подлежащих изменению данной операцией модификации, в [11] используется язык адресации структурных частей XML-документа XPath [12]. По семантике вычисления запроса на модификацию, операции модификации, входящие в запрос, независимы друг от друга: изменения, производимые в XML-документе одной операцией модификации, не учитываются при выборе узлов, подлежащих изменению другой операцией модификации.

Несмотря на то что набор введенных в [11] операций модификации позволяет выразить любое необходимое изменение XML-документа [13], в языке отсутствует возможность расширения функциональности этих операций или добавления пользовательских операций. В настоящей статье семантика операций модификации определяется с помощью *обработчика*, представляющего собой произвольную функцию с заданной сигнатурой, благодаря чему достигается полная выразительная мощность языка модификаций из [11] и обеспечивается возможность для приложения по определению собственных операций модификации, равноправных с базовыми операциями вставки, удаления, замены и переименования узлов.

Необходимо также отметить, что в языке модификаций, предложенном в [11], не предоставляется важной операция перемещения поддеревьев [13] и предлагается реализовывать ее, как комбинацию операций удаления и вставки узлов. Хотя операция перемещения действительно может быть реализована таким образом, в работе [11] не предоставляется никаких средств по обеспечению согласованности операций удаления и вставки при их комбинации с целью получения эффекта перемещения поддерева.

В настоящей статье разрабатывается механизм зависимой обработки совокупности из нескольких узлов, благодаря которому достигается возможность согласованной модификации узлов, расположенных в разных частях документа, и, в частном случае, функциональность операции

перемещения поддеревьев. Разработанный сокращенный синтаксис языка модификаций предоставляет приложению возможность записи операции перемещения поддеревьев в виде наглядного выражения.

Язык модификации XML-данных, предлагаемый в работе [14], также разработан как расширение к XQuery, и в нем используются богатые возможности XQuery по адресации частей обрабатываемого документа и конструированию новых узлов. Несмотря на всю мощность XQuery как языка запросов, при реализации на его основе языка модификаций авторам работы [14] потребовалось расширить язык XPath, являющийся составной частью XQuery, возможностью явной адресации значения конкретного атрибута. Проблема адресации отдельного значения атрибута с целью его модификации не возникает в SXPath — реализации XPath на языке функционального программирования Scheme, — поскольку SXPath позволяет обрабатывать элементы и атрибуты унифицированным образом ввиду их однородного способа представления на SXML [15].

В работе [14] вводится операция суб-модификации (sub-update), позволяющая производить поиск других узлов относительно целевого узла модификации и осуществлять для них рекурсивный вызов операции модификации. По своему назначению операция суб-модификации схожа с предлагаемым в настоящей работе механизмом зависимой обработки совокупности из нескольких узлов. При этом предлагаемый механизм обладает большей декларативностью по сравнению с операцией суб-модификации.

Более декларативным является и предлагаемый в настоящей работе способ задания обрабатываемых узлов, подлежащих модификации. Для сравнения, в языке модификации, разработанном в [14], итерация по обрабатываемым узлам полностью перекладывается на прикладного программиста, что приводит к необходимости использования чрезмерно большого числа итеративных конструкций (for-update) в запросах на модификацию.

Необходимо также отметить, что в работе [14] язык модификаций исследовался в контексте хранения XML-документов в реляционной базе данных, ввиду чего у авторов упомянутой работы возникли проблемы реализационного характера по поддержанию корректной контекстной позиции узлов при вставке нового содержимого между существующими данными [14].

В разрабатываемом группой XML:DB языке модификации XML-документов XUpdate [16] вводится набор операций, аналогичный уже рассмотренным, и для записи запросов на модификацию используется язык XML и специальное пространство имен. Язык XUpdate пока развит хуже по сравнению с рассмотренными выше работами [11] и [14], и до настоящего времени находится в черновом статусе разработки.

3. Операция модификации как обработчик узла

Модификация на предлагаемом в данной статье языке выражается с помощью декларативного запроса, который мы по аналогии с введенной в [11] терминологией будем называть *запросом на модификацию* (update query). В контексте функциональных методов программирования можно говорить о том, что запрос на модификацию является атомарным в том смысле, что в реализации, детали которой будут обсуждаться в разделе 6, модификация осуществляется за один проход по дереву модифицируемого документа.

Запрос на модификацию может состоять из одной или нескольких *операций модификации*, где операция модификации предназначена для выполнения некоторого базового действия над частью обрабатываемого документа. Каждая операция модификации адресуется к набору некоторых узлов в документе и выполняет модификацию каждого узла данного набора по отдельности.

Анализируя семантику предлагаемых в [11] операций модификации, можно заметить, что любая из этих операций может быть в обобщенном виде представлена как пара значений:

```
(выражение_XPath обработчик) ,     (1)
```

где `выражение_XPath` служит для выбора в обрабатываемом XML-документе набора узлов, подлежащих модификации, а `обработчик` специфицирует воздействие конкретной операции модификации на каждый из выбранных узлов. Обработчик может рассматриваться, как функция, которая получает на вход узел, подлежащий модификации, и возвращает результат воздействия конкретной операции модификации на данный узел.

Предложенное представление операции модификации в виде пары значений получает естественную реализацию в терминах языка функционального программирования Scheme, поскольку в Scheme функции обладают свойством объектов первого класса. Функции как объекты первого класса могут, в частности, передаваться в качестве аргументов другим функциям, что позволяет нам рассматривать обработчик как черный ящик и, таким образом, абстрагироваться от внутренней реализации конкретного обработчика в дизайне языка модификаций.

Рассматривая операцию модификации XML-данных в контексте языка функционального программирования Scheme и формата SXML, определим обработчик как функцию, имеющую следующую сигнатуру:

```
(lambda (node) ...) ,     (2)
```

где `node` — это узел, подлежащий обработке с помощью данной операции модификации. Поскольку результат операции модификации естественным образом зависит от узла, подлежащего обработке, функция, введенная для представления обработчика, принимает данный узел в качестве аргумента. В

соответствии с тем значением, которое вернет обработчик, будет реализовываться та или иная операция модификации.

Будем считать допустимыми возвращаемыми значениями обработчика либо единственный узел, либо набор узлов. В том случае, когда обработчик возвращает единственный узел, этот новый узел будет замещать собой в документе обрабатываемый узел (который был фактическим параметром для данного вызова обработчика). Если обработчик возвращает *набор* узлов, то этот набор узлов предполагается упорядоченным, и на место обрабатываемого узла в документе подставляются все узлы из полученного набора. В частности, если возвращаемый набор узлов пуст, то обрабатываемый узел удаляется из документа, и вместо него ничего не подставляется [17].

В соответствии с предложенным в данном разделе представлением операции модификации в виде пары значений, с помощью `выражения_XPath` будет выбираться набор узлов исходного документа, которые подлежат обработке в данной операции модификации. Каждый узел из выбранного набора рассматривается как обрабатываемый узел, и с ним, как с фактическим параметром, вызывается `обработчик`. Возвращаемый результат обработчика формирует новый узел или узлы, которые заменяют собой обрабатываемый узел. В частности, возвращаемый результат может включать в себя и сам обрабатываемый узел, что производит эффект добавления в дереве документа некоторых новых узлов к обрабатываемому узлу.

Несмотря на простоту рассмотренной семантики `обработчика` и сопоставленного с ним `выражения_XPath`, предложенное представление операции модификации в терминах, приближенных к языку функционального программирования Scheme, обладает достаточно широкими возможностями для выражения разнообразных модификаций документов формата SXML. Далее в данном разделе рассматривается реализация с помощью идеи обработчиков некоторых наиболее употребительных операций модификации, затем обсуждаются возможности обработчиков при выражении более сложных операций модификации.

3.1. Реализация некоторых операций модификации функциональными методами

В данном пункте показывается, как имеющиеся в работе [11] операции модификации получают естественное и лаконичное воплощение на языке функционального программирования Scheme при обработке документов в формате SXML.

3.1.1. Вставка нового узла в документ

В работах [11] и [14] предлагается 3 разновидности операции вставки нового узла в документ:

- *вставка после* (insert following), добавляющая новый узел сразу же после обрабатываемого узла таким образом, что они оба имеют общий родительский узел;
- *вставка перед* (insert preceding), добавляющая новый узел сразу же перед обрабатываемым узлом таким образом что они оба имеют общий родительский узел;
- *вставка внутрь* (insert into), добавляющая новый узел в качестве дочернего для обрабатываемого узла, последним по порядку документа (document order [12]).

В терминах SXML и функционального программирования эти операции вставки узла представляют собой простую комбинацию примитивов работы над списковыми структурами данных. Так, эффект “вставки перед” и “вставки после” достигается, когда обработчик формирует список, состоящий из добавляемого в документ нового узла и обрабатываемого узла. Соответственно, если в формируемом списке новый узел идет первым, то в дереве документа он будет добавлен *перед* обрабатываемым узлом (вставка перед), если вторым — то *после* обрабатываемого (вставка после).

С целью параметризации обработчиков новым узлом, подлежащим вставке, обработчики реализуются как возвращаемый результат функции, которая и осуществляет необходимую параметризацию. Данный подход возможен благодаря тому, что функции Scheme являются объектами первого класса. Реализация обработчиков для “вставки перед” (insert preceding) и “вставки после” (insert following) приведена ниже:

```
(define (insert-preceding new-node)
  (lambda (node)
    (list new-node node)))

(define (insert-following new-node)
  (lambda (node)
    (list node new-node)))
```

Заметим, что при параметризации каждой из записанных функций для нескольких разных значений нового узла `new-node` будет получено несколько разных обработчиков. Рассмотренный дизайн обработчиков по добавлению узлов в виде возвращаемых результатов функций удобен для приложения, поскольку подлежащий вставке новый узел `new-node`, как правило, известен на этапе формулирования операции модификации, тогда как обрабатываемый узел `node` становится известным только на этапе обработки дерева документа.

При определении семантики операции “вставки внутрь” следует особо выделять случай, когда обрабатываемый узел является текстовым узлом, т.е. по определению не может содержать узлов, вложенных в него. Реакция на

подобную ситуацию может различаться в зависимости от нужд конкретного приложения; ниже показывается реализация, которая оставляет обрабатываемый текстовый узел без изменения:

```
(define (insert-into new-node)
  (lambda (node)
    (if (not (pair? node))  ; текстовый
        node  ; оставляем без изменения
        (append node (list new-node)))))
```

В рассматриваемых далее в данном разделе примерах используются выражения модификаций, исходно предложенные в [11], и для них приводятся и обсуждаются аналоги на языке Scheme, записанные в терминах предлагаемой идеи обработчиков.

Пример 1 *Запрос, содержащий операцию модификации на "вставку перед", в терминах синтаксиса, разработанного в [*11*], записывается в виде:*

```
UPDATE
INSERT
  <warning>High Blood Pressure!</warning>
PRECEDING //blood_pressure[systolic>180]
```

В контексте предложенной в данной статье идеи использования обработчиков для выражения операций модификации, рассматриваемый запрос получает естественное эквивалентное воплощение на Scheme для обработки документов в форме SXML:

```
(sxml:modify
 `("//blood_pressure[systolic>180]"
   ,(insert-preceding
     '(warning "High Blood Pressure!"))))
```

Функция `sxml:modify` *реализует запрос на модификацию, который может состоять из одной или более операций модификации. Операция модификации получает естественную нотацию в виде списка, состоящего из двух членов: выражения XPath и функции, играющей роль обработчика. Использованные при записи операции модификации выражения квази-цитирования (quasiquote, сокращенно обозначаемое символом "`") и снятия цитирования (unquote, сокращенно обозначаемое символом ",") показывают, что первый член списка представляет собой константное выражение, а второй член должен быть вычислен.*

Результатом функции `sxml:modify` *является в свою очередь функция, которая и осуществляет обработку документа формата SXML.*

В полной аналогии с примером 1 реализуются примеры на "вставку после" и "вставку перед", и единственное отличие заключается в использовании соответствующего обработчика для каждой из этих операций.

3.1.2. Удаление узла

Операция удаления узла удаляет из документа обрабатываемый узел и все узлы, вложенные в него. Если рассматривать документ в виде дерева, то можно говорить о том, что в данной операции происходит удаление целого поддерева, корнем которого служит обрабатываемый узел.

Выше в данном разделе при обсуждении семантики различных возвращаемых значений обработчиков было замечено, что эффект удаления узла достигается в том случае, когда обработчик возвращает в качестве результата пустой набор узлов. В терминах SXML пустой набор узлов представляется пустым списком языка Scheme, и поэтому удаление узла обеспечивает такой обработчик, который игнорирует обрабатываемый узел и всегда возвращает пустой список:

```
(define delete
  (lambda (node) '()))
```

Необходимо отметить, что в отличие от рассмотренных в предыдущем пункте обработчиков по вставке узла, обработчик для удаления узла не требует параметризации.

Пример 2 *Запрос на удаление узлов на языке модификаций, выработанном в работе [*11*], записывается следующим образом:*

```
UPDATE
DELETE //blood_pressure[systolic>180]
```

С использованием обработчика для удаления узла, эквивалентный запрос на модификацию документов формата SXML имеет на Scheme следующий вид:

```
(sxml:modify
 `("//blood_pressure[systolic>180]"
   ,delete))
```

Заметим, что по сравнению с примером 1 в выражении языка Scheme поменялся только используемый обработчик. Благодаря гибкости предлагаемого подхода замена обработчика позволяет полностью изменить эффект производимой модификации.

3.1.3. Замена узла новым узлом

Операция замены узла замещает обрабатываемый узел и всё его содержимое новым узлом, который задается в качестве параметра операции. Если рассматривать документ в виде дерева, то можно говорить о том, что в данной операции происходит замещение новым поддеревом целого поддерева, корнем которого служит обрабатываемый узел.

Обработчик для замены узла новым узлом во многом напоминает рассмотренные ранее обработчики для вставки узла с тем упрощением, что

теперь обрабатываемый узел node полностью игнорируется, и возвращаемым значением обработчика является новый узел new-node:

```
(define (replace new-node)
  (lambda (node) new-node))
```

Пример 3 *Запрос на замену узла новым узлом на языке модификаций, разработанном в [11], имеет следующий вид:*

```
UPDATE
REPLACE //job[.="bit banger"]
WITH
 <profession>Comp. Scientist</profession>
```

С помощью предлагаемой в данной статье идеи обработчиков эквивалентный результат достигается на Scheme так:

```
(sxml:modify
 `("//job[.='bit banger']"
   ,(replace
     '(profession "Comp. Scientist"))))
```

3.1.4. Переименование узла

Операция переименования узла изменяет имя узла и оставляет неизменным его содержимое.

Операция переименования осмысленна для узлов, которые обладают именами, т.е. для элементов и атрибутов. При обработке XML-данных на языке Scheme элементы и атрибуты выражаются в виде S-выражений единообразным образом, и имя всегда является первым членом S-выражения, соответствующего данному элементу или атрибуту.

В случае, когда обрабатываемый узел не имеет имени (например, для текстового узла), в зависимости от семантики конкретного приложения может требоваться различная реакция обработчика. Рассмотрим такой вариант обработчика, который оставляет текстовый узел без изменения:

```
(define (rename new-name)
  (lambda (node)
    (if (pair? node)  ; именованный узел
        (cons new-name (cdr node))
        node)))
```

По аналогии с операциями вставки узла и замены узла новым узлом обработчик для переименования узла реализован как возвращаемый результат функции, которая осуществляет параметризацию обработчика новым именем new-name. Благодаря параметризации можно получить для нескольких разных значений нового имени new-name несколько разных обработчиков,

каждый из которых определяет переименование обрабатываемых узлов на соответствующее новое имя.

Приведенное тело обработчика может быть легко дополнено функциональностью, позволяющей для узла типа “инструкция обработки” переименовывать ее указание адреса (PI target) [19], что сделает семантику рассматриваемой операции переименования узла полностью совместимой с [11].

Пример 4 *Переименование узла в терминах языка модификаций [11] записывается в виде:*

```
UPDATE
RENAME //job[.='bit banger']
AS "profession"
```

С использованием предлагаемой идеи обработчиков, эквивалентный запрос на модификацию может быть записан на Scheme так:

```
(sxml:modify
 `("//job[.='bit banger']"
   ,(rename 'profession)))
```

Здесь `'profession` *имеет тип данных* символ*, и этот тип данных используется в SXML для представления имен элементов и атрибутов.*

3.2. Возможности языка Scheme по выражению операций модификации

Помимо рассмотренных в предыдущем пункте примеров конкретных обработчиков для наиболее употребительных операций модификации, этими операциями далеко не ограничивается возможности, которые предоставляет приложению использование предложенной идеи обработчиков. При определении обработчиков и, тем самым, операций модификации над документами формата SXML приложению становятся доступны выразительная мощность языка программирования общего назначения Scheme и большой набор стандартных функций для работы со списковыми структурами данных.

В частности, конструкторы различных типов узлов реализуются на Scheme конструкторами списков. Более того, наличие в языке Scheme выражений квази-цитирования (quasiquote) и снятия цитирования (unquote) позволяет компактным и наглядным образом комбинировать константные выражения и фрагменты вычисляемых выражений [18]. Аналогичные идеи используются в синтаксисе XSLT для комбинирования литеральных элементов результата [1] и исполняемых инструкций.

Тело обработчика может включать в себя вызовы библиотеки SXPath — реализации языка XPath на Scheme, — что позволяет приложению

адресоваться к структурным частям обрабатываемого SXML-документа в соответствии со взаимоотношением этих частей с обрабатываемым узлом как контекстным узлом XPath.

Примитивы `map` и `filter` языка Scheme обеспечивают приложение при работе с узлами SXML-документа возможностями итерирования и фильтрации последовательности узлов в терминах языка запросов к XML-документам XQuery. Арифметико-логические и условные выражения языка XQuery реализуются на Scheme соответствующими стандартными функциями.

Благодаря возможности использования в теле обработчика произвольного выражения языка программирования общего назначения Scheme и интегрированности Scheme с узлами SXML-документа на уровне списковых структур данных, предложенная в данной статье идея обработчиков обеспечивает приложение выразительным и мощным механизмом для определения операций модификации.

4. Зависимая обработка нескольких узлов

Хотя предложенные в предыдущем разделе обработчики узлов обеспечивают достаточно богатые возможности по выражению операций модификации документов, операции типа перемещения поддерева из одного места в документе на другое место обработчиками рассмотренного вида не покрываются. Это объясняется тем, что обработчик вида, рассмотренного в предыдущем разделе, является функцией только от одного обрабатываемого узла, тогда как операция перемещения затрагивает сразу два узла: исходное местонахождение поддерева и его желаемое новое положение. В обобщенном случае можно говорить о том, что обработчиками, рассмотренными в предыдущем разделе, не могут быть выражены такие операции модификации, которые включают в себя зависимую обработку сразу нескольких узлов, располагающихся в разнесенных друг от друга местах обрабатываемого дерева документа.

Позиционирование операции перемещения поддерева как одной из базовых операций редактирования древовидных структур данных [13] мотивирует дальнейшее расширение предлагаемой в данной статье идеи обработчиков возможностью осуществлять перемещения поддеревьев[1]. Необходимо отметить, что в работе [11] операция перемещения не обеспечивается, и вместо этого предлагается имитировать ее как независимое последовательное применения операций удаления и вставки, причем никаких средств согласованного использования этих двух операций не предоставляется.

[1] Название операции “перемещение поддерева” выбрано в соответствии с терминологией, введенной в работе [13]. В контексте предлагаемой в настоящей статье идеи обработчиков, корень поддерева, подлежащего перемещению, задается обрабатываемым узлом.

Предлагаемый в данной статье способ представления операции модификации в виде пары значений

```
(выражение_XPath обработчик)
```

по сути, есть не что иное, как способ сопоставления обработчиков и соответствующих обрабатываемых узлов. Поскольку в соответствии с семантикой сначала происходит сопоставление обработчиков с узлами, а затем за один проход по дереву документа осуществляется его модификация в соответствии с возвращаемыми значениями обработчиков, не имеет принципиального значения, является ли обработчик функцией только от одного узла или сразу от нескольких узлов. Теоретически, возможности функционального программирования и гибкость предлагаемого подхода не накладывают никаких принципиальных ограничений на количество узлов, которые могут быть обработаны вместе с помощью единого обработчика. С другой стороны, эксперименты с реализованными ранними прототипами показали, что функциональность, предоставляемая для зависимой обработки более чем двух разнесенных друг от друга по дереву документа узлов редко находит применение в практических задачах, но при этом значительно усложняет пользовательский интерфейс языка модификаций.

В качестве сбалансированного решения в данной статье предлагается дальнейшее расширение идеи обработчиков возможностью зависимой обработки не более чем двух разнесенных друг относительно друга по дереву документа узлов. Предлагаемое расширение предоставляет приложению гибкие и мощные средства для формулирования практических операции модификаций к документу (в частности, операцию перемещения поддеревьев) и сохраняет простоту пользовательского интерфейса. Дополнительно, как будет показано в разделе 7, предлагаемый способ обработки обладает хорошими априорными свойствами в плане алгоритмической сложности вычислений, проводимых при модификации документа.

Для обеспечения функциональности зависимой обработки двух узлов документа введем понятие *базового узла*.

4.1. Базовый узел

Базовым узлом для операции модификации назовем такой узел, относительно которого вычисляется `выражение_XPath` данной операции модификации. Поскольку при помощи выражения XPath в операции модификации выбираются обрабатываемые узлы, можно говорить о том, что базовый узел определяет те узлы, которые выбираются, и, соответственно, обрабатываются в данной операции модификации.

Как отмечалось в разделе 1, запрос на модификацию состоит, вообще говоря, из нескольких операций модификации, которые можно считать упорядоченными между собой. Считаем, что базовым узлом для первой по

порядку операции модификации в данном запросе на модификацию всегда является корневой узел обрабатываемого документа. Для последующих по порядку операций модификации корневой узел обрабатываемого документа служит базовым узлом в том и только том случае, если `выражение_XPath` данной операции модификации является абсолютным путем доступа (absolute location path) [12]. Если в операции модификации используется `выражение_XPath` другого вида, например, относительный путь доступа (relative location path), то в этом случае базовым узлом последовательно считается каждый из обрабатываемых узлов предыдущей операции модификации данного запроса на модификацию.

Сформулированное правило сопоставления базового узла с конкретной операцией модификации хорошо согласуется с семантикой вычисления путей доступа в спецификации XPath: абсолютный путь доступа вычисляется относительно корневого узла документа, а относительный путь доступа — относительно контекстного узла контекста вычисления. Использование относительного пути доступа XPath в операции модификации связывает эту операцию с предыдущей, и вычисление набора узлов, подлежащих обработке, осуществляется способом, во многом напоминающем вычисление последовательности шагов доступа в языке XPath:

- рассматривается набор обрабатываемых узлов предыдущей операции модификации;
- для каждого узла из данного набора, как для базового узла, вычисляется `выражение_XPath` текущей операции модификации;
- получаемые в результате вычисления `выражения_XPath` узлы становятся обрабатываемыми узлами в текущей операции модификации.

Расширим сигнатуру `обработчика` (являющегося функцией в подходе, предлагаемом в данной статье) вторым параметром — базовым узлом операции модификации:

```
(lambda (node base-node) ...) .  (3)
```

Рассмотренные в разделе 3 обработчики для наиболее употребительных операций модификации могут быть очевидным образом расширены на предмет наличия в сигнатуре обработчика нового аргумента `base-node`. Ввиду того, что базовый узел не влияет на результаты обработчиков для обсуждавшихся в разделе 3 наиболее употребительных операций модификации, базовый узел добавляется в эти обработчики просто в качестве фиктивного параметра.

С другой стороны, из расширенной базовым узлом сигнатуры обработчика легко видеть, что теперь он позволяет осуществлять зависимую обработку сразу двух узлов. Выразительные возможности языка XPath по адресации

структурных частей XML-документа позволяют базовому узлу и обрабатываемому узлу располагаться в разных частях обрабатываемого документа.

Функциональность зависимой обработки двух узлов, располагающихся в разнесенных местах документа, иллюстрируется в следующем подразделе реализацией упоминавшейся ранее в данном разделе операции перемещения поддерева.

4.2. Перемещение поддерева

При наличии введенного понятия базового узла перемещение поддерева может быть реализовано совокупностью двух операций модификации. На основе рассмотренной семантики вычисления нескольких операций модификации узел – корень поддерева, подлежащего перемещению, – может выступать в роли обрабатываемого узла для первой из двух операций модификации и в роли базового узла — для второй операции. Эффект перемещения поддерева достигается тогда, когда первая операция модификации удаляет свой обрабатываемый узел из документа, а вторая операция вставляет в документ свой базовый узел (являющийся обрабатываемым узлом для предыдущей операции модификации). Поскольку обе операции являются компонентами единого запроса на модификацию, влияние этих операций на обрабатываемый документ атомарно и с точки зрения реализации осуществляется за один проход по дереву документа.

Введенное правило вычисления базового узла естественным образом подходит для операции перемещения поддерева, потому что в практических задачах модификации новое местоположение перемещаемого поддерева часто задается относительно его исходного местоположения, и как раз при этом условии обрабатываемый узел первой из двух операций модификации, осуществляющих перемещение, становится базовым узлом для второй операции модификации.

Поскольку перемещение поддерева включает в себя удаление поддерева в его старом местоположении и вставку поддерева в новое местоположение, при формализации соответствующих действий в виде обработчиков могут быть почти без изменений использованы обработчики, рассмотренные в разделе 3. В зависимости от различных способов вставки перемещаемого поддерева на его новое местоположение разумно различать такие варианты, как “перемещение перед”, “перемещение после” и “перемещение внутрь”, соответствующие аналогичным вариантам операции вставки узла.

Пример 5 *Предположим, что электронная версия некоторой книги состоит из элементов с именем `глава (chapter)`, и каждая глава содержит элементы с именем `параграф (para)`. Переместим последний параграф главы, имеющей заголовок “Введение” (“Introduction”), в следующую по порядку главу в качестве ее первого параграфа.*

Запрос на желаемую модификацию реализуется следующей совокупностью из двух операций модификации, связанных друг с другом посредством базового узла:

```
(sxml:modify
   `("/book/chapter[title='Introduction']/
      para[last()]"
     ,(lambda (node base-node) '()))
   `("following::chapter[1]/para[1]"
     ,(lambda (node base-node)
        (list base-node node))))
```

Первая операция модификации выбирает параграф, подлежащий перемещению, и удаляет его по месту его исходного положения. Обрабатываемый параграф, удаленный из документа первой операцией модификации, тем не менее, является базовым узлом для второй операции модификации, поскольку во второй операции модификации в качестве выражения XPath используется относительный путь доступа. Вторая операция модификации осуществляет вставку перемещаемого параграфа – базового узла `base-node` *– перед первым по счету параграфом главы, следующей за главой с заголовком "Введение". Совокупный эффект обоих операций модификации рассматриваемого примера обеспечивает желаемое перемещение узла.*

Необходимо отметить, что обе операции модификации рассмотренного примера являются составляющими одного запроса на модификацию, и поэтому с точки зрения реализации, детали которой обсуждаются в разделе 6, выполняются за один проход по дереву обрабатываемого документа.

5. Сокращенный синтаксис для описания модификаций

Используемое в предыдущих разделах представление операции модификации в виде пары `(выражение_XPath обработчик)` может рассматриваться как полный синтаксис записи операции модификации. Помимо полного синтаксиса, являющегося универсальным для широкого класса операций модификации, в данной статье также предлагается сокращенный синтаксис для записи некоторых наиболее употребительных операций модификации.

Для рассмотренных выше операций вставки, удаления, переименования узла и т.п. предлагается идентифицировать каждую из этих операций своим ключевым словом, которое выражается типом данных "символ" языка Scheme. Благодаря быстроте операции сравнения в Scheme двух символов [20], при перезаписи запроса на модификацию из сокращенной формы записи в полную форму введенные ключевые слова можно быстро заменить сопоставленными им обработчиками.

Предлагаемые правила сокращенного синтаксиса для наиболее употребительных операций модификации приведены в табл. 2. Для операций модификации, требующих параметризации обработчиков (например, для вставки или переименования узлов), необходимые дополнительные параметры в сокращенном синтаксисе записываются после ключевого слова, идентифицирующего операцию модификации. Необходимо также заметить, что каждая из операций перемещения поддерева, представленная в табл. 2 одним списком, переписывается в терминах полного синтаксиса в совокупность из двух операций модификации, связанных друг с другом посредством базового узла. При перезаписи из сокращенного синтаксиса в полный синтаксис дополнительный параметр операции перемещения поддерева становится первым членом пары, выражающей вторую из двух операций модификации, в совокупность которых переписывается данная операция перемещения.

```
операция_модификации_в_сокращенном_синтаксисе ::=
`( ,выражение_XPath delete ) |
`(,выражение_XPath insert-following,добавляемый_узел_на_SXML)|
`(,выражение_XPath insert-preceding,добавляемый_узел_на_SXML)|
`(,выражение_XPath insert-into     ,добавляемый_узел_на_SXML)|
`(,выражение_XPath replace          ,заменяющий_узел_на_SXML)|
`(,выражение_XPath rename                    ,новое_имя_узла)|
`(,выражение_XPath move-following
                              ,выражение_XPath-новое_место_узла)|
`(,выражение_XPath move-preceding
                              ,выражение_XPath-новое_место_узла)|
`(,выражение_XPath move-into
                              ,выражение_XPath-
новое_место_узла)
```

Табл. 2: Сокращенный синтаксис для наиболее употребительных операций модификации

Помимо уменьшения чисто текстуального размера запроса на модификацию, предлагаемый сокращенный синтаксис также обеспечивает бо́льшую наглядность благодаря тому, что позволяет избавиться от функций в теле запроса на модификацию, и вместо них используются более привычные строки, символы и списки.

По сравнению с полным синтаксисом, сокращенный синтаксис обладает тем важным преимуществом, что может быть без потерь отображен на постоянное хранилище данных типа файла или на канал передачи данных, — тогда как тип данных *функция*, являющийся неотъемлемой частью полного синтаксиса, может существовать только во время вычисления программы, в оперативной памяти. Данное свойство сокращенного синтаксиса может быть полезно для таких областей применения предлагаемого языка модификации как генерация скрипта редактирования (edit script) в виде запроса на модификацию при определении различий между двумя деревьями [13].

6. Реализация

Предложенный в данной статье язык модификации XML-документов был реализован фукнциональными методами программирования на языке функционального программирования Scheme.

Необходимо заметить, что функциональный стиль программирования подразумевает определенные правила, накладываемые на сделанную реализацию. А именно, чистый функциональный стиль запрещает использование в программе таких побочных эффектов, как изменение значений, однажды связанных с глобальными идентификаторами. В контексте выполненной реализации языка модификации функциональный стиль программирования означает сохранение в неизменном виде документа, подлежащего модификации, и конструирование нового документа, выражающего собой результат вычисление требуемого запроса на модификацию к исходному документу.

В отличие от процедурной и объектно-ориентированной парадигм программирования, в которых модификация обычно реализуется как принудительное изменение исходных данных, функциональная парадигма программирования позволяет гарантировать, что данные, однажды связанные с идентификатором, останутся постоянными на протяжении всего времени, пока к этим данным есть доступ из какой-либо области программы [9]. Это свойство функционального программирования обладает тем важным преимуществом, что конструирование дерева модифицированного документа не требует глубокого копирования дерева исходного документа, и части, являющиеся общими для обоих деревьев, автоматически хранятся в единственной физической копии.

Возможность использования физически разделяемого поддерева, общего для нескольких деревьев, иллюстрируется на рис. 2. Предположим, что некоторый запрос на модификацию затрагивает узел, обозначенный на рис. 2 буквой O. Тогда в результате выполнения запроса на модификацию будут переконструированы только те части дерева обрабатываемого документа, которые обозначены на рис. 2 жирными линиями, а остальные части дерева останутся неизменными и, таким образом, будут существовать в разделяемой физической копии для исходного и сконструированного деревьев.

Из рис. 2 легко видеть, что модификация в SXML-документе некоторого узла затрагивает изменение этого узла и всех его узлов-предков. Поскольку на практике глубина XML-документов и соответствующих им SXML-документов обычно бывает небольшой, выполнение запроса на модификацию, выбирающего в документе лишь несколько узлов в качестве обрабатываемых, требует переконструирования только небольшой части модифицируемого документа. Узлы, остающиеся неизменными для исходной и модифицированной версий документа, существуют в единственной

физической копии, и обе версии документа адресуются к этим узлам по ссылкам.

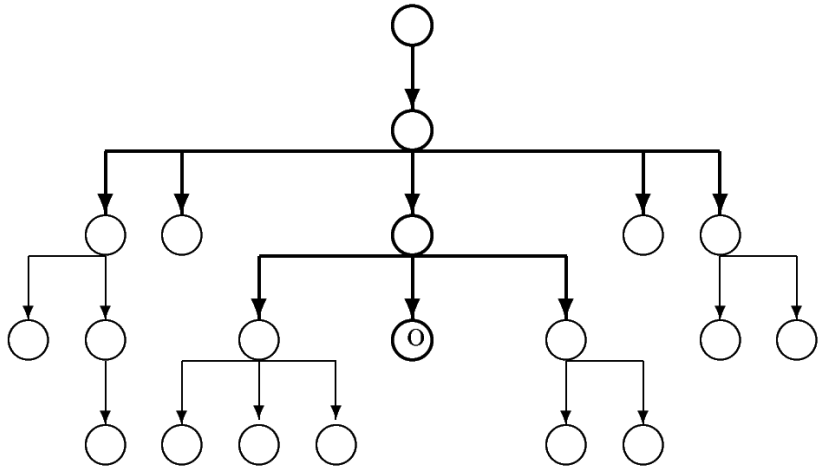

Рис. 2: Пример дерева SXML-документа, в которым производится модификация узла. Буквой O обозначен обрабатываемый узел, жирными линиями выделены те части дерева, которые изменяются в результате модификации

В зависимости от конкретной прикладной задачи, приложение, использующее функциональную реализацию предлагаемого языка запросов, само определяет, каким образом обрабатывать исходную и сконструированную новую версию модифицируемого SXML-документа. Например, приложение может использовать исходную версию документа только в локальном блоке своего кода, и тогда узлы, специфичные для исходной версии документа и не используемые в других местах прикладной программы, будут освобождены автоматическим сборщиком мусора [10]. Соображения по вопросу обеспечения конкурентной обработки общего SXML-документа несколькими приложениями обсуждается в разделе 8.

В сделанной реализации вычисление запроса на модификацию SXML-документа осуществляется в следующие два этапа:

- На первом этапе все обработчики сопоставляются с соответствующими им обрабатываемыми узлами, т.е. для каждой операции модификации вычисляется `выражение_XPath` этой операции относительно соответствующего базового узла. Заметим, что выработанный способ задания базовых узлов позволяет не

вызывать ни одного обработчика при определении базовых узлов для всех операций модификации.

- На втором этапе осуществляется обход дерева SXML-документа, для обрабатываемых узлов вызываются обработчики, и в соответствии с возвращаемыми результатами обработчиков конструируется дерево нового SXML-документа, соответствующего результату выполнения запроса на модификацию над исходным документом.

При вычислении выражений XPath, производимом на первом этапе обработки запроса на модификацию, желательно для каждого узла, выбранного выражением XPath, получить также все его узлы-предки, поскольку, как отмечалось выше, предков обрабатываемого узла в новой версии SXML-документа необходимо переконструировать. Для получения предков для узлов, адресуемых с помощью выражения XPath, была выбрана реализация XPath функциональными методами, обеспечивающая хранение предков контекстного узла в контексте вычисления и, благодаря этому, оптимизирующая вычисление обратных осей XPath [8]. Упомянутая реализация XPath позволяет получить результат вычисления выражения XPath в форме наборов контекстов, которые содержат всех предков для каждого из выбранных узлов SXML-документа.

Наличие известных предков каждого из обрабатываемых узлов позволяет организовать целенаправленный обход дерева SXML-документа на втором этапе обработки запроса на модификацию. А именно, благодаря тому, что корень дерева обрабатываемого документа является предком для каждого из обрабатываемого узлов, и обход дерева осуществляется сверху вниз, для любого поддерева можно сразу же установить, какие из обрабатываемых узлов располагаются внутри данного поддерева. Если в некотором поддереве не содержится ни одного обрабатываемого узла, то обход поддерева вниз может быть прекращен, потому что поддерево остается неизменным в результате выполнения запроса на модификацию.

Необходимо отметить, что семантика предлагаемого языка модификаций не противоречит тому, чтобы с одним и тем же узлом документа было ассоциировано сразу несколько обработчиков. Когда имеется несколько обработчиков, ассоциированных с одним узлом, результатом обработки данного узла служит суперпозиция применения обработчиков, и принимается следующая очередность применения обработчиков к узлу:

- Если какие-то два обработчика относятся к разным операциям модификации, то эти обработчики применяются в том порядке, в каком они записаны в запросе на модификацию;
- Если два обработчика относятся к одной операции модификации, то ввиду отсутствия совпадающих узлов в результате вычисления выражения XPath этим обработчикам с необходимостью

соответствуют разные базовые узлы. Для обработчиков, относящихся к одной операции модификации, очередность их применения принимается совпадающей с очередностью, которую имеют соответствующие обработчикам базовые узлы в порядке документа [12], подлежащего модификации.
Введенная очередность применения обработчиков позволяет ввести логичную и интуитивно ясную семантику операции перемещения поддеревьев. А именно, ситуации, когда с одним узлом ассоциировано несколько обработчиков одной и той же операции модификации, соответствует такой запрос на перемещение, при котором несколько разных поддеревьев перемещается на общее новое место. Например, если все сноски в электронном представлении некоторой книги перемещаются в главу “Приложение”, то в соответствии с принятым соглашением об очередности применения обработчиков сноски попадут в эту главу в точности в том порядке, в котором они встречались по тексту, что полностью согласуется с интуитивным представлением об ожидаемом результате перемещения.

Выполненная реализация языка модификаций обеспечивает проверку корректности производимых модификаций в соответствии с условиями правильной сформированности XML-документов (XML well-formedness) [19]. В частности, производится проверка на отсутствие дублирующих атрибутов у данного элемента и атомарность значений атрибутов. По аналогии с языком запросов к XML-документам XQuery [2], сделанная реализация языка модификаций разрешает приложению в результате модификации элемента записывать его атрибуты и прочее содержимое в произвольном порядке. Поскольку в синтаксисе SXML атрибуты данного элемента должны быть вложены в общий список атрибутов, непосредственно следующий за именем элемента, реализация производит объединение списков атрибутов, если в результате модификации у данного элемента их получилось несколько, и переносит список объединенных атрибутов в позицию, непосредственно следующую за именем элемента и предшествующую всему прочего содержимому.

7. Алгоритмическая сложность

В отношении алгоритмической сложности вычисления языка модификаций, предлагаемого в данной статье, можно сформулировать следующую теорему.

Теорема 1 *Введем следующие обозначения для характеристик запроса на модификацию и XML-документа, обрабатываемого с помощью данного запроса:*
n *— количество узлов в обрабатываемом документе;*
m *— количество операций модификации в запросе на модификацию;*

k_{max} *— максимальный размер* `выражения_XPath` *по всем операциям модификации [*21*];*
T(n) *— максимальное время работы каждого из обработчиков, присутствующих в запросе на модификацию. Время может зависеть от числа узлов в документе, например, в случае применения обработчика к корневому узлу документа;*
N(n) *— суммарное количество новых узлов, добавляемых в документ в результате выполнения запроса на модификацию.*
Тогда время вычисления запроса на модификацию может быть оценено сверху следующим выражением:

$$O(m \cdot k_{max} \cdot n^6) + \quad (4)$$
$$+ m \cdot n^2 \cdot T(n+N(n)) + \quad (5)$$
$$+ O((n+N(n))^2 \cdot \log_2(n+N(n))) \ . \quad (6)$$

Замечание 1 *Рассмотренные в разделах* 3 *и* 4 *обработчики для наиболее употребительных операций модификации вычисляются за константное время.*

В справедливости данного замечания легко убедиться простым анализом реализаций рассмотренных обработчиков. Отметим, что те из обработчиков, которые требуют параметризации (например, узлом, добавляемым в документ, или новым именем узла), будучи параметризованы, выполняются за константное время. Сама параметризация относится к фазе статического анализа запроса на модификацию, равно как и разбор участвующих в запросе на модификацию выражений XPath.

Замечание 2 *Данной теоремой устанавливается гарантированное полиномиальное от размера документа и от количества операций модификации время вычисления произвольного запроса на модификацию, при условии, что T(n) и N(n) полиномиальны от своих аргументов, т.е. время выполнения каждого обработчика полиномиально от размера документа, и суммарное количество новых узлов, добавленных в документ в результате выполнения запроса на модификацию, полиномиально от размера исходного документа.*

Доказательство 1 (теоремы) *Присутствующие в утверждении теоремы слагаемые, помеченные как (*4*), (*5*) и (*6*), относятся соответственно к следующим этапам вычисления запроса на модификацию:*

1. *к этапу сопоставления обработчиков с узлами;*

2. *к этапу обхода дерева документа и вызова обработчиков на обрабатываемых узлах;*

3. *к проверке корректности модификаций, произведенных по результатам вычисления обработчиков.*

Доказательство теоремы заключается в последовательном рассмотрении каждого из обозначенных этапов и получении верхней оценки времени вычисления по каждому этапу.

1. *Как обсуждалось в разделе* 6*, на этапе сопоставления обработчиков с узлами документа производится вычисление* `выражений_XPath` *всех операций модификации относительно базовых узлов этих операций. Ключевым моментом на данном этапе обработки является то, что в соответствии с семантикой предлагаемого в настоящей статье языка запросов максимальное количество базовых узлов для произвольной операции модификации не зависит от количества операций модификации в запросе на модификацию. Даже в том случае, когда все операции модификации в некотором запросе на модификацию связаны посредством базовых узлов (т.е. базовыми узлами каждой последующей операции модификации являются обрабатываемые узлы предыдущей операции модификации), максимальное количество базовых узлов для любой операции модификации не может превосходить общее число узлов в обрабатываемом документе, т.е. число n .*
   Для вычисления выражений `_XPath` *используется разработанная автором реализация XPath функциональными методами, которая включает в себя средства оптимизации [*21*], позволяющие получить гарантированную верхнюю оценку времени вычисления, равную:*

$$O(k_{max} \cdot n^5) \ .$$

   *Выражения XPath вычисляются относительно каждого из базовых узлов каждой из операции модификации. Умножая время вычисления выражения XPath на количество базовых узлов (которых, как показано выше, не больше n для каждой операции модификации) и на количество операций модификации m, получим (*4*).*

2. *На этапе обхода дерева SXML-документа производится вызов обработчиков на обрабатываемых узлах и формирование нового дерева в соответствии с возвращаемыми результатами обработчиков.*
   Количество обработчиков, которое может быть ассоциировано с одним узлом SXML-документа, не превосходит суммарного количества базовых узлов по всем операциям модификации, т.е. произведения m·n .
   Поскольку по семантике предлагаемого языка запросов несколько обработчиков, ассоциированных с одним и тем же узлом, вызываются по правилам суперпозиции, на время выполнения одного обработчика влияют узлы, которые могли быть добавлены в результате вызовов

предыдущих по порядку обработчиков, и поэтому это время оценивается сверху выражением T(n+N(n)).
Произведение максимального времени работы одного обработчика на максимальное количество ассоциированных с одним узлом обработчиков и на количество узлов в исходном документе и дает (5).

3. *Производимые проверки корректности сделанных модификаций уже не зависят от вида запроса на модификацию или количества операций модификации в нем, но зависят только от сконструированного результирующего SXML-документа. В терминах введенных в условии теоремы обозначений, данное соображение означает, что временная оценка этапа проверки корректности сделанных модификаций зависит только от числа (n+N(n)), т.е. от количества узлов в новом SXML-документе.*
Среди производимых проверок асимптотически доминирует проверка на отсутствие у данного элемента нескольких атрибутов с одинаковыми именами.
Проверка на отсутствие совпадающих имен атрибутов у данного элемента производится с помощью известного алгоритма сортировки слиянием по именам атрибутов данного элемента. Время работы алгоритма сортировки слиянием оценивается выражением

$$O((n+N(n))\cdot log_2(n+N(n))) \quad .$$

Ввиду того, что данная проверка производится для каждого элемента в результирующем SXML-документе, записанная оценка умножается на число узлов в документе, что окончательно дает (6).

Сумма полученных временных оценок по каждому из этапов вычисления запроса на модификацию и обеспечивает утверждение теоремы.

8. Будущие исследования

Объектом дальнейших исследований является предоставление полного набора свойств транзакций при модификации SXML-документов, и, прежде всего, обеспечение изоляции транзакций, осуществляющих конкурентную работу над общим SXML-документом.

Ввиду того, что выполнение любого запроса на модификацию SXML-документа подразумевает переконструирование корневого узла дерева документа[2], максимальный параллелизм транзакций, который представляется реальным обеспечить в выбранном подходе обработки XML-данных

[2] данное утверждение справедливо для любого запроса на модификацию, который содержит по крайней мере один обрабатываемый узел

функциональными методами, выражается схемой “много читателей, один писатель”. В то время как одна транзакция осуществляет модификацию SXML-документа, остальные транзакции могут продолжать читать те части документа, которые не затрагиваются проводимой модификацией.

Изолированность транзакций на уровне отсутствия чтения “грязных данных” [22] может быть реализована с помощью механизма блокировок древовидных структур [23] или мониторов Хоара. Целью текущих исследований является разработка таких методов изоляции транзакций, которые бы гарантировали отсутствие неповторяющихся чтений и проблемы фантомов, и которые бы в минимальной степени влияли на параллелизм транзакций.

9. Заключение

Статья посвящена вопросу исследования и разработки языка модификации для XML-данных. Были проанализированы ограничения, присущие существующим языкам модификации XML-данных, и для преодоления этих ограничений были предложены функциональные методы программирования. При определении семантики действий, осуществляемой конкретной операцией модификации, было предложено использовать функции, обладающие в языке функционального программирования Scheme свойством объектов первого класса, в роли обработчиков для операций модификации. Представление XML-данных в виде списков формата SXML позволило — благодаря наличию в языке Scheme большого количества стандартных функций для работы со списками — в простой и лаконичной форме сформулировать обработчики для всех наиболее употребительных операций модификации.

Для согласованного обеспечения операций модификации, подобных операции перемещения поддеревьев, была выработана более широкая возможность зависимой обработки сразу нескольких расположенных в разных местах документа узлов. Предложенная концепция зависимой обработки узлов естественным образом расширяет идею обработчиков благодаря введенному понятию базового узла операции модификации.

Для возможности написания запросов на модификацию без явной передачи функций в тело запроса, был разработан сокращенный синтаксис для набора наиболее употребительных операций модификации.

В отношении предложенного языка модификаций была сформулирована и формально доказана верхняя временная оценка, утверждающая гарантированное полиномиальное от размера документа и от количества операций модификации время вычисления любого запроса на модификацию, при условиях, что каждый обработчик выполняется за полиномиальное время и что суммарное количество новых узлов, добавляемых в документ в результате модификации, полиномиально от размера исходного документа.

Практическая применимость предложенных в статье идей была проиллюстрирована сделанной реализацией. Реализация использовалась при тестировании операций модификации в разрабатываемой в Институте системного программирования РАН XML-СУБД Sedna [24].

Список литературы

[1] Когаловский М.Р. Глоссарий по стандартам платформы XML. Версия 7 (17-12-2006). // Электронные библиотеки. – М.: ИРИО, 2006. http://www.elbib.ru/index.phtml?page=elbib/rus/methodology/xmlbase/glossary_XML

[2] The World Wide Web Consortium (W3C). XQuery 1.0: An XML Query Language : W3C Recommendation / Boag S. et al. (eds.). – 2007, 23 Jan. http://www.w3.org/TR/2007/REC-xquery-20070123/

[3] The World Wide Web Consortium (W3C). XQuery Update Facility 1.0 : W3C Working Draft / Chamberlin D. et al. (eds.). – 2007, 28 Aug. http://www.w3.org/TR/2007/WD-xquery-update-10-20070828/

[4] Лизоркин Д.А., Лисовский К.Ю. SXML: XML-документ как S-выражение // Электронные библиотеки. – М.: ИРИО, 2003. – Т. 6, вып. 2. – ISSN 1562-5419 = Russian digital libraries journal. http://www.elbib.ru/index.phtml?page=elbib/rus/journal/2003/part2/LK

[5] The World Wide Web Consortium (W3C). XML Information Set : W3C recommendation / Cowan J., Tobin R. (eds.). – 2nd edition. – 2004, 4 Feb. http://www.w3.org/TR/xml-infoset/

[6] Kiselyov O. SXML specification / Rev. 3.0. – ACM SIGPLAN Notices. – New York: ACM Press, 2002. – Vol. 37, N 6. – pp. 52-58. – ISSN 0362-1340. http://okmij.org/ftp/Scheme/SXML.html

[7] Kiselyov O. A better XML parser through functional programming : Proc. Practical Aspects of Declarative Languages: 4th int. symposium, PADL'2002, Portland, 19-20 Jan., 2002 // Lecture Notes in Computer Science / Krishnamurthi S., Ramakrishnan C.R. (eds.). – Springer-Verlag Heidelberg, 2002. – Vol. 2257. – pp. 209-224. – ISSN: 0302-9743. http://www.okmij.org/ftp/papers/XML-parsing.ps.gz

[8] Лизоркин Д.А. Оптимизация вычисления обратных осей языка XML Path при его реализации функциональными методами // Сборник трудов Института системного программирования РАН / Под ред. чл.-корр. РАН Иванникова В.П. – М.: ИСП РАН, 2004. – Т. 8, ч. 2. – 214 с. – с. 93-119. – ISBN 5-89823-026-2. http://modis.ispras.ru/Lizorkin/Publications/xpath-context.ps

[9] Abelson H., Sussman G.J., Sussman J. Structure and Interpretation of Computer Programs. – 2nd ed. – London; New York: The MIT Press; McGraw Hill, 1996. – 657 pp. – ISBN 0-262-01153-0. http://mitpress.mit.edu/sicp/

[10] Boehm H.-J. Space efficient conservative garbage collection : Proc. conf. on Programming Language Design and Implementation (SIGPLAN'93), Albuquerque, NM, Jun. 1993 // ACM SIGPLAN Notices. – New York: ACM Press, 1993. – Vol. 28(6). – pp. 197-206; New York: ACM Press, 2004. – Vol. 39, issue 4. – pp. 490-501. – ISSN 0362-1340. http://citeseer.ist.psu.edu/259238.html

[11] Lehti P. Design and Implementation of a Data Manipulation Processor for a XML Query Language : Ph.D. thesis. – 2001, Aug. www.lehti.de/beruf/diplomarbeit.pdf

[12] Консорциум W3C. Язык XML Path (XPath) версия 1.0 = XML Path Language (XPath) Version 1.0 : Рекомендация Консорциума W3C / Под ред. Clark J., DeRose S.; пер. с англ. Усманов Р. – 1999, 16 ноя. http://citforum.ru/internet/xpath/index.shtml

[13] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina and J. Widom. Change Detection in Hierarchically Structured Information. Technical report; detailed version of paper appearing in SIGMOD 1996. http://www-db.stanford.edu/c3/papers/html/tdiff3-8/tdiff3-8.html

[14] Igor Tatarinov, Zachary G. Ives, Alon Y. Halevy, Daniel S. Weld. Updating XML : Proc. ACM SIGMOD int. conf. on Management of Data (SIGMOD'01), Santa Barbara, California, 21-24 May, 2001 // SIGMOD Conference / Aref W.G. (ed.). – New York: ACM Press, 2001. – pp. 413-424. – ISBN 1-58113-332-4. http://citeseer.comp.nus.edu.sg/603731.html

[15] Лизоркин Д.А., Лисовский К.Ю. Язык XML Path (XPath) и его функциональная реализация SXPath // Электронные библиотеки. – М.: ИРИО, 2003. – Т. 6, вып. 4. – ISSN 1562-5419 = Russian digital libraries journal. http://www.elbib.ru/index.phtml?page=elbib/rus/journal/2003/part4/LL

[16] A. Laux, L. Martin. XUpdate update language : XML:DB Working Draft. – 14 Sep, 2000. http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html

[17] Лизоркин Д.А., Лисовский К.Ю. Реализация XLink – языка ссылок XML – с помощью функциональных методов // Программирование. – М.: Наука, 2005. – N 1. – С. 52-72. – ISSN 0361-7688 = Programming and computer software.

[18] Лизоркин Д.А. Язык запросов к совокупности XML-документов, соединенных при помощи ссылок языка XLink // Сборник трудов Института системного программирования РАН / Под ред. чл.-корр. РАН Иванникова В.П. – М.: ИСП РАН, 2004. – Т. 8, ч. 2. – 214 с. – с. 121-153. – ISBN 5-89823-026-2; Программирование. – М.: Наука, 2005. – N 3. – ISSN 0361-7688 = Programming and computer software.

[19] The World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.0 (Fourth Edition) : W3C Recommendation / Bray T. et al. (eds.). – 2006, 16 Aug. http://www.w3.org/TR/2006/REC-xml-20060816

[20] Лизоркин Д.А., Лисовский К.Ю. Пространства имен в XML и SXML // Электронные библиотеки. – М.: ИРИО, 2003. – Т. 6, вып. 3. – ISSN 1562-5419 = Russian digital libraries journal. http://www.elbib.ru/index.phtml?page=elbib/rus/journal/2003/part3/LL

[21] Лизоркин Д.А. Функциональные методы обработки XML-данных / Ph.D. thesis = Диссертация на соискание ученой степени кандидата физико-математических наук по специальности 05.13.11. – 2005, 11 ноя.

[22] Кузнецов С.Д. Современные технологии баз данных : Годовой курс для студентов 3-го курса. – Центр Информационных Технологий. http://citforum.ru/database/osbd/contents.shtml

[23] Гарсиа-Молина Г., Ульман Дж.Д., Уидом Дж. Системы Баз Данных. Полный курс / Пер. с англ. Варакина С.А.; под ред. Слепцова А.В. – М,: Вильямс, 2003. – 1088 с.: ил. – ISBN 5-8459-0384-X (рус.)

[24] Гринев М., Кузнецов С.Д, Фомичев А. XML-СУБД Sedna: технические особенности и варианты использования // Открытые системы. – 2004, вып 8. http://www.osp.ru/os/2004/08/185085/