УДК 681.3.06

В.И. Каук, Г.В. Слободырев

МЕТОДЫ ИССЛЕДОВАНИЯ ИНТЕЛЛЕКТУАЛЬНОЙ ОБРАБОТКИ ТЕКСТА С ИСПОЛЬЗОВАНИЕМ XML-ПАРСЕРОВ

1. Введение

В последние годы консорциум W3C очень интенсивно работал над спецификациями XML, были существенно изменены и доработаны многие из них, добавлено большое количество новых. В связи с этим, разработка приложений, поддерживающих в полном объеме рекомендации W3C, стала очень непростой задачей. Это привело к тому, что на рынке приложений для анализа XML-документов наметилась некоторая стабилизация: количество компаний, предлагающих в данный момент полнофункциональные процессоры XML, не так уж и велико. Среди них можно выделить следующие: IBM, Microsoft, Oracle, Ginger Alliance Ltd, Sun. Есть также некоторое количество инструментов, разработанные отдельными людьми, например, Patrice Bonhomme, James Clark. При этом, даже у перечисленных производителей процессоры имеют значительные отличия, не следуя в полном объеме рекомендациям консорциума. Из этого можно сделать вывод, что в ближайшее время могут появляться скорее новые версии этих процессоров, а не новые программные продукты. Остальные же приложения для работы с XML-документами будут использовать какие-либо из этих парсеров. В то же время было предпринято немало попыток сделать подобные приложения и другими фирмами, но уже на данный момент многие из этих проектов закрыты и процессоры больше не поддерживаются.

2. Постановка задачи

Динамическое развитие информационных технологий, ориентированных на работу с XML, предоставляет качественно новые возможности обработки XML-документов и способствует быстрому и удобному созданию ресурсов в сети Интернет. В связи с этим, возникает необходимость написания своего собственного XML-процессора, под конкретные цели проекта. Такая необходимость может привести к приличным затратам времени и средств. На данный момент уже существует достаточно парсеров, и следовательно можно выбрать наиболее подходящий для данного проекта парсер. Единственное, что нужно сделать, это выдвинуть четкие требования к парсеру. На данный момент наиболее часто используемыми парсерами являются, по-видимому, парсеры IBM и James Clark. В последнее время очень серьезные усилия направила на разработку процессора XSLT корпорация Microsoft. Последний вариант ее MSXML показывает очень хорошие результаты как по производительности, так и по соответствию последним рекомендациям W3C. Поэтому для определения эталонного парсера (парсера, который соответствует рекомендациям W3C) необходимо выработать методологию тестирования парсеров по определенным параметрам и разработать программно-инструментальное средство для тестирования существующих парсеров.

3. XML-парсеры

В последние годы консорциум W3C очень интенсивно работал над спецификациями XML, были существенно изменены и доработаны многие из них, добавлено большое количество новых. В связи с этим, разработка приложений, поддерживающих в полном объеме рекомендации W3C, стала очень непростой задачей. И следовательно на данный момент существует большое количество разнообразных приложений, которые в той или иной мере реализуют эти стандарты. При чем каждый это делает по своему. В ходе поиска и классификации можно выделить такие категории приложений для работы с XML-документом:

— язык реализации (влияет на скорость работы парсера, переносимость и удобство использования);

— способ реализации (библиотека процедур для разбора XML-документа или законченная программа);

— функциональное назначение (средства для просмотра XML и DTD, редакторы XML и DTD, программы для обеспечения поиска по XML-тексту, валидаторы, трансформаторы, отладчики, парсеры);

Парсеры, видимо, основная часть всего этого многообразия приложений. На данный момент для парсеров сложилось два стандартных API, одно из которых должен реализовывать каждый парсер. Первое это SAX (simple API for XML), автором которого является David Megginson, второе — стандарт DOM от W3C. SAX построен на механизме обратных вызовов, пользователь должен предоставить класс, который будет реагировать на события разбора. Примерами таких событий являются начало документа, начало тэга и т. п. Стандарт DOM, напротив, предоставляет пользователю определенное API к документу, который наиболее естественно отображается на структуру дерева. В качестве примера парсера, реализующего DOM, можно привести Xml4J от IBM. Зачем же потребовалось создавать эти две модели? Ответ очень прост. Очевидно,

представление DOM наиболее просто. С его помощью можно легко получить доступ к любому элементу документа или модифицировать его. Вся проблема заключена в цене этой простоты. Для некоторых приложений совсем не обязательно представление документа в таком виде, им нужно гораздо меньшее, например, выбрать все элементы первого уровня. Для других необходимо собственное внутреннее представление документа, возможно для оптимизации доступа к некоторым узлам. И в том, и в другом случае дополнительный уровень преобразования документа в DOM-дерево может оказаться непозволительной роскошью. Можно отметить, что разработка SAX-парсера гораздо более простое дело, чем парсера, поддерживающего DOM. Это отчетливо сказывается на предложении парсеров: SAX-парсеров можно найти гораздо больше, чем парсеров DOM. Кроме того, как правило, DOM-парсеры включают в себя классы, реализующие SAX-интерфейс.

Любой XML-процессор, являясь, по сути, транслятором языка разметки, может быть разбит на несколько модулей, отвечающих за лексический, синтаксический и семантический анализ содержимого документа. Понятно, что если бы мы были вынуждены каждый раз писать все эти блоки самостоятельно, необходимость в XML как в таковом бы отпала — основное его преимущество, как уже упоминалось ранее, заключается в стандартном способе извлечения информации из документа. Синтаксически правильно составленный XML-документ может быть разобран любым универсальным XML-анализатором, и нашему XML-обработчику остается лишь использовать полученные на его выходе «чистые» данные (прошедшие синтаксический анализ) — интерпретировать содержимое документа, в соответствии с его DTD-описанием или схемами данных (рис. 1).

4. Методика тестирования

Для тестирования парсеров изложим методологию проведения тестов процессоров и с этих позиций требования к тестовым примерам. Дело в том, что корректный результат преобразования может быть представлен неоднозначно. Например, может варьироваться порядок параметров, если они допускают перестановку по DTD. Этот факт сильно усложняет задачу проверки работоспособности приложения. Для тестирования парсеров XML можно предложить следующую схему: определим некоторое подмножество языка XML, которое назовем каноническим XML. Будем использовать канонический XML для представления результатов разбора документа. К каноническому представлению выдвинем следующие требования: каждый хорошо оформленный XML-документ должен обладать единственным структурно эквивалентным ему каноническим документом. Два структурно эквивалентных XML-документа должны иметь побайтно идентичные канонические представления.

Предложим грамматику для записи канонических XML-документов:

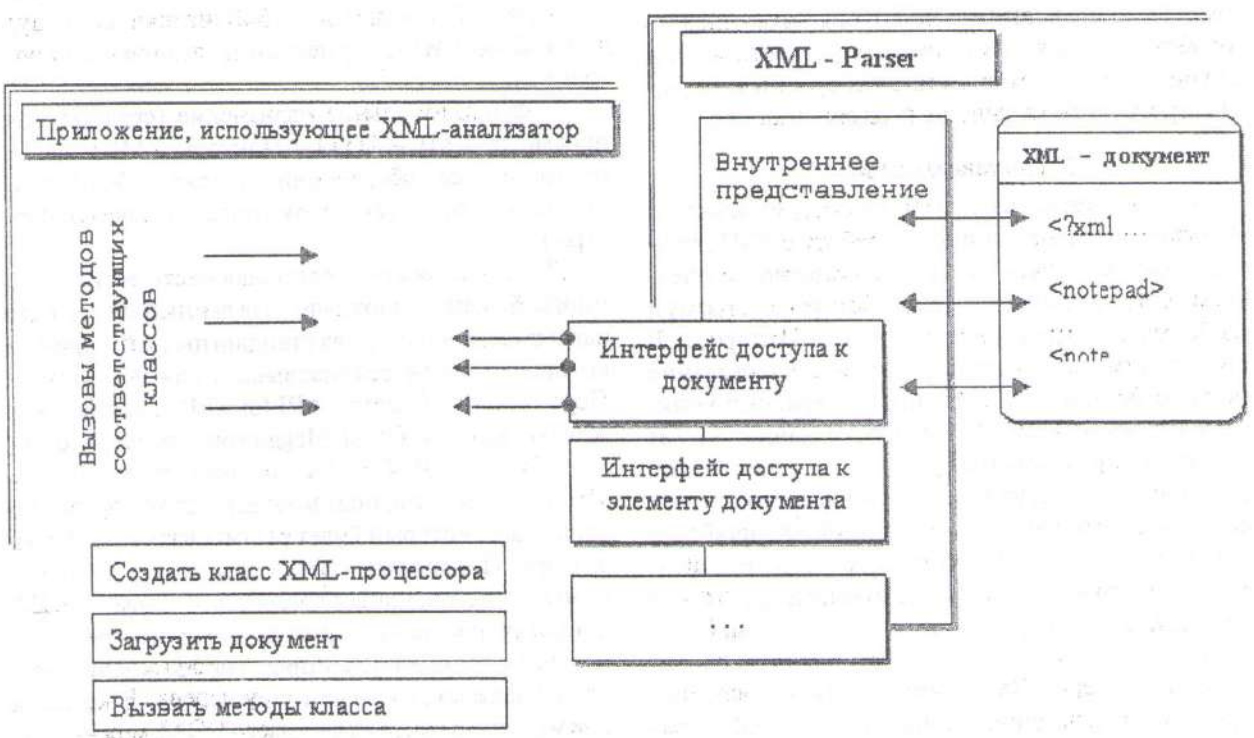

Рис. 1. Принцип работы парсера

```
CanonXML::= Pi* element Pi*
element::= Stag (Datachar | Pi | element)* Etag
Stag ::= '<' Name Atts '>'
Etag ::= '</' Name '>'
Pi::='<?'Name''(((Char-S)Char*)?-
(Char*'?>'Char*))'?>'
Atts ::= ('' Name '='''' Datachar* '''')*
Datachar::='&amp;'|'&lt;'|'&gt;'|'&quot;'|'&#9;'|
'&#10;'|'&#13;'|(Char - ('&'|'<'|'>'|''''|#x9|
#xA|#xD))
Name::= (see XML spec)
Char::= (see XML spec)
S::= (see XML spec)
```

Канонизация XML-документа требует только ту информацию о документе, которую парсер должен предоставить приложению.

Для атрибутов фиксируется лексикографический порядок (в соответствии со стандартом Unicode). Кодировка канонического XML-документа UTF-8. Пробельные символы, которые по спецификации могут быть игнорированы парсером, считаем значимыми и рассматриваем одинаково с прочими символьными данными.

Таким образом, построив некоторый канонизатор, мы можем автоматизировать процесс тестирования XML-документов. Всех проблем, однако, наличие возможности канонизировать XML и канонизатора не решает.

Тут предлагается следующая схема: в процессе тестирования готовых процессоров мы можем выявить наиболее доверенный на основании каких-либо критериев, которые можно обсудить, а затем пользоваться им для получения своего рода псевдоканонических результатов. Естественно, остается и прямолинейный вариант: сделать руками набор тестовых примеров, покрывающий максимально возможное число аспектов спецификации, и для них точно таким же образом написать каноническую форму. Создав такой набор тестов, можно достаточно легко анализировать конформность парсеров. Правда, спецификации меняются, следовательно, возникает проблема поддержания конформности самого набора тестов, что является непростой задачей. На данный момент существует по крайней мере одно достаточно комплексное решение этой проблемы. Это так называемый OASIS XML Conformance Test Suite набор тестов, подобранных для наиболее качественного тестирования конформности XML-процессоров. На него мы и будем опираться при проведении тестов.

Тестирование должно выявлять как конформность, так и быстродействие процессоров.

Для этого при подборе примеров осуществляется первичное тестирование, смысл которого без замера времени прохождения примеров определить, насколько данные примеры пригодны для тестирования различных процессоров. Проверка корректности результатов во многих случаях не осуществлялась. Полученная в результате работы библиотека перекрывает основные интересующие нас аспекты работы процессоров. В случае расширения или изменения классификации, возможно ее дополнение. Как дополнительную характеристику сложности теста можно пользоваться какой-либо статистикой. Например, хорошую согласованность с опытом дает формула затэггиности (taggingness ratio) документа, предложенная Microsoft:

$$Syntaxchar=NameChar+(Elements*2)+(AttributeNode*3)$$

$$Percent=syntaxChars/(syntaxChars+TextCChars)*100\%$$

Грубо говоря, это некоторое отношение количества символов в тэгах к полному размеру файла.

В процессе тестирования ставилась задача выявления следующих характеристик приложений:

— полнота реализации стандартов;

— диагностика ошибок;

— скорость работы;

— возможность работы с документами большого объема;

— соответствие требованиям пользователей.

Приведем требования к тестовым примерам. Очевидно, тестовые примеры должны быть подобраны таким образом, чтобы обеспечить наиболее полное исследование характеристик приложений. Очевидно, для тестирования правильности обработки XML-документов парсером большой размер документа не является необходимым, однако весь набор тестовых файлов, в котором присутствует около 500 файлов, является прекрасным тестом для парсера и практически полностью в состоянии покрыть изложенный выше перечень. Далее, для тестирования скорости работы полезным окажется набор файлов среднего размера. В качестве примера можно взять набор тестов Docbook. Для проверки способности работы парсеров с большими документами можно, например, искусственно увеличить размер предложенных выше тестов путем слияния файлов в один большой. Для тестирования пригодности для практического использования можно использовать как вышеперечисленные примеры, так и скрипты MathML, SVG.

Итак, в наборе тестовых примеров должны присутствовать следующие типы документов XML:

— документы, не являющиеся хорошо оформленными;

— документы, не являющиеся состоятельными;

— документы, являющиеся хорошо оформленными;

— документы, являющиеся состоятельными.

Для состоятельных документов можно предложить следующие важные подразделы:

— документы со значением yes атрибута standalone;

— документы со значением no атрибута standalone.

Тесты бывают двух типов:

— бинарные тесты и тесты с проверкой результата преобразования (бинарные тесты собраны в четыре категории, в зависимости от категории, процессор должен пройти или завершиться с ошибкой, в зависимости от того, валидирующий или нет XML-процессор);

— тесты с проверкой результата преобразования.

Конформность процессора измеряется в процентах пройденных тестов. Для общей оценки производительности процессоров необходимо замерять полное время обработки XML-документа и XSLT-кода.

Для общей оценки производительности процессоров необходимо замерять полное время обработки XML-документа и XSLT-кода. Однако это далеко не самый интересный показатель. Для анализа временных характеристик и осмысленного тестирования скоростей необходимо выделить основные этапы работы XML-процессоров. После их выявления нужно провести замеры времени как работы процессора в целом, так и время на выполнения этих фаз. Предлагается выделить такие этапы обработки документов во всех тестируемых процессорах.

4.1. Фазы работы XML-процессоров.

В качестве основного документа для выделения схемы работы процессора воспользуемся результатами анализа В. Рогановым процессора XSLT Джэймса Кларка. В этом документе достаточно подробно проанализирована структура кларковского парсера. Аналогичные этапы обработки документов могут быть выделены и в других тестируемых приложениях.

К счастью, для рассматриваемых процессоров, как правило, не составляет большого труда отделить фазы построения преобразователя от собственно преобразования XML-документа. Тем самым можно во многом ослабить влияние предкомпиляции XSLT. По наблюдениям, для большинства примеров фаза преобразования гораздо более трудоемкая, чем построение преобразователя.

Итак, мы будем выделять следующие фазы преобразоваия в тестируемых процессорах (на примере XT):

1) загрузка XSLT (правил преобразования XML);

2) переработка дерева распарсированного XSLT в дерево действий;

3) загрузка входного XML-документа;

4) обработка документа с параллельной выдачей результата;

5) выдача результата.

Завершение работы отдельно не выделяется.

Обобщим вторую фазу как этап предкомпиляции XSLT (по-видимому, именно в таком виде она и встречается в остальных процессорах). Будем называть эту фазу предкомпиляцией, подчеркивая, что она осуществляется до начала трансформации документа

В фазе 4 результат трансформации может быть представлен и иным способом. По этой причине выделим пятую фазу, которую назовем выдачей результата.

Исходя из общих задач, сформулируем следующие этапы работы:

1) создание приложений на JAVA или C++ для автоматизации тестирования процессоров в целом и по фазам обработки документов;

2) создание собственных тестовых примеров, объединяющих усложненный код XSLT с существенным объемом документа;

3) определение скоростей работы процессоров;

4) определение скоростей работы различных фаз работы процессоров.

5. Выводы

Наиболее интересным вариантом примеров является набор примеров XML Conformance Test Suite. Два первых набора: тесты XML Conformance Test Suite и примеры Docbook покрывают основные вопросы тестирования. Процессор, успешно прошедший эти тесты, имеет очень высокие шансы пройти все остальные. Однако, если при работе с DocBook возникли проблемы, можно испытать приложение на наборах более простых тестов. Необходимое в дальнейшем сравнительное тестирование времени работы процессоров рекомендуется проводить на указанных выше наборах примеров. Вопрос о выделении эталонного парсера пока оставлен открытым.

Список литературы: 1. *Б.Мак-Лахлин.* «Java и XML» 2-е издание. Издательство: Символ. 2002. — 480 с. 2. «XML для профессионалов». Издательство: Лори. 2001. — 866 с. 3. *С. Холзнер,* «XML. Энциклопедия — Real World XML». Издательство: Питер. 2004. — 1104 с. 4. *С. Холзнер,* «XSLT. Библиотека программиста. Inside XSLT». Издательство: Питер. 2003. — 544 с. 5. *П. Дейтел, Х. Дейтел, Т. Лин, Т. Нието, П.Садху,* «Как программировать на XML». Издательство: Бином. 2001. — 944 с. 6. *А.Гомер,* «XML и IE5. Справочник программиста». Издательство: Лори. 2001. — 418 с.

Поступила в редколлегию 29.11.2005