Государственное образовательное учреждение высшего профессионального образования
«Казанский государственный университет им. В.И. Ульянова-Ленина»

ЯЗЫКИ РАЗМЕТКИ СЕМАНТИЧЕСКОГО ВЕБА
Практические аспекты

А. М. Елизаров, Е. К. Липачёв, М. А. Малахальцев

Казань 2008

Учебно-методическое пособие по направлению «Электронные образовательные ресурсы». - Казань: КГУ, 2008.

Учебно-методическое пособие публикуется по решению Учебно-методической комиссии Института непрерывного образования КГУ от 4 июля 2008 г.

Авторы-составители:
доктор физико-математических наук, профессор, Елизаров А. М.
кандидат физико-математических наук, доцент, Липачёв Е.К.
кандидат физико-математических наук, доцент, Малахальцев М. А.

Рецензент: кандидат педагогических наук Абросимов А.Г.

Аннотация

Основной целью руководства является описание специализированных языков разметки, построенных на основе XML. Дано краткое описание XML, DTD, XML Schema, XML Namespace, XSL, приведены примеры, иллюстрирующие назначение и особенности указанных технологий. Показано, как начать проектирование собственного языка разметки на основе XML. Знакомство с уже созданными специализированными языками разметки, описанными в руководстве, призвано помочь читателю ориентироваться в постоянно расширяющемся множестве языков разметки семантического веба.

Для научных работников, преподавателей, аспирантов и студентов, специализирующихся в области естественных наук.

Оглавление

Введение

Руководство посвящено языкам разметки, спроектированных в соответствии с принципами семантического веба. Дается сжатое изложение спецификаций XML, DTD, XML Schema, XML Namespace, XSL, которое сопровождается примерами. Описываются языки разметки CML, MathML, MatML, WML и приводится ознакомительная информация о ряде других специализированных языков разметки. Показана схема проектирования собственного языка разметки на основе XML.

Разметка текста и языки разметки

Язык разметки (**текста**) представляет из себя набор символов или последовательностей, вставляемых в текст для передачи информации о его выводе или строении. Документ, написанный с использованием языка разметки, содержит не только сам текст (как последовательность слов и знаков препинания), но и дополнительную информацию о различных его участках — например, указание на заголовки, выделения, списки и т. д.

Различают логическую и визуальную разметки. В первом случае речь идет только о том, какую роль играет данный участок документа в его общей структуре (например, «данная строка является заголовком»). Во втором определяется, как именно будет отображаться этот элемент (например, «данную строку следует отображать жирным шрифтом») (см., например, [1]).

Один и тот же текст можно разметить, используя разные языки разметки. Например, следующий текст размечен с помощью HTML:

```
<html>
<body background="#FFFFFF">
   <h1>Что такое разметка</h1>
    <p>
    Текст обычно форматируют, выделяя определенные
фрагменты курсивом, подчеркиванием и т. д. Это
достигается путём расстановки в тексте специальных
значков, так называемых <I>кодов разметки</I>.
</body>
</html>
```

Этот же текст можно разметить, например, с помощью LaTeX. Документ, сформатированный в MS Word, также содержит разметку, но она является скрытой.

Generalized Markup Language - GML

Одним из первых языков разметки был язык GML (Generalized Markup Language), разработанный IBM в 1960 г.

Сотрудники IBM Чарльз Гольдфарб (Charles Goldfarb), Эд Мошер (Ed Mosher) и Рэй Лори (Ray Lorie) сформулировали три общих принципа, обеспечивающих возможность совместной работы с документами в разных операционных системах:

1. **Использование единых принципов форматирования во всех программах, выполняющих обработку документов.** Вполне логичное требование — всем нам хорошо известно, как трудно договориться между собой людям, говорящим на разных языках. Наличие единого набора синтаксических конструкций и общей семантики заметно упрощает взаимодействие между программами.
2. **Специализация языков форматирования.** Благодаря возможности построения специализированного языка на базе набора стандартных правил программист перестает зависеть от внешних реализаций и их представлений о потребностях конечного пользователя

Четкое определение формата документа. Правила, определяющие формат документа, задают количество и маркировку языковых конструкций, используемых в документе. Применение стандартного формата гарантирует, что пользователь будет точно знать структуру содержимого документа. Обратите внимание: речь идет не о формате *отображения* документа, а о его структурном формате. Набор правил, описывающих этот формат, называется «определением типа документа» (document type definition, DTD)

Эти три правила были заложены в основу GML (Generalized Markup Language). Исследования и разработка GML продолжались около десяти лет, пока в результате соглашения, заключенного международной группой разработчиков, не появился стандарт SGML.

Standard Generalized Markup Language – SGML

3. В 1980-х годах необходимость в общих средствах обмена информацией непрерывно возрастала, и SGML вскоре превратился в отраслевой стандарт (в 1986 году он был принят в качестве стандарта ISO («ISO 8879:1986 Information processing – Text and office systems – Standard Generalized Markup Language (SGML)»).

SGML представляет собой международный стандарт обмена электронной информацией между различными аппаратными и программными компонента-

ми. Он определяет формализованный набор правил для создания языков. На базе SGML были созданы языки разметки — HTML, XML и SGML Docbook. Отметим, что SGML предоставляет множество вариантов синтаксической разметки для использования различными приложениями, однако его сложность затруднила его широкое распространение.

HTML

Идея передачи гипертекстовых документов через web-браузер, предложенная Тимом Бернерсом-Ли (Tim Berners-Lee), не требовала многих возможностей, поддерживаемых полной реализацией SGML. В результате появился известный язык разметки HTML.

Тем не менее, у HTML имеется существенный недостаток: он не позволяет разработчику создавать собственные типы документов. Результатом стала «война браузеров», в ходе которой разработчики браузеров начали создавать свои собственные усовершенствования языка HTML. Эти модификации существенно отклонялись от идеи работы с единым стандартом HTML и вызвали настоящий хаос среди разработчиков, которые хотели создавать web-сайты, не зависящие от браузера. Более того, долгий период неопределенности в области стандартов привел к тому, что разработчики вывели язык из первоначально задуманных границ.

XML и XML-технологии

Спецификация XML (E**x**tensible **M**arkup **L**anguage) разработана в 1996 году Рабочей группой консорциума World Wide Web Consortium (W3C) и впоследствии предложена в качестве рекомендации. В нормативном документе указаны цели создания этой спецификации, сформулированные в виде десяти заповедей. По своему замыслу технология XML должна обеспечить отделение информации от разметки, что позволяет производить обработку, поиск и представление информации на более высоком технологическом уровне.

Последняя, на настоящий день, четвертая редакция XML 1.0 доступна по адресу http://www.w3.org/TR/2006/REC-xml-20060816/. Вторую редакцию спецификации XML 1.1 можно скопировать с ресурса http://www.w3.org/TR/2006/REC-xml11-20060816/.

Отметим, что спецификация XML является основой для построения грамматики языков разметки, и лишь условно сам XML можно назвать языком разметки. В настоящее время создано большое число языков разметки, являющихся подмножествами XML, как пример укажем язык беспроводной

разметки WML (Wireless Markup Language), программная поддержка которого встроена во все мобильные телефоны и коммуникационные устройства, заявленные как WAP–совместимые.

XML, как и SGML, не является языком; он также представляет собой набор рекомендаций, на базе которых создаются другие языки. Точнее говоря, XML является конгломератом из трех отдельных спецификаций:

1. **XML (Extensible Markup Language)** — спецификация, определяющая базовый синтаксис XML;
2. **XSL (Extensible Style Language)** — спецификация, направленная на отделение визуального оформления страницы от ее содержимого за счет применения к документу стилей (style sheets), определяющих конкретные атрибуты форматирования;
3. **XLL (Extensible Linking Language)** — спецификация, определяющая представление ссылок на другие ресурсы.

На основе этой технологии обработка, поиск и представление информации переходят на совершенно иной уровень. Первоначальный веб был ориентирован на работу человека, но веб следующего поколения (семантический веб) должен в значительной мере опираться на машинную обработку информации, стандарту XML при этом отводится роль одной из ключевых технологий.

Синтаксис XML построен на основе тегов, но в отличие от HTML, в котором множество тегов фиксировано, в рамках XML пользователь создает собственное множество тегов и задает структурные отношения между ними.

Правильно построенные XML-документы

При создании документов необходимо учитывать правила, основная задача которых – в отделении данных и формата. Эти правила определены в рекомендации консорциума W3C [2,3] и состоят из следующих требований:

- документы XML должны начинаться с **объявления XML**, в котором определяется версия XML, например, **<?xml version="1.0">** (это объявление является инструкцией приложению, обрабатывающему документ, в частности, браузеру)**;** любому открывающему тегу должен соответствовать закрывающий тег;
- в XML учитывается регистр символов;
- значения атрибутов, используемых в определении тегов, должны заключаться в кавычки;
- в любом XML-документе должен содержаться один корневой эле-

мент для всего документа; элементы не должны перекрываться;

- вся информация, размещенная между открывающим и закрывающим тегами, рассматривается как данные, при этом учитываются все символы форматирования (пробелы, конец строки, табуляция);
- в XML имеются зарезервированные символы, которые используются как элементы синтаксиса; в тексте эти символы нужно заменять последовательностями других символов, называемых *объектами* (*entities*; в русскоязычной литературе часто используют термин *сущности*).

Зарезервированный символ	**Объект**
<	&lt;
>	&gt;
&	&amp;
“	&quot;
‘	&apos;

Если перечисленные правила выполнены, то говорят, что документ *правильно построен (well-formed)*, и только в этом случае он является *документом XML*. Такой документ будет правильно отображен в браузере, в ином случае будет выдано сообщение об ошибке. Для обработки документов XML используется программный модуль, называемый XML-процессором. Поддержка XML браузером зависит от того, включен ли в него XML-процессор и какие возможности этот процессор имеет. Например, браузеры Mozilla и Firefox поддерживают почти все технологии, основанные на XML.

Пример XML-документа

На простом примере покажем особенности синтаксиса XML. Опишем электронный журнал, причем включим в описание только название журнала и адрес, а также несколько статей с указанием их названия и авторов.

Пример 1. Простейшее описание электронного журнала. Листинг файла Example1.xml.

```
<?xml version="1.0" encoding="windows-1251"?
standalone=yes>
<!-- XML – пример -->
<journal>
<title>Lobachevsky&apos;Journal</title>
```

```
<contacts>
<address>Kazan State University</address>
<url>ljm.ksu.ru</url>
</contacts>
<articles>
<article ID="1">
<title>MathML and TeX</title>
<author>M. Malakhaltsev</author>
</article>
<article ID="2">
<title>MathML and RDF</title>
<author>E. Lipachev</author>
</article>
</articles>
</journal>
```

Все документы XML начинаются с пролога, в котором сообщается, что документ написан на XML, а также указывает, какая версия XML при этом использовалась. Атрибут **version** является обязательным и указывает на версию XML, применяемую для структурирования. Например, стандартный XML-файл начинается строкой:

```
<?xml version="1.0">
```

Получая документ, версию которого он не поддерживает, XML-процессор должен выдать соответствующие сообщения.

Пролог может содержать другие инструкции. Например, объявление можно расширить, указав, что документ является автономным:

```
<?xml version="1.0" standalone="yes">
```

Присваивание **yes** атрибуту **standalone** сообщает механизму обработки XML-кода о том, что документ не импортирует других файлов (например, DTD).

Атрибут **encoding** определяет кодировку документа, и его желательно указывать. Например, если документ содержит символы в одной из кодировок кириллицы (к примеру, CP1251, KOI8-R, CP866, UTF-8), в отсутствие этого атрибута XML-процессор выдаст сообщение об ошибке и прекратит обработку.

Во второй строке показано, как добавить комментарии, – правила те же, что и в HTML.

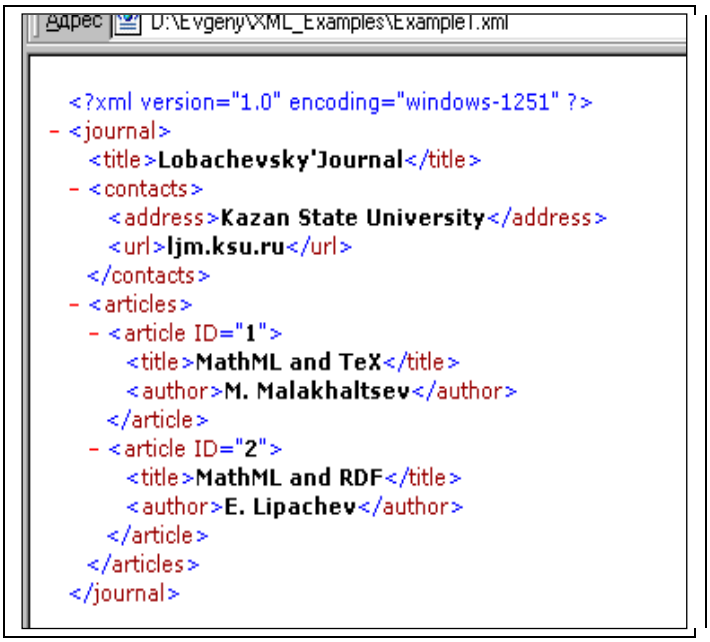
В последующих строках объявлены элементы. Элемент состоит из открывающего и закрывающего тегов, а также данных между этими тегами. Пустой элемент, например, **
</br>,** может записываться в виде **
. Имена не могут содержать пробелы, кроме того, имеет значение регистр символов. Одни элементы могут быть вложены в другие, как, например, элемент **<author> </author> из приведенного примера вложен в элемент **<article> </article>**, который в свою очередь вложен в элемент **<articles> </articles>**.

Непустые элементы должны содержать как открывающий, так и закрывающий тег. В элементах, которые логически не могут иметь закрывающего тега, используется альтернативная форма синтаксиса **<элемент />**.

Один из элементов документа должен содержать все остальные, этот элемент называют *корневым элементом* или *элементом документа*. В нашем примере таким элементом является **<journal> </journal>**. Таким образом, элементы документа должны образовывать древовидную структуру.

Теги XML могут обладать атрибутами. *Атрибуты* содержат дополнительную информацию о содержании, которая в дальнейшем используется при форматировании или обработке XML. Значения атрибутов присваиваются в формате «**имя=значение**», и, в отличие от HTML, атрибуты XML *должны* быть заключены в апострофы или кавычки. В приведенном примере элемент **<article> </article>** содержит атрибут **ID**, с помощью которого каждой статье присваивается числовой идентификатор.

Файл **Example1.xml** можно набрать в любом текстовом редакторе. При просмотре этого файла в каком-либо браузере, например, MS Internet Explorer будет выведено представление XML-документа в виде дерева



С помощью XML задаются правила, по которым данные из одного формата (не обязательно текстового) преобразуются в другой. Самым известным и применяемым языком этого типа есть XSL (XSLT). XSLT-схемы позволяют трансформировать документ XML в любой другой формат, например, в html-формат (см. раздел XSL-преобразования XML-документов).

Пространство имен

Пространство имён (namespace) – это некоторое множество имён, терминов, слов, означающих объекты реального мира или концепты. Имена в одном пространстве имён не могут иметь более одного значения. Пространство имён называют также контекстом, так как значение термина может изменяться в зависимости от того, в какое пространство имён он входит. Простым примером является имя человека. В пространстве имён «Семья» его достаточно, чтобы обозначить конкретного человека. В пространстве имён «Граждане Российской Федерации» этого не достаточно – нужно добавить дополнительную информацию – фамилию, адрес и т. п. Пространство имён в программировании – область определения переменных, типов, констант и т. п., которое используется для исключения конфликтов с другими именами вне данного блока.

Спецификация XML Namespaces (пространство имен XML) [16, 17] рассчитана на разрешение проблем, связанных с конфликтами имен. Поскольку множество тегов в XML не фиксировано, как, например, в HTML, пользователь имеет возможность выбирать имена тегов. При подготовке сложных

документов, например, использующих несколько DTD, некоторые теги, разные по своему назначению, могут получить одинаковые имена. Поясним эту ситуацию простым примером.

Пример 2.

```
<?xml version="1.0" encoding="windows-1251"?>
<!-- Конфликт имен title -->
<journal>
<title>Lobachevsky&apos;Journal</title>
<url>ljm.ksu.ru</url>
<articles>
  <article ID="1">
    <title>MathML and TeX</title>
  </article>
  <article ID="2">
    <title>MathML and RDF</title>
  </article>
</articles>
</journal>
```

В этом примере имя **title** используется для тега, содержащего название журнала, и тега с названием статьи. Заметим, что даже при наличии конфликта имен в документе он может без проблем отображаться в браузере, но более сложная обработка документа может оказаться невозможной.

Пример 3. Разрешение конфликта имен с помощью Namespaces.

```
<?xml version="1.0" encoding="windows-1251"?>
<journal xmlns:x="http://www.kcn.ru/one"
         xmlns:y="http://www.kcn.ru/two">
<x:title>Lobachevsky&apos;Journal</x:title>
<url>ljm.ksu.ru</url>
<articles>
  <article ID="1">
   <y:title>MathML and TeX</y:title>
  </article>
  <article ID="2">
    <y:title>MathML and RDF</y:title>
  </article>
</articles>
```

```
</journal>
```

Как правило, для обозначения пространства имен используются ссылки, образованные по спецификации URL, т. е. с каждым пространством имен связывают некоторый адрес. Механизм образования веб-адресов в силу своей природы должен обеспечить уникальность обозначения пространства имен. Поскольку используется лишь алгоритм образования адреса, нет необходимости в выборе адреса реально существующего веб-сайта, и поэтому адрес можно составить совершенно произвольно. Атрибут **xmlns** используется как ключевое слово XML для обозначения объявления пространства имен. Пространству имен назначается префикс пространства имен – он указывается после атрибута **xmlns** и отделяется двоеточием, а затем после знака равенства записывается адрес, однозначно идентифицирующий пространство имен. В приведенном примере образовано два пространства имен с префиксами **x** и **y**. Префиксы используются в тегах, указывая, к какому пространству имен относится данный тег.

Пример 4. Пространства имен **html** и **journal**.

```
<?xml version="1.0" encoding="windows-1251"?>
<!-- Пространства имен html и journal -->
<html xmlns:h="http://www.kcn.ru/three"
      xmlns:j="http://www.kcn.ru/four">
<h:head>
 <j:jrntitle>Lobachevsky&apos;Journal</j:jrntitle>
</h:head>
<h:body>
<j:jrntitle>Lobachevsky&apos;Journal</j:jrntitle>
<j:url>ljm.ksu.ru</j:url>
<j:articles>
<j:article ID="1">
<j:title>MathML and TeX</j:title>
</j:article>
<j:article ID="2">
<j:title>MathML and RDF</j:title>
</j:article>
</j:articles>
</h:body>
</html>
```

Действительные XML-документы и DTD

С каждым XML-документом можно связать DTD-объявления. Технология DTD (Document Type Definition) разработана как средство описания структуры документа и представляет собой набор инструкций для описания составных частей документа и порядка их следования. DTD определяет множество правил для тегов и атрибутов, допустимых в XML-документе, а также порядок и число вложений тегов, т. е. задается грамматика для определенного типа документов. Для единичного XML-документа создавать DTD не целесообразно, но при разработке некоторого стандарта документа, например, анкеты, желательно написать DTD, чтобы «отсечь» документы, не отвечающие стандарту. О существовании DTD надо помнить и в случае, если XML–документ включается в уже работающую систему без учета грамматики, определённой в DTD этой системы, Ваш документ не будет действительным и не пройдет обработку.

Документ XML имеет три уровня корректности: лексический, синтаксический и семантический. Если документ правильно построен, например, не нарушена вложенность тегов, все теги закрыты и т. д., то он лексически корректен. Синтаксическая корректность означает, что документ соответствует *заданным* правилам DTD. Таким образом, каждый набор DTD-правил определяет стандарт некоторого класса документов.

Инструкции DTD предназначены для XML-процессора (например, встроенного в браузер), анализирующего содержание документа перед его отображением. Если в документе есть ссылка на набор DTD-правил, и он синтаксически им удовлетворяет, то он называется *действительным (valid)* или *корректным*, используется также термин *допустимый* документ.

Инструкции DTD можно записать в отдельном файле или же включить непосредственно в XML–документ. Файл с таблицей DTD является обычным текстовым файлом и может быть создан в любом текстовом редакторе.

Атрибут имя_корневого_элемента соответствует имени корневого элемента в тегах, содержащих весь документ XML. В секции «прочих объявлений» находятся определения элементов, атрибутов и т. д.

Для рассматренного примера описания электронного журнала содержание DTD-файла может быть таким.

Пример 5. Файл example1.dtd.

```
<!-- DTD for journal XML -->
<!ELEMENT journal (jrntitle, contacts, articles)>
<!ELEMENT jrntitle (#PCDATA)>
```

```
<!ELEMENT contacts (address,url)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT articles (article)*>
<!ELEMENT article (title, author+)>
<!ATTLIST article
          id ID #REQUIRED>
<!ELEMENT title (#PCDATA)>
```

Для включения внешнего файла DTD в XML-документ используется тег

```
<!DOCTYPE имя_корня SYSTEM "имя_файла.dtd">
```

Имя, указанное за словом DOCTYPE, должно совпадать с именем корневого элемента.

Пример 6. Файл **example2.xml**.

```
<?xml version="1.0" encoding="windows-1251"?>
<!-- DTD for journal XML -->
<!DOCTYPE journal SYSTEM "example.dtd">
<journal>
<jrntitle>Lobachevsky&apos;Journal</jrntitle>
<contacts>
<address>Kazan State University</address>
<url>ljm.ksu.ru</url>
</contacts>
<articles>
<article ID="1">
<title>MathML and TeX</title>
<author>M. Malakhaltsev</author>
</article>
<article ID="2">
<title>MathML and RDF</title>
<author>E. Lipachev</author>
</article>
</articles>
</journal>
```

Определения DTD можно включать непосредственно в XML-файл. Инструкции DTD размещают в начале файла в контейнере

```
<!DOCTYPE имя_корня [
```

```
...
]>
```

Отметим, что блок DTD заканчивается символами **]>**.

Пример 7. Тот же пример, но с внутренним включением DTD.

```
<?xml version="1.0" encoding="windows-1251"?>
<!DOCTYPE journal [
<!ELEMENT journal (jrntitle, contacts, articles)>
<!ELEMENT jrntitle (#PCDATA)>
<!ELEMENT contacts (address,url)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT url (#PCDATA)>
<!ELEMENT articles (article)*>
<!ELEMENT article (title, author+)>
<!ATTLIST article
          id ID #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
]>
<!-- Конец DTD определений -->
<journal>
<jrntitle>Lobachevsky&apos;Journal</jrntitle>
<contacts>
<address>Kazan State University</address>
<url>ljm.ksu.ru</url>
</contacts>
<articles>
<article ID="1">
<title>MathML and TeX</title>
<author>M. Malakhaltsev</author>
</article>
<article ID="2">
<title>MathML and RDF</title>
<author>E. Lipachev</author>
</article>
</articles>
</journal>
```

Поясним на основе этого набора DTD-правил наиболее существенные де-

тали DTD. Как уже было отмечено, с помощью правила **<!DOCTYPE** указывается местонахождение DTD.

Для описания элементов используется правило

```
<!ELEMENT  имя_элемента содержание>
```

Для описания содержания используется довольно простая система обозначений. Круглые скобки в содержании означают, что элемент составной. Так, в приведенном примере элементы **journal, contacts, articles** и **article** составные. Запятая в скобках используется в качестве разделителя элементов. Вместо запятой можно использовать символ **“|”** (вертикальная черта), который разделяет варианты содержимого, если их у элемента несколько. Например,

```
<!ELEMENT phone (#PCDATA|EMPTY)>
```

определяет, что элемент **phone** может содержать символьные данные (номер телефона) или оставаться пустым (это указано с помощью ключевого слова **EMPTY**). Скобки задают также порядок появления элементов в документе. Например,

```
<!ELEMENT contacts (address,url)>
```

указывает, что элемент **contacts** содержит элемент **address,** и этот элемент должен быть первым, а также элемент **url**. Кроме того, можно задать количество повторений элемента, поставив один из знаков “+” (плюс), “*” (звёздочка) или “?” (знак вопроса).

Количество вложенных элементов можно задать несколькими способами. Полный список операторов элементов приведен в следующей таблице.

Операторы элементов

Признак	Значение
?	Ноль или ровно один экземпляр
*****	Ноль или несколько экземпляров
+	Один или несколько экземпляров
	Ровно один экземпляр
|	Один из элементов
,	Перечисление элементов

Эти знаки можно записывать не только после элементов, но и после скобок – в этом случае действие знака распространяется на множество элементов, заключенных в скобки. Знак “+” означает, что элемент или множество элементов встречаются в документе один или более раз. Знак “*” указывает, что в документе может быть любое число вхождений данного элемента, в частности, элемент может отсутствовать. Знак “?” используется в тех случаях, когда элемент или множество элементов встречаются не более одного раза. Например,

```
<!ELEMENT article (title, author+)>
```

указывает, что элемент **article** содержит элемент **title** и он должен быть первым, а также один или более элементов **author**.

Если элемент содержит символьные данные, то при его описании используется содержание **(#PCDATA).** Ключевое слово **EMPTY** в содержании документа указывает, что это пустой элемент. Как примеры пустых элементов приведём теги **
** и **<hr>** в HTML. Ключевое слово **ANY** в содержании означает, что элемент может содержать как символьные данные, так и другие элементы.

Если у элемента есть атрибуты, как, например, атрибут **ID** у элемента **article**, то необходимо использовать инструкцию

```
<!ATTLIST        имя_элемента
                 имя_атрибута  тип        значение_по_умолчанию

                 имя_атрибута  тип        значение_по_умолчанию>
```

В этом объявлении необходимо перечислить все атрибуты, которые могут использоваться с данным элементом. Типы атрибутов можно разбить на три группы: строковые атрибуты, маркированные и перечислимые. Строковые атрибуты описываются ключевым словом **CDATA,** и в документе такие атрибуты могут содержать любые символьные данные. Маркированные атрибуты включают в себя предопределенные атрибуты **ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN** и **NMTOKENS**. Эти атрибуты предназначены для конкретного использования, например, атрибут **ID** задаёт уникальный идентификатор для элемента в документе. В нашем примере с помощью **ID** каждой статье присваивался числовой идентификатор. Подробности о назначении этих атрибутов можно найти в [4, 5]. Если атрибут содержит список значений, то атрибут называется перечислимым. Например,

```
<!ELEMENT volume (#PCDATA)>
<!ATTLIST    volume
        month (1|2|3|4|5|6|7|8|9|10|11|12) #REQUIRED>
```

Параметр **#REQUIRED** определяет, что данный атрибут является обязательным. Есть ещё два параметра: **#IMPLIED**, указывающий, что атрибут является необязательным, и **#FIXED**, требующий, чтобы только указанное в атрибуте значение использовалось в документе.

Атрибуты ENTITY и ENTITIES

Данные в документах XML не всегда являются текстовыми — документ может содержать и двоичную информацию (например, графику). На такие данные можно ссылаться при помощи атрибута **entity**. Например, в описании элемента **description** можно указать атрибут **recipePicture** с графическим изображением:

```
<!ATTLIST description recipePicture ENTITY #IMPLIED>
```

Также можно объявить сразу несколько сущностей, заменив **ENTITY** на **ENTITIES**. Значения разделяются пробелами.

Ссылки на сущности

Концепция *сущности* (entity) упрощает сопровождение документа, обеспечивая возможность ссылки на некоторое содержание по ключевым словам. Ключевое слово может относиться как к простейшему фрагменту вроде расширения аббревиатуры, так и к совершенно новому фрагменту кода XML. Сущности удобны тем, что они могут многократно использоваться в документах XML. При последующей обработке документа все ссылки на сущность заменяются конкретным содержанием, указанным при объявлении сущности. Объявление сущности включается в DTD документа XML.

Чтобы сослаться на некоторую сущность в документе HTML, следует указать ее имя с префиксом «амперсанд» (&) и суффиксом «точка с запятой» (;). Допустим, вы объявили сущность с информацией об авторских правах. После этого на данную сущность можно ссылаться следующим образом:

```
&Copyright:
```

При этом строка документа XML может выглядеть так:

```
<footer>
...прочие данные колонтитула...
&Copyright:
</footer>
```

(из Php и XML – 12)

ссылка на сущность используется в качестве замены для другого фрагмента содержания. В процессе обработки документа XML все вхождения сущности заменяются содержанием, которое она представляет. Существует два вида сущностей: внутренние и внешние.

Внутренние сущности

Внутренние сущности напоминают строковые переменные, связывающие имя с фрагментом текста. Например, если вы хотите определить имя для ссылки на информацию об авторских правах, можно объявить сущность следующего вида:

```
<!ENTITY Copyright "Copyright 2000 YourCompanyName. All
Rights Reserved.">
```

В процессе обработки документа все экземпляры **&Copyright** заменяются текстом **«Copyright 2000 YourCompanyName. All Rights Reserved»**. Весь код XML в заменяющем тексте обрабатывается так, словно он присутствовал в исходном документе.

Внешние сущности

Внешние сущности используются для ссылок на содержание, находящееся в другом файле. Сущности этого типа могут содержать текстовую информацию, но также могут ссылаться и на двоичные данные (например, графику). Возвращаясь к предыдущему примеру, допустим, что вы решили сохранить информацию об авторских правах в отдельном файле, чтобы упростить ее редактирование в будущем. Ссылка на созданный файл выглядит следующим образом:

```
<!ENTITY Copyright SYSTEM
"http://yoursite.com/administer/copyright.xml">
```

При последующей обработке документа XML все ссылки **&Copyright** заменяются содержимым документа **copyright.xml**. Весь код XML в заменяющем тексте обрабатывается так, словно он присутствовал в исходном документе.

Внешние сущности также удобно использовать для ссылок на графические изображения. Например, если вы хотите включить в документ XML графический логотип, создайте внешнюю сущность:

```
<!ENTITY food_picture SYSTEM
http://yoursite.com/food/logo.gif>
```

Как и в предыдущем примере, все ссылки **&food_picture** заменяются графическим изображением, на которое указывает ссылка. Поскольку данные

являются двоичными, а не текстовыми, они не интерпретируются.

XML схема

Помимо использования DTD, для описания структуры документа можно использовать XML-схемы, которые приняты консорциумом W3C в качестве рекомендации [6, 7].

XML-схема определяет:

- набор элементов документа
- набор атрибутов документа
- элементы, которые могут иметь дочерние элементы
- порядок дочерних элементов
- число дочерних элементов
- является ли элемент пустым или он может содержать текст
- типы данных элементов и атрибутов
- значения по умолчанию и фиксированные значения элементов и атрибутов

Отметим наиболее существенные свойства XML-схем.

XML-схемы поддерживают типизацию данных. Это позволяет

- описывать разрешенное содержание документа
- проверять правильность данных
- работать с данными из баз данных
- задавать ограничения на данные
- задавать форматы данных
- преобразовывать данные различных типов

XML-схемы пишутся на XML. Благодаря этому:

- Для редактирования схем можно использовать XML-редактор
- Для анализирования схем можно использовать XML-парсер
- Можно работать с XML-схемами посредством XML DOM
- Можно преобразовывать схемы посредством XSLT

XML-схемы расширяемы (также, как и XML), поскольку они пишутся на XML. Благодаря расширяемости каждой конкретной схемы можно:

- Встраивать одни схемы в другие
- Создавать свои собственные типы данных, производя их из стандартных типов
- Ссылаться из документа на несколько схем

Приведем простейший пример применения XML-схемы. Допустим, что

некоторый XML-файл обрабатывается двумя различными программными модулями. При этом конечно необходимо, чтобы они одинаково интерпретировали данные, содержащиеся в XML-файле. С помощью XML-схем можно описать каким образом программа должна интерпретировать данные XML-файла. Например, дата 2008-09-05 может быть интерпретирована как 9 мая или как 5 сентября в зависимости от принятого стандарта даты. XML-элемент, содержащий описание типа данных, например:

```
<date type="date">2008-09-05</date>
```

обеспечит верную трактовку содержания, поскольку тип данных **date** требует использования формата **CCYY-MM-DD.**

Теперь покажем, как описать структуру данных в XML-документе **example2.xml** (пример 6) с помощью XML-схемы (отметим, что в предыдущем разделе мы решали эту задачу с помощью DTD).

Вначале возьмем простейший XML-файл

Пример 8. Journal1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<journal>
</journal>
```

XML-схема для описания этого файла имеет вид:

Пример 9. Schema_1.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="journal">
  </xsd:element>
</xsd:schema>
```

Поясним этот пример. Как уже отмечалось XML-схема является XML-документом, в частности, имеет корневой элемент **<xsd:schema>.** Этот XML-документ состоит из элементов, имена которых принадлежат пространству имен (см. раздел Пространство имен), идентифицированное адресом "http://www.w3.org/2001/XMLSchema". Объявление пространства имен содержится в элементе schema:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Все XML-документы построены из элементов. Для определения элементов

в XML-схеме используется элемент **xsd:element**. Его атрибут **name** задает имя соответствующего элемента. Простейшая схема **Schema_1.xsd** позволяет организовать проверку XML-документов. Например, следующий XML-документ не пройдет проверку на соответствие схеме **Schema_1.xsd**:

Пример 10. Journal_1.xml

```
<?xml version="1.0" encoding="utf-8"?>
<paper>
</paper>
```

так как элемент в этом файле имеет имя **paper**, а не **journal**. Естественно любой XML-документ, дерево которого состоит из двух и более элементов, также не пройдет проверку.

Теперь рассмотрим более сложный XML-документ

Пример 11. Journal2.xml

```
<?xml version="1.0" encoding="utf-8"?>
<journal>
<jrntitle>Lobachevsky&amp;apos;Journal</jrntitle>
<contacts>
<address>Kazan State University</address>
<url>ljm.ksu.ru</url>
</contacts>
</journal>
```

Здесь элемент **journal** содержит подэлементы **jrntitle** и **contacts**, причем подэлемент **contacts** сам содержит подэлементы **address** и **url**.

Элементы, которые не содержат подэлементов и атрибутов называются элементами простого типа. Таким образом, **jrntitle**, **address** и **url** – элементы простого типа. Элементы, которые содержат подэлементы и аттрибуты называются элементами сложного типа. В нашем примере – это элементы **journal** и **contacts**.

Описание элементов простого типа

Напомним, что для определения элементов в XML –схеме используется элемент **xsd:element** и его атрибут **name** имя элемента. Элемент **<xsd:element>** имеет ряд других атрибутов, позволяющих проверять

значение соответствующих элементов XML-документов. (подробности см. в [6, 7]). Атрибут **type** позволяет указать тип данных. Значениями элементов **jrntitle**, **address** являются строки, а значением элемента **url** является URI (например, данные вида **http://www.example.com/**, **http://www.example.com/doc.html#ID5**). Поэтому, эти элементы можно объявить так:

```
<xsd:element name="jrntitle" type="xsd:string"/>
<xsd:element name="address" type="xsd:string"/>
<xsd:element name="url" type="xsd:anyURI"/>
```

Если XML-файл будет содержать элемент **<url>1111</url>**, то XML-процессор выдаст сообщение об ошибке, так как значение этого элемента не удовлетворяет типу **anyURI**.

В рекомендации XML-схемы предусмотрено 42 простых типа данных, включая **string, int, date, decimal, boolean, timeDuration, uriReference** (см. [6, 7]).

Другим примером ограничения, налагаемого на элемент, является ограничение числа вхождений данного элемента (в соответствующий фрагмент XML-дерева) с помощью атрибутов **maxOccurs** и **minOccurs**. В нашем примере, название журнала может быть только одно, а веб-сайтов несколько (но, допустим, не более трех). Соответствующие объявления имеют вид:

```
<xsd:element name="jrntitle" maxOccurs="1"
type="xsd:string"/>
<xsd:element name="address" maxOccurs="1"
type="xsd:string"/>
<xsd:element name="url" maxOccurs="3" type="xsd:anyURI"/>
```

Теперь, если в XML –документе два раза встретится элемент **jrntitle**, то XML– процессор выдаст сообщение об ошибке. Отметим, что по умолчанию значения атрибутов **maxOccurs** и **minOccurs** установлено в **1**, то есть если они отстутствуют, то элемент должен входить в соответствующий фрагмент XML-дерева ровно один раз.

В дополнение к предопределенным простым типам можно создать собственные простые типы. Они могут описывать текстовые данные в определенном формате (например, номер телефона), числовые данные, данные из предопределенного списка и т.д. Приведем несколько примеров.

Предположим, что нам надо описать номер ежемесячного журнала, который может изменяться от 1 до 12. Фрагмент XML-схемы, определяющий тип **issuenumber**, имеет вид:

```
<xsd:simpleType name="issuenumber">
<xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="12"/>
</xsd:restriction>
</xsd:simpleType>
```

а соответствующий описание элемента будет

```
<xsd:element name="issue" type="issuenumber">
```

Допустим, что в описании статьи надо указать тематику статьи (алгебра, геометрия, математический анализ и т.п.). Тогда нам потребуется тип **fieldtype**, который принимает заранее определенные значения, скажем, **algebra, geometry, calculus**.

```
<xsd:simpleType name="field">
<xsd:restriction  base="xsd:string">
      <xsd:enumeration value="algebra" />
      <xsd:enumeration value="geometry" />
      <xsd:enumeration value="calculus" />
   </xsd:restriction>
</xsd:simpleType>
```

Отметим, что никакого упорядочивания значений (в данном случае **algebra**, **geometry**, **calculus**) это описание не задает.

Описание элементов сложного типа

В примере 11 элементы **journal** и **contacts** являются элементами сложного типа, так как они содержат подэлементы. Элементы сложного типа описываются элементом XML–схемы **<xsd:complexType>**, а последовательность подэлементов с помощью элемента XML-схемы **<xsd:sequence>**. Для нашего примера описание элемента **contacts** можно задать так:

```
<xsd:element name="contacts">
  <xsd:sequence>
```

```
    <xsd:element name="address" maxOccurs="1"
type="xsd:string"/>
    <xsd:element name="url" maxOccurs="3"
type="xsd:anyURI"/>
  </xsd:sequence>
</xsd:element>
```

Теперь понятно как записать XML– схему для примера 11:

Пример 12. Schema_2.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="journal">
  <xsd:complexType>
   <xsd:sequence>
     <xsd:element name="jrntitle"/>
     <xsd:element name="contacts">
       <xsd:complexType>
         <xsd:sequence>
          <xsd:element name="address" maxOccurs="1"
type="xsd:string"/>
          <xsd:element name="url" maxOccurs="3"
type="xsd:anyURI"/>
        </xsd:sequence>
       </xsd:complexType>
     </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

В XML–схеме можно создавать собственные сложные типы. Например, можно создать сложный тип **contactType**:

```
<xsd:complexType name="contactType">
  <xsd:sequence>
    <xsd:element name="address" maxOccurs="1"
type="xsd:string"/>
    <xsd:element name="url" maxOccurs="3"
```

```
type="xsd:anyURI"/>
  </xsd:sequence>
</xsd:complexType>
```

и записать XML-схему для примера 11 в виде

Пример 13. Schema_2_1.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="journal">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="jrntitle" type="xsd:string"/>
      <xsd:element name="contacts" type="contactType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:complexType name="contactType">
  <xsd:sequence>
    <xsd:element name="address" type="xsd:string"/>
    <xsd:element name="url" type="xsd:anyURI"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Теперь можно привести схему для XML–документа из примера 6 (ссылку на DTD мы убрали):

Пример 14. Journal3.xml

```
<?xml version="1.0" encoding="utf-8"?>
<journal>
<jrntitle>Lobachevsky&amp;apos;Journal</jrntitle>
<contacts>
<address>Kazan State University</address>
<url>ljm.ksu.ru</url>
</contacts>
<articles>
<article ID="1">
<title>MathML and TeX</title>
```

```
<author>M. Malakhaltsev</author>
</article>
<article ID="2">
<title>MathML and RDF</title>
<author>E. Lipachev</author>
</article>
</articles>
</journal>
```

Пример 15. Schema_3.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="journal">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="jrntitle" type="xsd:string"/>
      <xsd:element name="contacts" type="contactType"/>
      <xsd:element name="articles" type="articlesType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:complexType name="contactType">
  <xsd:sequence>
    <xsd:element name="address" type="xsd:string"/>
    <xsd:element name="url" type="xsd:anyURI"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="articlesType">
  <xsd:sequence>
    <xsd:element name="article" maxOccurs="unbounded">
      <xsd:complexType>
         <xsd:sequence>
           <xsd:element name="title" maxOccurs="1"
type="xsd:string"/>
           <xsd:element name="author" maxOccurs="15"
type="xsd:string"/>
         </xsd:sequence>
```

```
          <xsd:attribute name="ID" type="xsd:int"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Конструкции схемы были объяснены выше, осталось лишь указать, что элемент **article** в XML-документе **Journal3.xml** имеет атрибут ID, который принимает целые значения. Соответствующее объявление в XML–схеме **Schema_3.xsd** имеет вид:

```
<xsd:attribute name="ID" type="xsd:int"/>
```

причем оно помещено после объявления подэлементов элемента **article**. Отметим, что элемент, имеющий атрибут, необходимо имеет сложный тип, независимо от того есть у него подэлементы или нет.

Приведем пример файла, не удовлетворяющего XML–схеме **Schema_3.xsd**.

Пример 16. **Journal3_1.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<journal>
<jrntitle>Lobachevsky&apos;Journal</jrntitle>
<contacts>
<address>Kazan State University</address>
<address>Saratov State University</address>
<url>ljm.ksu.ru</url>
</contacts>
<articles>
<article ID="A">
<title>MathML and TeX</title>
<title>TeX and MathML</title>
<author>M. Malakhaltsev</author>
</article>
<article ID="2">
<title>MathML and RDF</title>
<author>E. Lipachev</author>
</article>
```

```
</articles>
</journal>
```

XML-процессор выдает сообщение о следующих ошибках:

- Два раза встречается элемент **address**. Хотя в XML-схеме не установлен атрибут **MaxOccurs**, по умолчанию элемент **address** может входить точно один раз.
- Атрибут **ID** установлен в значение **A**. В XML–схеме его тип определен как целое число.
- Элемент **title** два раза входит в элемент **article**. В XML-схеме с помощью атрибута **MaxOccurs** указано, что он может встречаться не более одного раза.

Во втором элементе **article** вначале идет элемент **title**, а затем **author**. В XML–схеме описан другой порядок подэлементов элемента **article**: вначале элемент **author**, а затем – **title**.

Этот пример иллюстрирует какие элементы структуры XML– документы могут быть регламентированы с помощью XML– схемы.

Отметим, что разоаботаны программные средства, автоматически генерирующие XML–схему по данному XML-документу. Например, с помощью расширения **XML Developer** браузера **Firefox** можно получить следующую XML–схему для документа **Journal3.xml**:

Пример 17. Schema_4.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="address">
    <xs:complexType mixed="true" />
  </xs:element>
  <xs:element name="article">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title" />
        <xs:element ref="author" />
      </xs:sequence>
      <xs:attribute name="ID" type="xs:integer" use =
```

```
"required" />
    </xs:complexType>
  </xs:element>
  <xs:element name="articles">
    <xs:complexType>
      <xs:sequence>
       <xs:element ref="article" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="author">
    <xs:complexType mixed="true" />
  </xs:element>
  <xs:element name="contacts">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="address" />
        <xs:element ref="url" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="journal">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="jrntitle" />
        <xs:element ref="contacts" />
        <xs:element ref="articles" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="jrntitle">
    <xs:complexType mixed="true" />
  </xs:element>
  <xs:element name="title">
    <xs:complexType mixed="true" />
  </xs:element>
  <xs:element name="url">
```

```
    <xs:complexType mixed="true" />
  </xs:element>
</xs:schema>
```

В этой XML–схеме используется ссылка на элементы, заданная атрибутом **ref**. Значение атрибута ref должно быть глобальным элементом, который был объявлен в элементе schema, а не как часть определения комплексного типа. Например, элемент article содержит подэлементы title, article и атрибут ID. С помощью атрибута ref этот элемент описывается с помощью следующего фрагмента XML–схемы:

```
<xs:element name="article">
<xs:complexType>
<xs:sequence>
<xs:element ref="title" />
<xs:element ref="author" />
</xs:sequence>
<xs:attribute name="ID" type="xs:integer" use="required"
/>
</xs:complexType>
```

Здесь идет ссылка на элементы **title** и **author**, которые определены в той же XML–схеме:

```
<xs:element name="title">
  <xs:complexType mixed="true" />
  </xs:element>
```

и

```
xs:element name="author">
  <xs:complexType mixed="true" />
</xs:element>
```

Стоит отметить, что модуль программы XML Developer, который сгенерировал данную XML–схему автоматически, задал данные элементы как элементы сложного типа, допускающие любые сочетания элементов и текста (это задано значением **true** атрибута **mixed**). Однако, в нашем примере эти элементы имеют тип **string**, поэтому можно изменить определение этих элементов на определение простых элементов:

```
<xs:element name="title" type="xs:string"/>
```

и

```
<xs:element name="author" type="xs:string"/>
```

При таком изменении XML–схемы документ **journal.xml** останется правильным.

Ссылка на XML-схему в XML-документе

Вначале отметим, что выбор XML–схемы для проверки XML–документа полностью зависит от программы, использующей данный XML–документ. Например, программа может проверять XML–документ, не содержащий ссылку на XML–схему, на соответствие некоторой XML–схеме или игнорировать ссылку на XML–схему и не производить проверку. Браузер Firefox не сообщает об ошибке в XML–документе, содержащем ссылку на XML–схему, которой этот документ не удовлетворяет. С другой стороны, упоминавшееся расширение XML Developer осуществляет проверку XML-документа, не содержащего ссылку на XML–схему, при этом XML-схема указывается пользователем.

Тем не менее, рекомендацией W3C [7] предусмотрен механизм ссылки на XML–схему в XML–документе, то есть в XML–документе можно указать какую XML–схему следует использовать для данного документа. Для этого используются следующие атрибуты корневого элемента : **SchemaLocation, noNamespaceSchemaLocation**.

Атрибут **SchemaLocation** используется для ссылки на XML–схемы, которые определяются в целевом пространстве имен.

```
<journal xmlns="http://www.example.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.journal.org
schema.xsd">
```

Этот атрибут может содержать набор пространств имен и указаний на расположение схемы, разделенных пробелом.

```
<journal xmlns="http://www.journal.org"
    xmlns:ns2="http://www.journal2.com"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.journal.org
example.xsd
                        http://www.journal2.com
example2.xsd">
```

Атрибут **noNamespaceSchemaLocation** используется для того, чтобы ссылаться на XML–схемы, которые не определены в целевом пространстве имен.

```
<journal xmlns="http://www.example.org"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="example.xsd">
```

Атрибут **noNamespaceSchemaLocation** может содержать только ссылку на файл схемы и не может содержать **uri**, идентифицирующий пространство имен.

Еще раз отметим, что значения атрибутов **schemaLocation** и **noNamespaceSchemaLocation** не являются обязательными для XML–процессора.

Теперь возьмем пример **Journal3.xml** и переделаем его так, чтобы он ссылался на XML–схему **Schema_3.xsd**.

Пример 18. **Journal4.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<journal xmlns:xsi='http://www.w3.org/2001/XMLSchema-
instance'
  xsi:noNamespaceSchemaLocation='Schema_3.xsd'>
<jrntitle>Lobachevsky&apos;Journal</jrntitle>
<contacts>
<address>Kazan State University</address>
<url>ljm.ksu.ru</url>
</contacts>
<articles>
<article ID="1">
<title>MathML and TeX</title>
<author>M. Malakhaltsev</author>
</article>
<article ID="2">
<title>MathML and RDF</title>
<author>E. Lipachev</author>
</article>
</articles>
```

```
</journal>
```

Изменение состоит в том, что в элементе **journal** объявлено пространство имен с указанием XML–схемы.

```
<journal xmlns:xsi='http://www.w3.org/2001/XMLSchema-
instance'  xsi:noNamespaceSchemaLocation='Schema_3.xsd'>
```

В настоящее время в интернете можно найти много информации о различных вариантах ссылки на XML-схемы в XML-документах (например, см.

XSL-преобразование XML-документов

Как видно из приведенных примеров, язык XML совершенно не «заботится» о представлении информации, поскольку он разработан для других целей. Для отображения информации в «привычном» виде, как это делалось с помощью языка HTML, необходимо использовать другие технологии.

Универсальным механизмом управления отображением XML-документов является расширяемый язык таблиц стилей XSL (eXtensible Stylesheet Language). С помощью XSL можно трансформировать XML документ в любой формат, например HTML, WML, RTF. PDF, SQL и, в частности, в XML. На языке XSL можно описать, как будет оформлен итоговый документ, где и как должны располагаться данные.

Спецификация XSL [9, 10] состоит из двух частей: XSL-T (XSL Transformations) – язык для преобразования XML-документов и XSL-FO (XSL Formatting Objects) – язык для вёрстки XML. Язык XSLT используется для преобразования XML-документа в документ, предназначенный для отображения браузером. Технически это осуществляется с помощью применения к исходному XML-документу *таблицы стилей* XSLT, состоящей из набора *шаблонов.* XSL-FO является языком разметки, который предназначен для описания внешнего вида (макета) документа. Мы не будем здесь рассматривать XSL-FO, отметим только, что схематически использование этого языка можно описать следующим образом. Исходным является документ на любом языке, являющимся подмножеством XML, например, XHTML или DocBook. Затем применяется XSLT-преобразование, подходящее к необходимому типу документа, и получается XSL-FO-документ. Этот документ передается приложению (FO-процессору), который конвертирует XSL-FO-документ в какой-либо читаемый и/или печатаемый формат, например, PDF, PS, RTF, или выводит его на мони-

тор.

Перейдем к описанию XSLT и начнем с примеров. Создадим таблицу стилей для отображения файла **example2.xml**, приведенного в примере 6, и сохраним ее в XSL-файле.

Пример 19. XSL-файл (one.xsl) для XML-файла из примера 6.

```
<xsl:stylesheet version="1.0"
            xmlns:xsl="
http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<h2>
 <xsl:value-of select="//jrntitle"/>
</h2>
<hr/>
<P>
<I><xsl:value-of select="//contacts/address"/></I>
</P>
<P>
<xsl:value-of select="//contacts/url"/>
</P>
</xsl:template>
</xsl:stylesheet>
Теперь в файл example2.xml добавим  следующую ссылку на
файл one.xsl:
<?xml-stylesheet type='text/xsl' href='one.xsl'?>
```

Пример 20. Листинг файла example2.xml. Добавлена ссылка на XSL-файл.

```
<?xml version="1.0" encoding="windows-1251"?>
<?xml-stylesheet type='text/xsl' href='one.xsl'?>
<journal>
<jrntitle>Lobachevsky&amp;apos;Journal</jrntitle>
<contacts>
<address>Kazan State University</address>
<url>ljm.ksu.ru</url>
</contacts>
<articles>
<article ID="1">
<title>MathML and TeX</title>
<author>M. Malakhaltsev</author>
```

```
</article>
<article ID="2">
<title>MathML and RDF</title>
<author>E. Lipachev</author>
</article>
</articles>
</journal>
```

Если мы теперь откроем файл **example2.xml** в браузере Internet Explorer, то получим «привычную» форму представления документа:

На этом примере поясним принципы работы XSL-преобразований. Преобразование исходного XML-документа (в нашем случае, **example2.xml**) с помощью таблицы стилей (в нашем случае, **one.xsl**) осуществляется программным модулем, который называется XSLT-процессор. Большинство современных браузеров (например, Internet Explorer, Mozilla, Firefox) имеют встроенные XSLT-процессоры, позволяющие «на лету» осуществлять XSL-преобразование, то есть производить XSL-преобразование исходного XML-документа в соответствии с заданной таблицей стилей и передавать полученный документ другому модулю браузера для вывода на экран. В результате пользователь, открывая в браузере XML-документ, содержащий ссылку на таблицу стилей, видит документ, определяемый этой таблицей стилей. Таким образом, один и тот же XML-документ может быть показан в браузере различными способами (см. примеры ниже).

Отметим, что отображение информации в браузере – только одна из задач, для решения которых может быть применено XSL-преобразование XML-файлов. XSLT является мощным средством обработки информации, структурированной с помощью XML. Поэтому XSLT-процессоры включены в интерпретаторы языков, применяемых в веб-программировании (например, PHP, Java,

JavaScript). В Linux XSLT-процессор включен в библиотеку libxml и может быть вызван в консоли командой xsltproc.

Кратко опишем алгоритм работы XSL-преобразования (детальное описание см. в [18], [19]).

Имеются исходный XML-файл (исходное дерево) и таблица стилей, содержащая набор правил шаблона. Правило шаблона состоит из двух частей: это образец, который сопоставляется с узлами в исходном дереве, и шаблон, который может быть обработан для формирования фрагмента в конечном дереве. В ходе XSL-преобразования образец сравнивается с элементами исходного дерева, а шаблон используется для создания частей конечного дерева. Отметим, что структура конечного дерева может полностью отличаться от структуры исходного дерева. В ходе построения конечного дерева элементы исходного дерева могут подвергаться фильтрации и переупорядочению, также может быть добавлена новая структура.

Чтобы в конечном дереве построить фрагмент, обрабатывается перечень исходных узлов. Обработка узла осуществляется путем нахождения всех правил шаблона, чей образец соответствуют этому узлу, выбора среди них самого «лучшего» (правила, по которым выбирается лучший шаблон, см. в [9, 10]) и последующего применения к узлу инструкций выбранного шаблона. Этот процесс продолжается до тех пор, пока для обработки можно найти новые исходные узлы.

Посмотрим, как этот алгоритм работает в нашем примере. Начнем с таблицы стилей **one.xsl**. Таблица стилей является XML-файлом с корневым элементом

```
<xsl:stylesheet version="1.0"
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
```

Его атрибуты – номер версии и ссылка на пространство имен **(xmlns:xsl)** – являются обязательными, а ссылка на **http://www.w3.org/1999/XSL/Transform** – стандартной.

Как было указано, таблица стилей состоит из правил шаблона. Эти правила задаются элементами **<xsl:template>**. При этом образец для выбора узлов XML-дерева, к которому будет применяться шаблон, задается атрибутом **match**, а шаблон является значением этого элемента. Таблица стилей |**one.xsl** содержит одно правило шаблона

```
<xsl:template match="/">
<h2>
```

```
 <xsl:value-of select="//jrntitle"/>
</h2>
<hr/>
<P>
<I><xsl:value-of select="//contacts/address"/></I>
</P>
<P>
<xsl:value-of select="//contacts/url"/>
</P>
</xsl:template>
```

Образцом в этом правиле является "/", а самим шаблоном – значение элемента **<xsl:template>**, то есть

```
<h2>
 <xsl:value-of select="//jrntitle"/>
</h2>
<hr/>
<P>
<I><xsl:value-of select="//contacts/address"/></I>
</P>
<P>
<xsl:value-of select="//contacts/url"/>
</P>
```

Образец шаблона является выражением, написанным на языке Xpath, который предназначен для адресации узлов (и подмножеств узлов) XML-дерева. В данном руководстве нет возможности подробно описать синтаксис и применения языка Xpath, поэтому мы вынуждены ограничиться тем, что приведем несколько простых примеров выражений. Подробно об этом языке можно узнать из рекомендаций W3C [11] (также см. руководства [12], [13], [14]). Отметим, что в 2007 году завершилась разработка версии 2.0 языка XPath, который теперь является составной частью языка XQuery.

Примеры выражений на языке Xpath:

Выражение	Результат
Paper	Выбирает всех прямых потомков узла **paper**
/paper	Выбирает элемент **paper**, являющийся прямым

	потомком корневого узла **Замечание:** Если выражение начинается с /, то оно описывает абсолютный путь к элементу
paper/title	Выбирает все узлы **title**, являющиеся прямыми потомками **paper**
//paper	Выбирает все узлы **paper**, независимо от того, где этот элемент находится в дереве
journal//author	Выбирает все элементы **author** являющиеся потомками элементов **journal**
//@lang	Выбирает все атрибуты с именем **lang**

В нашем примере значение **"/"** атрибута **match** определяет, что шаблон будет применяться ко всем элементам XML-дерева.

Теперь перейдем к инструкциям шаблона. Вначале идет текстовый узел **<h2>.** Каждый текстовый узел в шаблоне создаст в конечном дереве текстовый узел с тем же самым строковым значением, то есть в результирующее дерево будет записано **<h2>**.

Далее идет элемент **<xsl:value-of select="//jrntitle"/>.** Этот элемент позволяет получить значение элемента исходного XML-документа, определяемого значением атрибута **select**, то есть **"//jrntitle"** (отметим, что значениями этого атрибута являются XPath-выражения). В нашем примере в результирующее дерево будет записано значение элемента

<jrntitle> Lobachevsky'Journal</jrntitle>

исходного XML-дерева из файла **Example2.xml**, то есть

Lobachevsky'Journal

Применение остальных инструкций шаблона происходит аналогично и в результате получается документ

```
<?xml version="1.0"?>
<h2>Lobachevsky'Journal</h2>
<hr/>
<P><I>Kazan State University</I></P>
<P>ljm.ksu.ru</P>
```

который отображается браузером вышеуказанным способом.

Теперь покажем как, изменив стилевую таблицу, можно из того же самого

файла **Example2.xml** получить файл, содержащий список авторов журнала. Возьмем вначале стилевую таблицу

```
<xsl:stylesheet version="1.0"
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="//article">
<xsl:value-of select="author"/>
</xsl:template>
</xsl:stylesheet>
```

Эта таблица содержит одно правило шаблона, образцом является **//article**, поэтому можно предположить, что результирующий документ будет представлять собой список авторов:

```
M. Malakhaltsev
E. Lipachev
```

На самом деле, мы получим следующий документ:

```
<?xml version="1.0"?>
Lobachevsky'Journal
Kazan State University
ljm.ksu.ru
M. Malakhaltsev
E. Lipachev
```

Дело в том, что для узлов, которые не удовлетворяют ни одному образцу, применяется шаблон по умолчанию, который в данном случае просто выводит содержимое соответствующих элементов исходного XML-файла. Если же применить следующую таблицу стилей

```
<xsl:stylesheet version="1.0"
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="/">
</xsl:template>
<xsl:template match="//article">
<xsl:value-of select="author"/>
</xsl:template>
</xsl:stylesheet>
```

которая содержит два правила шаблона (шаблон первого правила является пустым для того, чтобы подавить вывод ненужной информации), для элементов **article** возникнет конфликт правил шаблона (этот элемент удовлетворяет двум образцам). Этот конфликт разрешается в пользу первого правила шаблона

– «ничего не делать» (подробно о разрешении конфликтов правил шаблона и о том, как выйти из этой ситуации, см. [11], [13]).

Чтобы все же получить список авторов, можно попытаться применить следующее правило шаблона:

```
 <xsl:stylesheet version="1.0"
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
  <xsl:template match="/">
  <xsl:value-of select="//article/author"/>
 </xsl:template>
  </xsl:stylesheet>
```

Однако результатом будет документ, содержащий только первого автора

```
M. Malakhaltsev
```

Причина в том, что к корневому элементу применяется шаблон, который указывает, что надо вывести содержимое элемента, определяемого выражением **//article/author**. Однако таких элементов в данном случае два и нет указаний, какой из них надо выбрать. Поэтому выбирается первый.

Для того чтобы вывести список всех авторов, применяется элемент **<xsl:for-each>**, дающий инструкцию осуществить обработку каждого элемента из набора элементов, определенного атрибутом **select**, значением которого является XPath-выражение. В данном случае стилевая таблица будет иметь вид:

```
<xsl:stylesheet version="1.0"
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="/">
<xsl:for-each select="//article">
<p>
<xsl:value-of select="author"/>
</p>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

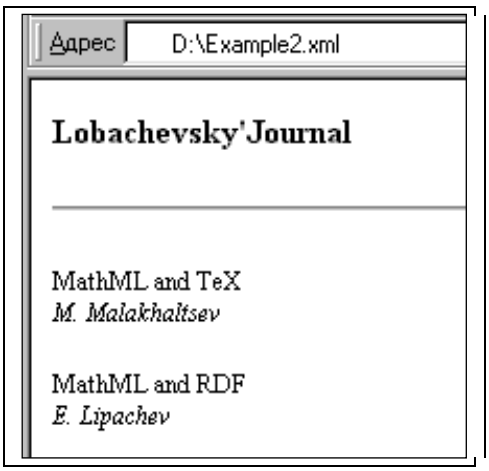
При применении этого шаблона последовательно обрабатываются ветви XML-дерева, начинающиеся с элементов **article** и для каждой из них выводится содержимое элемента **author**.

Приведем еще несколько примеров XSLT-преобразований для вывода информации, содержащейся в файле **example2.xml** в разных видах.

Пример 21. Ещё одна версия файла **one.xsl**.

```
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<h2>
 <xsl:value-of select="journal/jrntitle"/>
</h2>
<hr/>
<xsl:for-each select="journal/articles/article">
<P>
<xsl:value-of select="title"/><br/>
<I><xsl:value-of select="author"/></I>
</P>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

При запуске XML-файла **example2.xml** из примера 20 результат будет иной:



Если в файле **one.xsl** изменить тег **<xsl:for-each** … на

<xsl:for-each select="journal/articles/article" order-by="author">,

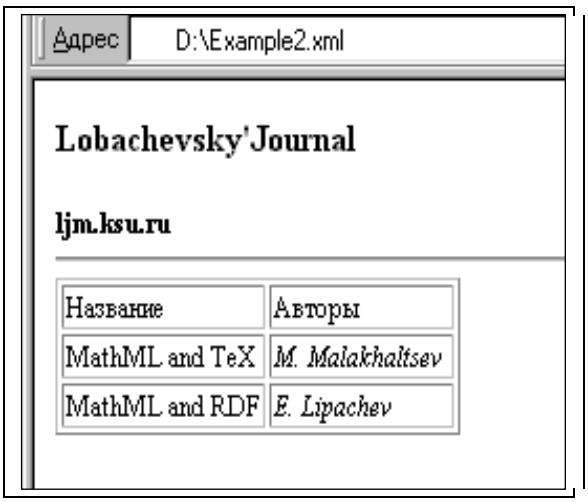
то перед выводом статьи будут упорядочены по авторам. Критерий сортировки задаётся атрибутом **order-by,** при этом символ **";"** используется как разделитель, если критериев несколько.

Пример 22. Следующий стилевой файл используется для отображения XML-файла в виде таблицы. Обращаем внимание на русские названия столбцов таблицы. Для этого в XSL-файле с помощью атрибута **encoding** указана коди-

ровка символов.

```
<?xml version="1.0" encoding="windows-1251"?>
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<h2>
  <xsl:value-of select="journal/jrntitle"/>
</h2>
<b><xsl:value-of select="journal/contacts/url"/> </b>
<hr/>
<table border="1">
<tr>
 <td> Название </td><td> Авторы </td>
</tr>
<xsl:for-each select="journal/articles/article">
<tr>
 <td><xsl:value-of select="title"/></td>
 <td><I><xsl:value-of select="author"/></I></td>
</tr>
</xsl:for-each>
</table>
</xsl:template>
</xsl:stylesheet>
```

Если этот файл сохранить как **one.xsl**, то при запуске файла **example2.xml,** код которого приведен ранее, получим



Язык XSLT имеет все управляющие конструкции языков программирования, например, как мы видели, элемент **<xsl:for-each>** задает оператор

цикла. В рамках настоящего руководства мы не будем описывать все возможности языка XSLT (они изложены в [11]), а приведем пример условного оператора. Рассмотрим следующую таблицу стилей:

```
<xsl:stylesheet version="1.0"
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match="/">
<xsl:for-each select="//article">
<p>
<xsl:if test="@ID=1">
<xsl:value-of select="author"/>
</xsl:if>
</p>
</xsl:for-each>
</xsl:template>
</xsl:stylesheet>
```

В результате применения этой стилевой таблицы к файлу **example2.xml** получим

```
<?xml version="1.0"?>
<p>M. Malakhaltsev</p>
<p></p>
```

При обходе ветвей, начинающихся с элемента **article**, проверяется значение атрибута **Id** этого элемента, и, если он равен 1, то выводится значение элемента **author**. Элемент **xsl:if** имеет атрибут test, который определяет некое выражение. Содержимое элемента является шаблоном. Указанное выражение обрабатывается, а полученный объект преобразуется в булево значение как при вызове функции **boolean**. Если результатом является **true**, то подставляется шаблон, имеющийся в выражении. В противном случае не создается ничего.

Приведенные примеры показывают, что язык XSLT является удобным средством обработки информации, структурированной с помощью XML; с его помощью можно решать достаточно широкий круг задач, связанных с представлением информации в различных формах.

Создание собственного языка на основе XML

Как было отмечено ранее, XML предлагает технологию создания документов. Набор тегов и их назначение в каждом конкретном случае

определяются программистом. Если созданы DTD–объявления, то определен набор правил образования документов и можно подключить проверку документа на синтаксическую корректность. XML Schema служит той же цели и, в большинстве случаев, предпочтительнее DTD–определений. Тогда из заявленного множества тегов можно создавать действительные (valid) документы, – при обработке XML–процессором будут автоматически исключены xml–файлы, не прошедшие синтаксическую проверку. Также необходим набор xsl–преобразований, обеспечивающий трансляцию документов в другие, в том числе стандартные, форматы. Развитые языки разметки, имеющие большие наборы тегов, предполагают, что для отображения документов должны использоваться специализированные программные средства. Например, одним из таких средств для CML является Java–вьюер Jmol, а для MathML – модуль MathPlayer. Распространение нового языка невозможно без указания назначения и возможностей языка, описания набора тегов и правил формирования документов, ссылок на ресурсы, обеспечивающие работу с документами.

Выделяя только основные шаги, обязательные для разработки нового языка разметки, основанного на XML–технологиях, перечислим, какие действия необходимо выполнить.

1. Определить набор тегов и указать их атрибуты.
2. Создать DTD–схемы и файлы XML Schema.
3. Предложить набор xsl–преобразований для обработки информации в новой нотации.
4. Разработать программные средства для управления документами, в частности, обеспечивающими представление в стандартных форматах
5. Подготовить спецификацию языка.

Специализированные языки разметки

Консорциум World Wide Consortium (W3C) предложил XML в качестве рекомендации в 1996 году. С этого времени создано большое число языков разметки, основанных на этой технологии.

Самой ранней технологией разметки на основе XML можно считать CML (Chemical Markup Language). Этот язык предложен в качестве языка разметки химических формул и предназначен для описания и обработки данных о химических соединениях. Близкими по назначению являются, разработанные впоследствии, Analytical Information Markup Language (AniML), Bioinformatic Sequence Markup Language (BSML), BIOpolymer Markup Language (BIOML),

CellML, Computational Chemistry Markup Language (CCML), SpectroML, ThermoML и ряд других (см., например, [16]).

В 1998 году консорциум W3C предложил спецификацию MathML (Mathematical Markup Language). Специфика этого языка разметки – описание и обработка математических формул.

Созданы языки разметки и для других предметных областей, названия языков, как правило, отражают их назначение: Business Rules Markup Language (BRML), Geography Markup Language (GML), , Finite Element Modeling Markup Language (femML), Ink Markup Language (InkML), Mathematics Education Markup Language (MeML), Materials Markup Language (MatML), Numerical Data Markup Language (NDML), Relational-Functional Markup Language (RFML), Robotic Markup Language (RoboML), Voice Extensible Markup Language (VoiceXML).

В работе [15] предложена классификация уже созданных языков разметки в виде карты языков XML.

Далее приводятся ознакомительные сведения о двух языках разметки – CML и MathML, показано какие технологии и программные средства используются для создания и поддержки этих спецификаций. Кратко дается справочная информация о других распространенных языках разметки.

Chemical Markup Language (CML)

Назначение CML и область применения

Язык разметки химических формул **CML** (**C**hemical **M**arkup **L**anguage) разработан как часть проекта Open Molecule Foundation.

С помощью CML можно записать

6. Молекулярные структуры
7. Химические реакции
8. Спектры
9. Вычислительные процессы
10. Неорганические кристаллы
11. Объекты квантовой химии
12. Физические величины

Файлы, содержащие разметку, согласно правилам CML, являются xml–файлами, поэтому для их создания и обработки можно использовать как обычные текстовые редакторы, так и специальные программные средства, ориентированные на XML – технологии. Имеется большой набор программных продуктов для поддержки CML, в том числе для отображения в браузерах

Internet.

Поясняющие примеры

Пример 23. Фрагмент CML – файла [17].

```
<cml>
  <molecule id="m1">
    <atomArray>
      <atom elementType="N"/>
      <atom elementType="O"/>
    </atomArray>
  </molecule>
</cml>
```

Тегом верхнего уровня является **<cml>**, – все последующие теги записаны в контейнере **<cml> - </cml>**. Молекулярная структура представлена внутри тега **<molecule id="m1"> - </molecule>.** Список атомов записан с помощью тега **<atomArray> - </atomArray>**. Тег **<atom elementType="N"/>** определяет атом азота, а тег **<atom elementType="O"/>** – атом кислорода.

Пример 24. Молекула воды H_2O в нотации CML

```
<?xml version="1.0" encoding="UTF-8" ?>
<cml xmlns="http://www.xml-cml.org/schema"
xsi:schemaLocation="http://www.xml-cml.org/schema
../../schema.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<molecule title=”Water1”>
 <atomArray>
   <atom id=”a1” elementType=”H” hydrogenCount=”0”/>
   <atom id=”a2” elementType=”O” hydrogenCount=”2”/>
   <atom id=”a3” elementType=”H” hydrogenCount=”0”/>
</atomArray>
<bondArray>
  <bond atomRefs2=”a1 a2” order=”1”/>
  <bond atomRefs2=”a2 a3” order=”1”/>
</bondArray>
</molecule>
```

```
</cml>
```

Набор интересных примеров можно найти в [18, 19].

Набор тегов

Полный список элементов (всего 137 элементов) и атрибутов языка CML можно найти на сайте http://cml.sourceforge.net/schema/attributetable.html.

DTD и XML Schema для CML

DTD-схема для CML-документов подключается следующей инструкцией

```
<!DOCTYPE cml SYSTEM "http://www.xml-
cml.org/dtd/cml1_0_1.dtd">
```

В настоящее время используются XML-схемы CML2.4 (http://www.sourceforge.net/projects/cml/schema24).

Программные инструменты CML

В работе [20] приведён обзор основных программных инструментов, используемых при работе с документами CML.

Приведем в качестве примера отображение молекулы с помощью 3D-просмотрщика химических структур **Jmol** (http://jmol.sourceforge.net/). Jmol — комплекс кроссплатформенных приложений на языке Java вколючающий в себя: JmolApplet — java-апплет для браузеров, который можно интегрировать в web-страницы; **Jmol** — приложение, запускаемое на локальном компьютере (на котором установлена Java) и набор библиотек **JmolViewer**, которые могут быть интегрированы в другие java-приложения. Отметим, что Jmol является свободно распространяемой программой с исходным открытым кодом.

Следующий CML-файл описывает молекулу, состоящую из двух атомов

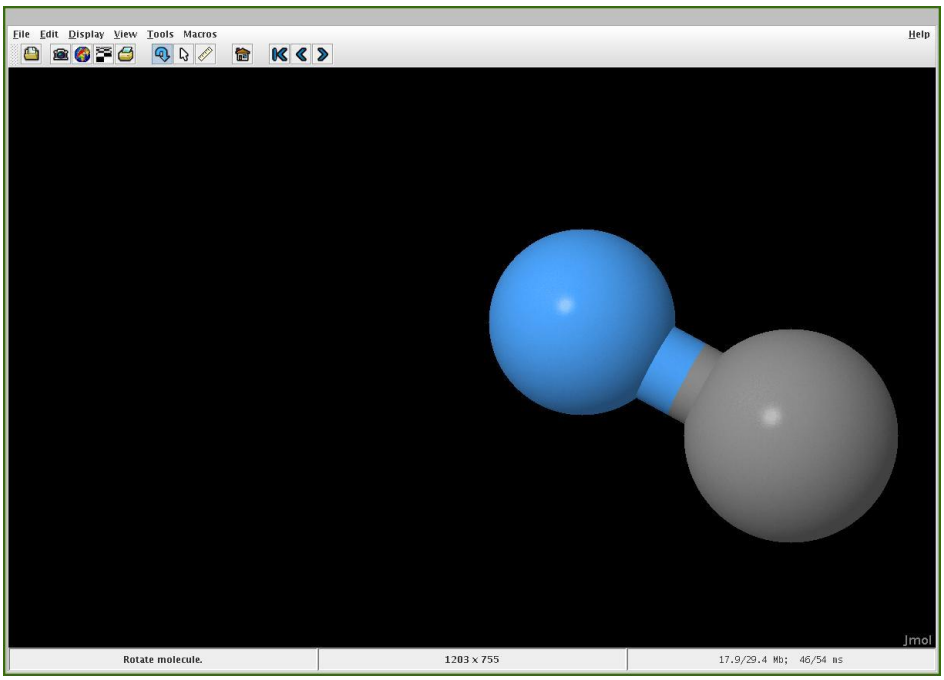
```
<?xml version="1.0" encoding="UTF-8" ?>
<cml
   xmlns="http://www.xml-cml.org/schema"
   xsi:schemaLocation="http://www.xml-cml.org/schema
../../schema.xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:siUnits="http://www.xml-cml.org/units/siUnits"
   xmlns:units="http://www.xml-cml.org/units/units"

xmlns:castep="http://www.materialsgrid.org/castep/namespa
ce"
   xmlns:cml="http://www.xml-cml.org/dict/cmlDict"
 >
```

```
 <parameterList id="input" title="Input Parameters">
  <parameter name="Release" dictRef="castep:release"
value="Academic Release" />
  <parameter name="Version" dictRef="castep:version"
value="3.2" />
  <parameter name="Task" dictRef="castep:task"
value="GEOMETRYOPTIMIZATION" />
  <parameter dictRef="castep:xcFunctional" value="PBE" />
  <parameter dictRef="castep:cutoff">
   <scalar units="units:ev">330.000000000
   </scalar>
  </parameter>
  <parameter name="Number of Electrons"
dictRef="castep:nElectrons" value="9" />
  <parameter name="Net Charge" dictRef="castep:netCharge"
value=" 0" />
  <parameter name="Net Spin" dictRef="castep:netSpin"
value="1" />
  <parameter name="Fix Spin" dictRef="castep:fixSpin"
value="false" />
  <parameter name="Fix Occupancy" dictRef="castep:fixOcc"
value="false" />
  <parameter name="MetalsMethod"
dictRef="castep:metalsMethod" value="DM" />
  <parameter name="Geom Method"
dictRef="castep:geomMethod" value="BFGS" />
  <parameterList id="pspots" title="Pseudopotential
Files">
   <parameter name="PSPFile" dictRef="castep:pspFile"
value="C_00PBE.usp" />
   <parameter name="PSPFile" dictRef="castep:pspFile"
value="Ta_00PBE.usp" />
  </parameterList>
 </parameterList>
 <module title="Initial System">
  <molecule>
   <atomArray>
```

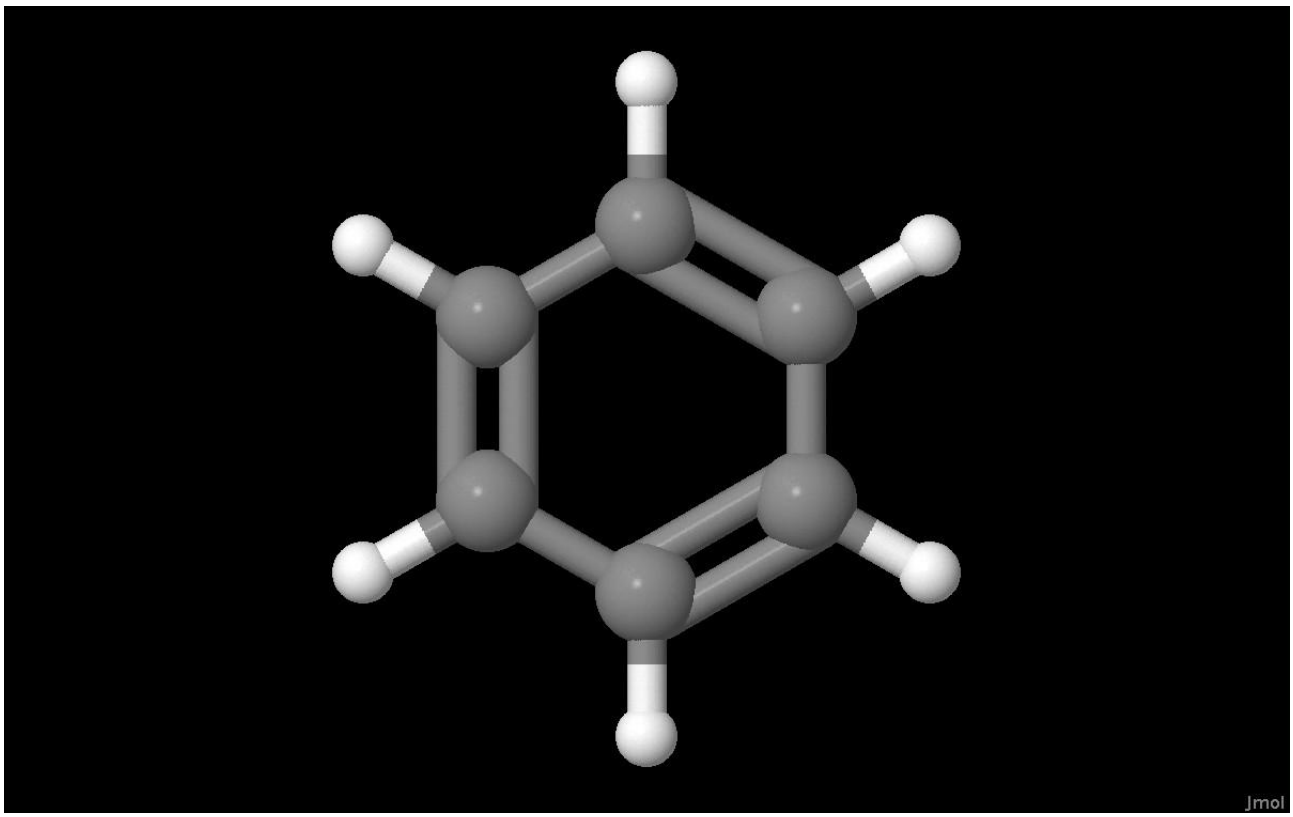
```
    <atom elementType="C" id="a1" xFract="   0.000"
yFract="0.000" zFract=" 0.000" />
    <atom elementType="Ta" id="a2" xFract="   0.500"
yFract=" 0.500" zFract=" 0.500" />
   </atomArray>
  </molecule>
  <crystal id="initStruct" dictRef="castep:ucell">
   <scalar title="a" dictRef="cml:a" units="units:ang"
    xmlns:cml="http://www.xml-cml.org/dict/cmlDict">
   3.230
   </scalar>
   <scalar title="b" dictRef="cml:b" units="units:ang">
   3.230
   </scalar>
   <scalar title="c" dictRef="cml:c" units="units:ang">
   3.230
   </scalar>
   <scalar title="alpha" dictRef="cml:alpha"
dataType="xsd:double"
    units="units:deg">60.000
   </scalar>
   <scalar title="beta" dictRef="cml:beta"
    units="units:deg">60.000
   </scalar>
   <scalar title="gamma" dictRef="cml:gamma"
    units="units:deg">60.000
   </scalar>
  </crystal>
 </module>
 <module title="Final System">
  <molecule>
   <atomArray>
    <atom elementType="C" id="a1" xFract="   0.000"
yFract="0.000" zFract=" 0.000" />
    <atom elementType="Ta" id="a2" xFract="   0.500"
yFract=" 0.500" zFract=" 0.500" />
   </atomArray>
```

```
  </molecule>
  <crystal id="initStruct" dictRef="castep:ucell">
   <scalar title="a" dictRef="cml:a" units="units:ang">
   3.230
   </scalar>
   <scalar title="b" dictRef="cml:b" units="units:ang">
   3.230
   </scalar>
   <scalar title="c" dictRef="cml:c" units="units:ang">
   3.230
   </scalar>
   <scalar title="alpha" dictRef="cml:alpha"
    units="units:deg">60.000
   </scalar>
   <scalar title="beta" dictRef="cml:beta"
    units="units:deg">60.000
   </scalar>
   <scalar title="gamma" dictRef="cml:gamma"
    units="units:deg">60.000
   </scalar>
  </crystal>
  <property title="Total Energy" dictRef="castep:Etot">
   <scalar units="units:ev">-2.92669908E+002
   </scalar>
  </property>
 </module>
 <metadata name="castep:totalTime"
  content="12.2900000792"
  convention="siUnits:s" />
</cml>
```

При открытии этого файла с помощью Jmol видим 3D-изображение этой молекулы:



Jmol позволяет имеет достаточно много возможностей, включая анимацию и возможность конвертации изображений в разные форматы, включая и создание веб-страниц, содержащих изображения молекул. Вот, изображение, молекулы бензина (соответствующий cml-документ слишком велик, чтобы привести его здесь), которое сохранено как jpg-файл с помощью Jmol.



CellML

Назначение этого языка – электронное хранение и обмен математическими моделями. Широко применяется в биологическом моделировании. Поддерживает спецификацию MathML.

Спецификацию этого языка см. в http://www.cellml.org/specifications/cellml_1.0/index_html

DTD-схема для CellML-документов подключается следующей инструкцией

```
<!DOCTYPE
 model SYSTEM "http://www.cellml.org/cellml/cellml_1_0.dtd">
```

Если в модели используются математические формулы и используется язык MathML, также подключается DTD-схема MathML

```
<!DOCTYPE model SYSTEM
http://www.cellml.org/cellml/cellml_1_0.dtd [
  <!ENTITY % use_mathml_dtd "INCLUDE">
]>
```

DTDсхема CellML доступна по адресу http://www.cellml.org/cellml/cellml_1_0.dtd, а XML-схема – по адресу http://www.cellml.org/cellml/cellml_1_1.xsd.

Инструменты для работы с документами можно найти на http://www.cellml.org/tools; http://www.cellml.org/downloads

Mathematical Markup Language (MathML)

Для математического сообщества наибольший интерес представляет MathML (Mathematical Markup Language) – технология, предназначенная для представления математических формул. Разработка этой технологии ведется консорциумом W3C с 1998 года. Поскольку технологии семантического веба предоставляют новый способ организации веб-информации, который дает возможность на более высоком уровне решать задачи программной обработки документов (в частности, задачу поиска), MathML изменяет принципы организации и управления электронными публикациями по математике. В настоящее время язык MathML фактически стал стандартом представления математической информации в электронной форме в силу следующих причин:

- технология обработки данных на основе языка MathML реализует одну из основных тенденций современной информатики – разделение разметки и данных, поэтому она дает широкие возможности многоуровневого структурирования данных и расширенного поиска;
- создано программное обеспечение, использующее технологию MathML; в частности, созданы программные средства, позволяющие конвертировать в MathML документы, подготовленные с помощью имеющихся стандартных технологий (таких, например, как LaTeX, Mathematica, Maple, Word). MathML поддерживается основными браузерами: Internet Explorer (при установке соответствующего плагина), Mozilla, Firefox;
- технология MathML поддерживается системами Maple® и MathCAD 2001, а компания Wolfram Research предложила собственную концепцию использования технологии MathML, которая реализована в пакете Mathematica®, в частности, в этом пакете предусмотрено сохранение документов в формате MathML.

Разметка математических текстов по технологии MathML

Вводный пример

Допустим, что нам надо осуществить MathML-разметку для отображения следующего текста:

Quadratic equation

Корни квадратного уравнения $ax^2+bx+c=0$

имеют вид

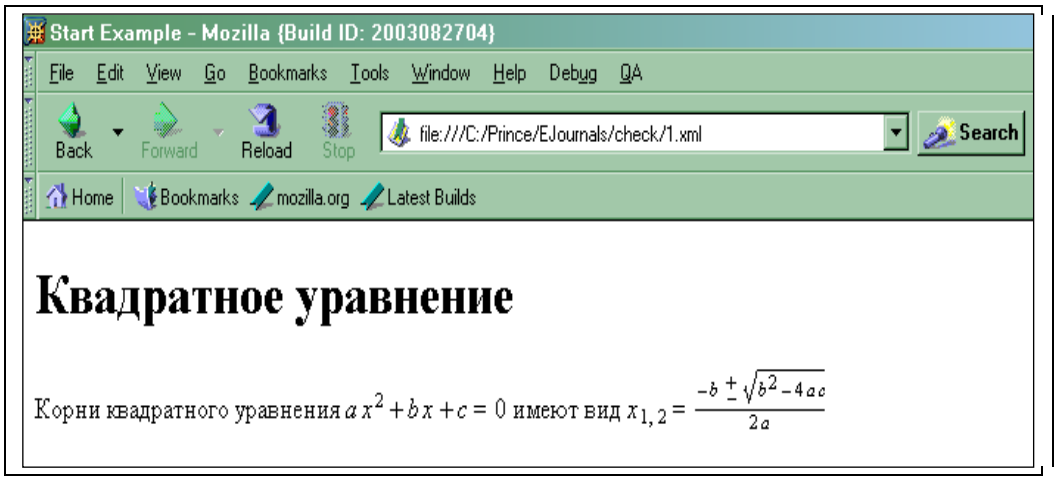
$$x_{1,2}=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

Код для отображения этого фрагмента можем подготовить в стандартном блокноте Windows. Откроем блокнот и наберем:

```
<?xml version="1.0" encoding=”windows-1251”?>
<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
<title>Start Example</title>
</head>
  <body>
    <h1>Quadratic equation</h1>
  Корни квадратного уравнения
<math xmlns="http://www.w3.org/1998/Math/MathML">
      <mi>a</mi>
     <msup>
       <mi>x</mi>
       <mn>2</mn>
    </msup>
   <mo>+</mo>
   <mi>b</mi>
   <mi>x</mi>
    <mo>+</mo>
   <mi>c</mi>
   <mo>=</mo>
   <mn>0</mn>
 </math>
имеют вид
 <math xmlns="http://www.w3.org/1998/Math/MathML">
     <msub>       <mi>x</mi>
        <mrow>
         <mn>1</mn>
        <mo>,</mo>
        <mn>2</mn>
       </mrow>
     </msub>
```

```
      <mo>=</mo>
      <mfrac>
        <mrow>
            <mo>-</mo>
            <mi>b</mi>
            <mfrac linethickness="0">
               <mo>+</mo>
               <mo>-</mo>
            </mfrac>
            <msqrt>
               <msup>
                  <mi>b</mi>
                  <mn>2</mn>
              </msup>
              <mo>-</mo>
              <mn>4</mn>
              <mi>a</mi>
              <mi>c</mi>
           </msqrt>
        </mrow>
        <mrow>
           <mn>2</mn>
           <mi>a</mi>
        </mrow>
      </mfrac>
    </math>
  </body>
</html>
```

Теперь сохраним файл под именем **example1.xml**. Откроем его в браузере Mozilla, мы должны увидеть картинку



Теперь попытаемся разобраться в тексте созданного нами файла. Первые две строки указывают, что данный документ представляет код XHTML. Напомним, что первая строка отмечает, что это XML-документ, а вторая определяет пространство имен XHTML как **`http://www.w3.org/1999/xhtml`**. Хотя эта строка выглядит как ссылка, она является лишь уникальным идентификатором пространства имен, который распознается браузером, и, следовательно, не требует соединения с интернетом. Отметим, что пространство имен используется по умолчанию (т. к. нет префикса после xmlns), поэтому перед тегами не указан префикс, задающий пространство имен (подробно о пространстве имен см. выше раздел «Пространство имен»).

Для разметки текста используются стандартные теги HTML, а для представления формул – теги MathML. Блоки с кодом MathML заключатся в теги

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
……..
</math>
```

Открывающий тег **`<math …..>`** определяет ссылку на пространство имен MathML. В примере необходимо отобразить две формулы, поэтому он содержит два блока **`<math> </math>.`**

Выражения MathML строятся в виде дерева, каждый узел которого является элементом MathML (например, числом или операцией). Все элементы MathML разделены на три группы: *элементы представления* (presentation elements), *элементы содержания* (content elements) и *интерфейсные элементы* (interface elements). Одно и то же выражение может быть размечено в виде «разметки представления» (presentation markup) и «разметки содержания» (content markup). В нашем примере используется разметка представления. *Так как разметка содержания в настоящий момент не поддерживается браузера-*

ми, мы ее не рассматриваем. В рекомендации MathML [23] элементы представления называются также *токенами представления* (*presentation token elements*).

Элементы представления **mi**, **mn** и **mo** используются чаще всего – с их помощью производится разметка для идентификаторов (например, переменных), чисел и операций. В примере имеются элементы **msup** и **msub**, которые используются для отображения верхних и нижних индексов. Во второй формуле нижний индекс двойной, поэтому потребовалась группировка элементов, представляющих индекс. Для группировки использовали элемент **mrow**, который предназначен для обозначения горизонтального ряда частей выражения. Элемент **msqrt** отвечает за представление квадратного корня, а элемент **mfrac** – за представление дроби. Подробно теги MathML описаны в [23].

Работа по подготовке MathML-документов, включая обзор и настройку инструментов, см. в [24], [25].

Materials Markup Language (MatML)

MatML предназначен для описания свойств материалов. Документацию по работе с этим языком можно найти на официальном сайте этого языка www.matml.org.

Geography Markup Language (GML)

Этот язык используется географическим сообществом. Информацию об этом языке можно найти на сайтах http://www.opengis.net/gml/
http://www.opengeospatial.org/standards/gml, http://schemas.opengis.net/gml/

Wireless Markup Language (WML)

WML – язык разметки документов для использования в сотовых телефонах и других мобильных устройствах по стандарту WAP.

Официальная спецификация WML разработана и поддерживается WAP Forum, производственным консорциумом, основанном Nokia, Phone.com, Motorola и Ericsson. Эта спецификация определяет синтаксис, переменные и элементы используемые в файлах WML. Последнее определение типа документа (Document Type Definition) доступны по адресу: http://www.wapforum.org/DTD/wml_1.1.xml

В телефоне или в любом другом коммуникационном устройстве, заявленном как WAP-совместимое, загружено специальное программное обеспечение (известное как микроброузер), которое полностью понимает, как

обрабатывать все вариации WML 1.1 DTD.

WML был разработан для для устройств с низкой пропускной способностью и маленьким дисплеем. В качестве составной этого дизайна была применена концепция дек и карт. Один WML-документ (а точнее элементы, содержащиеся внутри элемента <wml>) называется декой (deck). Интерактивное взаимодействие с пользователем осуществляется с помощью карт (card). Достоинство такой реализации заключается в том, что несколько экранов могут быть загружены на клиентское устройство за один раз. Используя WMLScript, обработка действий пользователя может быть произведена с использованием находящихся в одной деке карт, исключая тем самым множественные транзакции с сервером.

Пример 26. WML-документ, организующий выбор имени пользователя из предложенного списка, проверку пароля и вывод на экран полученных данных.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"

"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
        <card id="Login" title="Login">
                <do type="accept" label="Password">
                        <go href="#Password"/>
                </do>
                <p>
                UserName:
                <select name="name" title="Name:">
                        <option value="John Doe">John
Doe</option>
                        <option value="Paul Smith">Paul
Smith</option>
                        <option value="Joe Dean">Joe
Dean</option>
                        <option value="Bill Todd">Bill
Todd</option>
                </select>
                </p>
        </card>
```

```
        <card id="Password" title="Password:">
                <do type="accept" label="Results">
                        <go href="#Results"/>
                </do>
                <p>
                Password: <input type="text"
name="password"/>
                </p>
        </card>
        <card id="Results" title="Results:">
        <p>
        You entered:<br/>
        Name: $(name)<br/>
Password: $(password)<br/>
        </p>
        </card>
</wml>
```

Comic Book Markup Language (CBML)

В заключение приведем пример любопытного языка. Этот язык применяется для создания электронных коллекций комиксов. Подробную информацию можно найти на сайте языка http://www.cbml.org/technical.html

Верификация документов осуществляется с помощью DTD-объявлений, которые подключаются в документ следующей инструкцией

```
<!DOCTYPE cbml PUBLIC "-//CBML Project//CBML Main Driver
File//EN" "cbml.dtd">
```

Литература

1. *Язык разметки* // Википедия /http://ru.wikipedia.org/wiki.
2. *Extensible Markup Language (XML) 1.0 (Fourth Edition)* /

http://www.w3.org/TR/2006/REC-xml-20060816
3. Extensible Markup Language (XML) 1.1 (Second Edition) /
4. Н. Питц-Моултис, Ч. Кирк *XML*. – СПБ: БХВ – Петербург, 2000. – 736 с.
5. И. Ш. Хабибуллин *Самоучитель XML*. – СПБ: БХВ – Петербург, 2003.
6. *XML Schema Language: Part 0 Primer* /http://www.w3.org/TR/xmlschema-0/
7. *XML Schema Version 1.1 Part 1: Structures*. / http://www.w3.org/TR/2005/WD-xmlschema11-1-20050224/structures.html
8. Ashvin Radiya , Vibha Dixit *The basics of using XML Schema to define elements* /http://www.ibm.com/developerworks/xml/library/xml-schema/.
9. *XSL Transformations (XSLT) Version 1.0 W3C Recommendation* 16 November 1999 http://www.w3.org/TR/1999/REC-xslt-19991116; Рус. перевод: http://www.rol.ru/news/it/ helpdesk/xslt01.htm.
10. *XSL Transformations (XSLT) Version 2.0. W3C Recommendation* 23 January 2007 – http://www.w3.org/TR/xslt20. *XML Path Language (Xpath) Version 1.0 W3C Recommendation* 16 November 1999 – http://www.w3.org/TR/xpath.
11. *XML Path Language (Xpath) Version 1.0 W3C Recommendation* 16 November 1999 – http://www.w3.org/TR/xpath.
12. *Miloslav Nic, Jiri Jirat*, XPath Tutorial, http://www.zvon.org/xxl/XpathTutorial/ General/examples.html; Рус. перевод: – http://www.zvon.org/xxl/XPathTutorial/ General_rus/ examples.html
13. XPath Tutorial, – http://www.w3schools.com/xpath/.
14. *Школы консорциума W3C*, – www.xml.nsu.ru.
15. А. Лозовюк. *Комета по имени XML* – http://www.marketer.ru/
16. Peter Murray-Rust, Henry S. Rzepa: *STMML. A markup language for scientific, technical and medical publishing*. Data Science Journal 1(2): 1-65 (2002)
17. Amies, *Introduction to Extensible Markup Language for Chemistry and Biosciences* – http://www.medicalcomputing.net/xml_biosciences.html
18. A. Amies, *Basic Biological Chemistry with Extensible Markup Language* http://www.medicalcomputing.net/biological_chem_computer2.html

19. J. Jirat, *Chemical Markup Language 1.0 reference with examples* http://www.zvon.org/xxl/CML1.0/
20. Amies, *Tools for Working with Chemical Markup Language* – http://www.medicalcomputing.net/*cmltools.html*
21. *CellML 1.0 Specification* / http://www.cellml.org/specifications/cellml_1.0/index_html
22. *CellML 1.1 XML Schema* / http://www.cellml.org/cellml/cellml_1_1.xsd
23. *Mathematical Markup Language (MathML) Version 2.0 (Second Edition)* / http://www.w3.org/TR/2003/REC-MathML2-20031021/
24. А.М. Елизаров, Е.К. Липачёв, М.А. Малахальцев. *Основы MathML Представление математических текстов в Internet. Практическое руководство*. – Казань: Изд-во Казанского математического общества, 2004. – 60 с. www.ksu.ru/
25. А.М. Елизаров, Е.К. Липачёв, М.А. Малахальцев. *Основы MathML Представление математических текстов в Internet.* – Казань, 2008. – 101 с. http://www.niimm.ksu.ru/data/preprints/
26. P. Murray-Rust and H. S. Rzepa, *Scientific publications in xml - towards a global knowledge base* // Data Sci. J. 2002, 1, 84-98. / http://www.ch.ic.ac.uk/rzepa/codata/
27. H. S. Rzepa, P. Murray-Rust and B. J. Whitaker The Internet as a Chemical Information Tool, Chem. Soc. Revs, 1997, 1-10. doi:10.1039/CS9972600001
28. P. Murray-Rust, *Chemical Markup Language. A Simple introduction to Structured Documents* http://www.xml.com/pub/a/w3j/s3.rustintro.html
29. P. Murray--Rust and H. S. Rzepa, *Chemical Markup, XML, and the Worldwide Web. 1. Basic Principles*, J. Chem. Inf. Comput. Sci., **1999**, *39*, 928-942. doi:10.1021/ci990052b
30. P. Murray--Rust and H. S. Rzepa, *Chemical Markup, XML and the World--Wide Web. 2. Information Objects and the CMLDOM*, J. Chem. Inf. Comput. Sci., **2001**, *41*. doi:10.1021/ci000404a
31. G. V. Gkoutos and P. Murray--Rust and S. Rzepa and M. Wright, *Chemical Markup, XML, and the World-Wide Web. 3. Toward a Signed Semantic Chemical Web of Trust*, J. Chem. Inf. Comput. Sci., **2001**, *41*, 1124-1130. doi:10.1021/ci000406v
32. P. Murray-Rust, H. S. Rzepa and M. Wright, Development of Chemical Markup Language (CML) as a System for Handling Complex Chemical

Content, New J. Chem., **2001**, 618-634.
33. P. Murray-Rust and H. S. Rzepa, *Chemical Markup, XML and the World--Wide Web. 4. CML Schema*, J. Chem. Inf. Comput. Sci. **2003**, *43*, 757-772. doi:10.1021/ci0256541
34. P. Murray-Rust and H. S. Rzepa and J. Williamson and E. L. Willighagen, *Chemical Markup, XML and the World--Wide Web. 5. Applications of Chemical Metadata in RSS Aggregators*, J. Chem. Inf. Comput. Sci., **2004**, *44*, 462-469. doi:10.1021/ci034244p
35. P. Murray-Rust and H. S. Rzepa, *Towards a semantic web for chemistry in lifescience*. http://lists.w3.org/Archives/Public/public-swls-ws/2004Sep/att-0025/molsem.html
36. G. L. Holliday, P. Murray-Rust, H. S. Rzepa, *Chemical Markup, XML and the Worldwide Web. Part 6. CMLReact; An XML Vocabulary for Chemical Reactions*, J. Chem. Inf. Mod., **2006**, *46*, 145-157. doi:10.1021/ci0502698
37. S. Kuhn, T. Helmus, R. J. Lancashire, P. Murray-Rust, H. S. Rzepa, C. Steinbeck, E. L. Willighagen, *Chemical Markup, XML, and the World Wide Web. 7. CMLSpect, an XML Vocabulary for Spectral Data*, J. Chem. Inf. Mod., **2007**, *47*, 2015 -2034. doi:10.1021/ci600531a
38. WML, **http://www.codenet.ru/webmast/wml/wwml.php**
39. E.F. Begley and C.P. Sturrock, The Background and Development of MatML, a Markup Language for Materials Property Data
40. *Materials Markup*, Technology Reports. http://xml.coverpages.org/materials.html
41. *CellML 1.0 Specification* / http://www.cellml.org/specifications/cellml_1.0/index_html
42. *CellML 1.1 XML Schema* / http://www.cellml.org/cellml/cellml_1_1.xsd
43. *Namespaces in XML 1.0 (Second Edition)* / http://www.w3.org/TR/2006/REC-xml-names-20060816
44. *Namespaces in XML 1.1 (Second Edition)* / http://www.w3.org/TR/2006/REC-xml-names11-20060816