A Gentle Introduction to SGML

Содержание

1 Введение в SGML

Схема кодирования, определяемая этим Руководством, является приложением системы, известной как стандартный язык обобщенной разметки (Standard Generalized Markup Language, SGML).[1] SGML — международный стандарт на определение не зависящих от платформы и системы методов представления текстов в электронной форме. Эта глава представляет из себя краткое введение в основы SGML для тех читателей, кто не встречались с ним ранее. . . .

SGML является международным стандартом на описание размеченного электронного текста. Точнее, SGML — это *метаязык* (metalanguage), то есть, средство формального описания языка, в данном случае, *языка разметки* (markup language). Прежде, чем продолжать, определим эти термины.

Исторически слово *разметка* (markup) использовалось для описания аннотаций или других отметок в тексте, предназначенных для указания машинистке или наборщику, как именно должна быть напечатана или набрана определенная фраза. Примеры включают волнистое подчеркивание для обозначения жирного шрифта, специальные символы для обозначения пропуска отдельных предложений или их печати определенным шрифтом, и т.п. С автоматизацией форматирования и печати текстов термин был расширен, охватывая сейчас всяческие *коды разметки* (markup codes), вставляемые в электронные тексты для управления форматированием, печатью или иной обработкой.

Обобщая, мы определяем разметку или *кодирование* (encoding), как любой метод выявления интерпретации текста. На примитивном уровне все печатные тексты кодированы в этом смысле: знаки пунктуации, использование заглавных букв, размещение букв на странице, даже пробелы между словами можно считать своеобразной разметкой, функция которой — помочь читателю определить, где заканчивается одно слово и начинается другое, или как отделить структурные элементы, например, заголовки, или элементы локальной структуры, например, подчиненные предложения. Кодирование текста для компьютерной обработки в принципе, так же, как и транскрипция рукописи из scriptio continua, — процесс выявления того, что неявно или предположительно, процесс указания пользователю, как интерпретировать содержимое текста.

Под *языком разметки* мы понимаем набор соглашений о разметке, используемых в комплексе для кодирования текстов. Язык разметки должен специфицировать, какая

[1] International Organization for Standardization, *ISO 8879: Information processing — Text and office systems — Standard Generalized Markup Language (SGML),* ([Geneva]: ISO, 1986).

разметка является допустимой, какая — необходимой, как различаются разметка и текст, и что разметка означает. SGML предоставляет средства решения первых трех задач; последняя требует документов, подобных настоящему Руководству.

Настоящая глава пытается дать неформальное введение — гораздо менее формальное, чем сам стандарт — в те составляющие SGML, понимание которых необходимо для наилучшего понимания этого Руководства.

2 Особенности SGML

Существуют три характеристики SGML, отличающие его от других языков разметки: его упор на описательную, а не на процедурную разметку; его концепция *типа документа* (document type); его независимость от конкретной системы в представлении текста. Эти три аспекта кратко обсуждаются ниже, и, более подробно, в разделах 4 и 8.

2.1 Описательная разметка

Система описательной разметки использует коды разметки, просто предоставляющие названия для классификации частей документа. Коды, такие, как **<para>** или **\end{list}** просто идентифицируют часть документа и утверждают про нее: «следующий элемент — параграф» или «это — конец начатого последним списка» и т.д. Напротив, система процедурной разметки определяет, какая обработка должна производиться в конкретной точке документа: «здесь вызвать процедуру PARA с параметрами 1, b и x», или «сдвинуть левую границу на 2см влево, правую — на 2см вправо, пропустить строку и встать на новую левую границу», и т.д. В SGML инструкции, необходимые для обработки документа с определенными целями (например, для его форматирования) четко отделяются от описательной разметки, встречающейся внутри документа. Обычно они собираются вне документа в отдельных процедурах или программах.

При описательной, а не процедурной, разметке один и тот же документ можно обрабатывать различными программами, каждая из которых может применять различные правила обработки к тем частям документам, которые она считает важными. Например, программа анализа содержимого может совершенно игнорировать сноски в аннотируемом тексте, тогда как программа форматирования может извлекать и собирать их вместе для печати в конце каждой главы. С одними и теми же частями файла могут ассоциироваться разные правила обработки. Например, одна программа может выделять имена людей и географические имена для создания индекса или базы данных, а другая, оперирующая тем же текстом, может печатать имена собственные шрифтом отличающегося начертания.

2.2 Типы документов

SGML вводит понятие *типа документа* и, как следствие, *определения типа документа* (document type definition, DTD). Тип документа формально определяется его

составными частями и их структурой. Например, определение отчета может констатировать, что он состоит из заголовка, возможно, автора, за которым следуют аннотация и один или несколько абзацев. Все, что не имеет заголовка, в соответствии с этим формальным определением, отчетом не является, так же, как не является им последовательность абзацев, за которой следует аннотация, вне зависимости от того, насколько такие документы похожи на отчет для читателя–человека.

Раз документы имеют известные типы, можно использовать специальную программу, называющуюся *анализатором* (parser), для проверки документа, утверждающего свою принадлежность определенному типу. Анализатор проверяет, что все элементы, требуемые типом документа, на самом деле присутствуют и расположены в правильном порядке. Что более важно, разные документы одного и того же типа могут обрабатываться одинаковым образом. Можно конструировать программы, использующие знание структуры документа, которые, таким образом, могут действовать в более осмысленной манере.

2.3 Независимость данных

Основная цель создания SGML заключалась в том, чтобы обеспечить транспортабельность закодированных документов из одной аппаратной и программной среды в другую без потери информации. Два описанных выше свойства решают эту задачу на абстрактном уровне; третье свойство — на уровне строк байтов (символов), из которых составляется документ. SGML предоставляет универсальный механизм *строковой подстановки* (string substitution), то есть, простой машинно–независимый способ обозначить, что некоторая последовательность символов в документе должна заменяться при его обработке некоторой другой последовательностью. Одно очевидное применение этого механизма — обеспечение согласованности номенклатуры; другое, и более важное, — противодействие печально известной неспособности различных компьютерных систем понимать наборы символов друг друга, или способ в любой системе предоставить все графические символы, необходимые для конкретного приложения, путем использования описательных обозначений непереносимых символов. Строки, определенные этим механизмом подстановки, называются *объектами* (entities) и обсуждаются ниже в разделе 8.

3 Структура текста

Текст не является просто однородной последовательностью слов, или, тем более, байтов. Для различных целей он может быть разделен на множество разных единиц разных типов или размеров. Текст в прозе, такой, как этот, можно разделить на разделы, главы, абзацы и предложения. Стихотворный текст можно делить на песни, строфы и строки. Будучи напечатанным, и проза и поэзия может делиться на тома, сборники, страницы.

Такого рода структурные единицы чаще всего используют для идентификации конкретной позиции или ссылки внутри текста («третье предложение второго абзаца десятой главы»; «песнь 10, строка 1234»; «страница 412» и т.п.), но они могут также применяться для деления текста на осмысленные сегменты для аналитических целей («отличаются ли

средние длины предложений раздела 2 и раздела 5?»; «сколько абзацев разделяют появления слова *природа*?»; «сколько страниц?»). Иные структурные единицы имеют явно аналитический характер, характеризуя часть текста. Драматический текст может считать каждую реплику разных действующих лиц единицей одного сорта, а ремарки или описания перемещений — единицами другого сорта. Такой анализ менее полезен для нахождения частей текста («93я реплика Горацио во втором акте»), чем для сравнения слов, использованных разными действующими лицами, или одним и тем же лицом в разных местах спектакля.

В прозаическом тексте также можно выделять в качестве единиц разнообразные предложения прямой или непрямой речи, фразы, использующие разную стилистику (описание, спор, комментарий и т.п.), фразы разных людей и так далее. А для некоторых типов анализа (например, критики) может оказаться важным физическое представление конкретного печатного или рукописного источника: парадоксально, что описательная разметка может использоваться для описания свойств представления, таких, как шрифт, разбиение на строки, использование пробелов и тому подобных.

Эти структуры тестов пересекаются друг с другом сложным и непредсказуемым образом. В частности, при работе с текстами, существующими в бумажном виде, читатель должен понимать и физическую организацию книги, и логическую структуру работы, в ней содержащейся. Многие вещи (например, *Tristram Shandy* Sterne) нельзя полностью оценить, не зная игры между повествовательными единицами (такими, как главы и абзацы) и границами страниц. Для многих типов исследований существенным является как раз взаимосвязь различных уровней анализа: степень связи синтаксической или повествовательной структур, или отсутствие таковой, или степень отражения морфологических структур в фонетике.

4 SGML структуры

Этот раздел описывает простой и согласованный механизм разметки или идентификации структурных единиц текста, предоставляемый SGML. Он также описывает, какие способы SGML предлагает для выражения правил, определяющих возможные осмысленные комбинации этих единиц в любых текстах.

4.1 Элементы

В стандарте SGML для текстовых единиц, рассматриваемых как структурные компоненты, используется термин *элемент* (element). Различным типам элементов даются различные названия, но SGML не предлагает никаких способов выразить значение конкретного типа элементов, кроме его отношения к другим типам элементов. То есть, все, что можно сказать про элемент, называющийся (например) **<blort>**, — это то, что его экземпляры могут встречаться (а могут и не встречаться) внутри элементов типа **<farble>**, и что он может раскладываться (а может и не раскладываться) на элементы типа **<blortette>**. Следует подчеркнуть, что стандарт SGML совершенно не заботит

семантика текстовых элементов: она зависит от приложения[1] Дело создателей SGML–совместимых наборов разметок (таких, как описанный в этом Руководство) — выбрать осмысленные имена идентификаторов элементов и документировать правильное их использование в разметке текстов. Это — одна из целей данного документа. От необходимости выбора названий элементов, кодирующих их функцию, происходит технический термин для названия типа элемента: *обобщенный идентификатор* (generic identifier), или GI.

В размеченном тексте (*экземпляре документа*, document instance) каждый элемент должен быть явно размечен или отмечен некоторым образом. Стандарт предоставляет несколько разных способов это сделать, наиболее часто используемый из них — вставить метку (tag) в начале элемента (*открывающая метка*, start–tag) и еще одну — в конце элемента (*закрывающая метка*, end–tag). Пара открывающей и закрывающей меток используется для выделения элементов в тексте, так же, как разные скобки или кавычки используются в обычной пунктуации. Например, элемент цитирования может быть отмечен в тексте так:

```
... реплика Розалинды <quote>Ничего глупее я никогда не
слышала!</quote> ясно показывает ...
```

Как показывает данный пример, открывающая метка имеет вид **<название>**, где открывающая угловая скобка означает начало открывающей метки, «название» — обобщенный идентификатор отмечаемого элемента, и закрывающая угловая скобка означает конец метки. Закрывающая метка имеет аналогичный вид, за исключением того, что за открывающей угловой скобкой стоит символ косой черты, так что соответствующая закрывающая метка будет **</название>**.[2]

4.2 Модели содержимого элемента: пример

Элемент может быть *пустым* (empty), то есть, не содержать внутри вообще ничего; элемент может содержать просто текст. Чаще, однако, элементы одного типа будут целиком содержаться (embed) внутри элементов другого типа.

Для иллюстрации, рассмотрим очень простую структурную модель. Предположим, что в антологии мы хотим идентифицировать только стихотворения, их заголовки, строфы и строки, из которых они состоят. В терминах SGML тип нашего документа — антология (anthology), и он состоит из последовательности стихотворений (poem). Каждое стихотворение содержит один элемент, заголовок (title), и несколько экземпляров другого, строфы (stanza), каждая строфа содержит несколько элементов строк (line). Полностью размеченный, текст, отвечающий этой модели может выглядеть так:[3]

[1] В настоящий момент идет работа по созданию (с использованием синтаксиса SGML) определения стандартного «языка семантики и спецификации стилей документов (document style and semantics specification language, DSSSL)».

[2] На самом деле символы, используемые в качестве ограничителей (угловые скобки, косая черта, восклицательный знак) могут переопределяться, но удобно использовать символы, приведенные в этом описании.

[3] Пример взят из *Songs of innocence and experience*, William Blake, 1794. Разметка иллюстративная и не отвечающая TEI.

```
<anthology>
  <poem><title>The SICK ROSE</title>
     <stanza>
          <line>O Rose thou art sick.</line>
          <line>The invisible worm,</line>
          <line>That flies in the night</line>
          <line>In the howling storm:</line>
     </stanza>
     <stanza>
          <line>Has found out thy bed</line>
          <line>Of crimson joy:</line>
          <line>And his dark secret love</line>
          <line>Does thy life destroy.</line>
     </stanza>
  </poem>

          <!-- тут идут другие стихотворения    -->

</anthology>
```

Нужно подчеркнуть, что этот пример *не* использует те названия, которые предлагаются для соответствующих элементов в данном Руководстве: вышеприведенный фрагмент *не является* допустимым документом TEI. Он будет, однако, служить введением в синтаксис SGML. Пробелы и возвраты каретки в этом примере служат только для облегчения чтения; они не несут определенного смысла в SGML кодировании. Кроме того, строка

```
<!-- тут идут другие стихотворения -->
```

является *комментарием* (comment) SGML и не считается частью текста.

В этом примере не делается никаких предположений о правилах, устанавливающих, например, может ли заголовок встречаться в других местах, кроме как предшествуя первой строфе, или допустимы ли строки, не являющиеся частью строфы: поэтому разметка выглядит такой многословной. В подобных случаях начало и конец каждого элемента должны быть явно отмечены из–за отсутствия ясных правил о разрешенном появлении каждого элемента. Однако, на практике обычно формулируют такие правила, которые позволяют уменьшить число меток. Например, рассмотрев нашу упрощенную модель стихотворения, мы можем установить следующие правила:

1. Антология состоит из нескольких стихотворений и ничего более.
2. Стихотворение всегда имеет единственный заголовок, который предшествует первой строфе и не содержит других элементов.
3. Кроме заголовка, стихотворение состоит только из строф.
4. Строфы состоят только из строк, и каждая строка содержится в строфе.
5. За строфой не может следовать ничего, кроме другой строфы или конца стихотворения.

6. За строкой не может следовать ничего, кроме другой строки или начала новой строфы.

Из этих правил можно вывести отсутствие необходимости явно отмечать концы строф или строк. Из правила 2 следует, что нам не нужно отмечать конец заголовка — он подразумевается началом первой строфы. Аналогично, из правил 3 и 1 следует, что нам не нужно отмечать конец стихотворения: так как стихотворения не могут встречаться внутри стихотворений, а должны встречаться внутри антологий, то конец стихотворения подразумевается началом следующего или концом антологии. С этими упрощениями то же стихотворение может быть размечено так:

```
<anthology>
     <poem><title>The SICK ROSE
     <stanza>
          <line>O Rose thou art sick.
          <line>The invisible worm,
          <line>That flies in the night
          <line>In the howling storm:
     <stanza>
          <line>Has found out thy bed
          <line>Of crimson joy:
          <line>And his dark secret love
          <line>Does thy life destroy.

     </poem>
          <!-- тут идут другие стихотворения    -->

</anthology>
```

Возможность использования правил, устанавливающих, какие элементы могут быть вложены в другие, является очень важным свойством SGML. Не переходя к дальнейшему разбору этих правил, можно попытаться рассмотреть, как размеченный вышеприведенным образом текст может быть обработан с различными целями. Простая индексирующая программа может выделять только значимые элементы текста для генерации списка заголовков, или слов, использованных в тексте стихотворения; простая программа форматирования может вставлять пустые строчки между строфами, возможно, начиная с красной строки первую строчку каждой строфы, или вставляя номер строфы. Разные части каждого стихотворения могут набираться разными способами. Более сложная аналитическая программа может соотносить использование знаков пунктуации со строфовыми и метрическими разделами.[1]

Исследователи, желающие видеть следствия изменений разделов строф или строк, выбранных редактором этого стихотворения, могут это сделать просто меняя положения меток. И, конечно, представленный выше текст может быть перенесен с одного компьютера на другой и обработан любой программой (или человеком), понимающей смысл

[1]Заметим, что этот простой пример не принимает во внимание проблему явной разметки таких элементов, как предложения; следствия этого рассматриваются ниже в разделе 6.2.

внесенных в него меток, безо всяких преобразований и трансляций, необходимых обычно для перемещения файлов текстовых процессоров.

5 Определение структуры документов SGML: DTD

Правила наподобие вышеописанных — первый шаг в создании формальной спецификации структуры SGML документа или *определения типа документа*, обычно сокращаемого как *DTD*. При создании DTD дизайнер документа может задавать произвольно жесткую или сколь угодно гибкую структуру. Нужно найти компромисс между удобством следования простым правилам и сложностью поддержки реальных текстов. Это особенно справедливо, когда определяемые правила относятся к уже существующим текстам: дизайнер может иметь очень туманное представление об изначальном предназначении или смысле старых текстов, и задание непротиворечивых правил, касающихся их структуры, может быть очень сложным. С другой стороны, когда специфицируется новый текст, например, для ввода в некоторую текстовую базу данных, то чем точнее установлены правила, тем лучше они могут быть выдержаны. Даже в случае разметки уже существующего текста может иметь смысл определить ограничивающий набор правил, относящихся к определенному видению текста или гипотезе, касающейся текста, — хотя бы как средство проверки полезности этого видения или гипотезы. Важно помнить, что каждое определение типа документа является интерпретацией текста. Не существует единственного DTD, охватывающего все сведения о тексте, хотя может быть удобно предпочитать одни DTD другим для конкретных типов анализа.

В настоящее время SGML шире всего применяется там, где основным требованием является единообразие структуры документов. Например, при производстве технической документации весьма важно, чтобы разделы и подразделы были соответствующим образом вложены, чтобы перекрестные ссылки были корректны, и так далее. В таких ситуациях к документам относятся как с сырому материалу, к которому применяется заранее определенный набор правил. Однако, как говорилось выше, использование простых правил может также сильно упростить задачу аккуратной разметки элементов и менее ограниченных текстов. Делая такие правила явными, исследователь уменьшает свою работу по разметке и проверке электронного текста, в то же время выявляя интерпретацию структуры и значимые особенности кодируемого текста.

5.1 Пример DTD

DTD в SGML выражается в виде набора описательных утверждений с использованием определенного в стандарте простого синтаксиса. Для нашей простой модели стихотворения подойдут следующие описания:

```
<!ELEMENT anthology      - -  (poem+)>
<!ELEMENT poem           - -  (title?, stanza+)>
<!ELEMENT title          - O  (#PCDATA) >
<!ELEMENT stanza         - O  (line+)   >
```

```
<!ELEMENT line              O O  (#PCDATA) >
```

Эти пять строчек — примеры формальных описаний элементов SGML. Описание, как и элемент, ограничено угловыми скобками; первым символом за открывающей скобкой должен быть восклицательный знак, за которым сразу следует одно из небольшого набора определенных в SGML ключевых слов, указывающее на тип описываемого объекта. Все пять этих описаний — одинакового типа: каждое начинается с ключевого слова **ELEMENT**, показывающего, что описывается элемент, в определенном выше техническом смысле. Каждое состоит из трех частей: название или группа названий, два символа, задающих *правила минимизации* (minimization rules) и *модель содержимого* (content model). Каждая из этих частей обсуждается ниже. Компоненты описания разделяются «пустым местом», то есть, одним или более пробелом, табуляцией или переводом строки.

Первая часть каждого из описаний дает обобщенный идентификатор описываемого элемента, например, *poem*, *title*, и т.д. Как будет показано ниже, в одном утверждении можно описывать несколько элементов.

5.2 Правила минимизации

Вторая часть описания задает *правила минимизации* для элемента. Эти правила определяют, обязаны ли присутствовать открывающая и закрывающая метки для каждого появления данного элемента. Они имеют вид пары символов, разделенных пробелом, первый из которых относится к открывающей, а второй — к закрывающей метке. В обоих случаях должны присутствовать или минус или буква O; минус означает, что метка должна присутствовать, а буква O — что она может быть опущена. Так, в нашем примере каждый элемент, кроме **<line>**, должен иметь открывающую метку. Только элементы **<poem>** и **<anthology>** обязаны также иметь и закрывающую метку.

5.3 Модель содержимого

Третья часть каждого описания, заключенная в круглые скобки, называется *моделью содержимого* элемента, потому что она указывает, что могут содержать экземпляры элемента. Содержимое указывается либо в терминах других элементов, либо при помощи специальных зарезервированных слов. Есть несколько таких зарезервированных слов, из которых самое часто используемое — #PCDATA. Это сокращение от parsed character data (разобранные символьные данные), и оно означает, что описываемый элемент может включать любые разрешенные символьные данные. Если представить себе SGML описание в виде структуры наподобие генеалогического дерева, с одним предком наверху (в нашем случае, это будет **<anthology>**), то почти всегда, если следовать по ветвям дерева вниз (например, от **<anthology>** к **<poem>**, **<stanza>**, **<line>** или **<title>**), мы придем к #PCDATA. В нашем примере так определены **<title>** и **<line>**. Так как в их модели содержимого указано только #PCDATA и не названо никаких включаемых элементов, то они не могут содержать другие элементы.

5.4 Обозначения включения

Вышеприведенное описание для **<stanza>** устанавливает, что строфа состоит из одной или более строк. Оно использует *обозначение включения* (occurence indicator) — знак плюс — для указания того, сколько раз может встречаться элемент, поименованный в модели содержимого. В синтаксисе SGML есть три обозначения включения, обычно представленных знаком плюс, вопросительным знаком и звездочкой.[1] Знак плюс означает, что соответствующий элемент может встречаться один или более раз; вопросительный знак означает, что может быть не более одного элемента; звездочка означает, что элемент может или отсутствовать, или появляться один и более раз. Так, если бы модель содержимого для **<stanza>** была **(LINE*)**, были бы допустимы строфы без строк, так же, как и с более чем одной строкой. Если бы она была **(LINE?)**, то пустые строфы были бы тоже допустимы, но ни одна строфа не могла бы иметь более чем одну строку. Описание **<poem>** в примере устанавливает, что **<poem>** не может иметь больше одного заголовка (но может не иметь ни одного) и что оно должно иметь как минимум одну **<stanza>** (и может иметь несколько).

5.5 Связки

Модель содержимого **(TITLE?, STANZA+)** содержит больше одного компонента. Поэтому нужно дополнительно указать порядок, в котором эти элементы (**<title>** и **<stanza>**) могут появляться. Это упорядочение определяется *связкой* (group connector) — запятой — использованным между ее компонентами. Существуют три возможных связки, обычно представляемых запятой, вертикальной чертой и знаком «&».[2]

Запятая означает, что оба компонента, которые она соединяет, должны встречаться в порядке, указанном в модели содержимого. Знак «&» указывает, что компоненты, которые он соединяет, должны встречаться оба, но в произвольном порядке. Вертикальная черта означает, что может встречаться только один из компонентов, которые она соединяет. Если бы в нашем примере запятую заменить на знак «&», то заголовок мог бы появляться или перед строфами стихотворения, или в его конце (но не между строфами). Если ее заменить на вертикальную черту, то стихотворение могло бы состоять или из заголовка, или только из строф — но не из того и другого.

5.6 Группы модели

До сих пор в нашем примере компоненты каждой модели содержимого были или единственным элементом, или #PCDATA. Вполне можно, однако, определять модели содержимого, в которых компонентами являются списки элементов, объединенные связками. Такие списки, известные как *группы модели* (model groups), могут также модифицироваться обозначениями включения и, в свою очередь, быть объединенными связками.

[1] Так же, как и ограничители, эти знаки имеют формальные наименования и могут быть переопределены соответствующим SGML описанием.

[2] Так же, как ограничители и обозначения включения, связки имеют в стандарте формальные имена и могут быть переопределены соответствующим SGML описанием.

Чтобы продемонстрировать эти возможности, расширим наш пример так, чтобы включать нестрофовые формы стихов. Для демонстрации классифицируем стихотворения на *строфовые* (stanzic), *двустишия* (couplets), и *белые* (blank) или *??* (stichic). Белый стих состоит просто из строк (игнорируем пока возможность стихотворных абзацев)[1], так что дополнительных элементов не нужно. Двустишие определяется как **<line1>**, за которой идет **<line2>**.

```
<!ELEMENT couplet O O (line1, line2) >
```

Элементы **<line1>** и **<line2>** (которые различаются, например, чтобы сделать возможными изучение схемы рифмования) имеют в точности ту же модель содержимого, что и существующий элемент **<line>**. Они, следовательно, могут разделять одно и то же описание. В этой ситуации удобно указать *группу названий* (name group) в качестве первого компонента единого описания элемента, а не записывать последовательность описаний, отличающихся только используемыми именами. Группа названий — это список GI, соединенный связками и заключенный в круглые скобки:

```
<!ELEMENT (line | line1 | line2) O O (#PCDATA) >
```

Описание элемента **<poem>** теперь можно изменить так, чтобы включить все три варианта:

```
<!ELEMENT poem - O (title?, (stanza+ | couplet+ | line+) ) >
```

То есть, стихотворение состоит из необязательного заголовка, за которым следует одна или несколько строф, либо одно или несколько двустиший, либо одна или несколько строк. Отметьте разницу между этим определением и следующим:

```
<!ELEMENT poem - O (title?, (stanza | couplet | line)+ ) >
```

Второй вариант, используя обозначение включения у группы, а не у каждого элемента внутри группы, позволит одному стихотворению состоять из смеси строф, двустиший или белого стиха.

Таким образом можно строить довольно сложные модели, отражая структурную сложность различных типов текстов. В следующем примере мы рассмотрим строфовый стих, в котором появляется рефрен (refrain). Он может состоять из повторений элементов–строк или быть просто текстом, не разделенным на стихотворные строки. Рефрен может появляться только в начале стихотворения или как необязательное дополнение после каждой строфы. Это можно выразить моделью содержимого наподобие следующей:

```
<!ELEMENT refrain - - (#PCDATA | line+)>
<!ELEMENT poem    - O (title?,
                       ( (line+)
                       | (refrain?, (stanza, refrain?)+ ) )) >
```

То есть, стихотворение состоит из необязательного заголовка, за которым следует или последовательность строк, или безымянная группа, открывающаяся рефреном, за

[1]Проницательный читатель заметит тот факт, что стихотворные абзацы не обязательно начинаются на границе строк, что серьезно усложняет жизнь; см. далее раздел 6.2.

которым идет одна или несколько других групп, каждый член которой состоит из строфы с необязательным рефреном. Этому образцу отвечает последовательность *рефрен – строфа – строфа – рефрен*, так же, как и *строфа – рефрен – строфа – рефрен*. А последовательность *рефрен – рефрен – строфа – строфа* ему не удовлетворяет, так же, как и *строфа – рефрен – рефрен – строфа*. Среди прочих условий, накладываемых этой моделью, — требования, чтобы в стихотворении было хотя бы одна строфа, если оно не состоит просто из строк, и чтобы при наличии и заголовка и строфы они появлялись именно в этом порядке.

6 Усложнение: еще об описаниях элементов

В простых случаях, о которых шла речь до сих пор, предполагалось, что можно легко идентифицировать содержимое каждого элемента, определенного в текстовой структуре. Стихотворение состоит из строф, а антология состоит из стихотворений. Строфы не болтаются сами по себе, без стихотворений; стихотворение не может содержать в себе антологию. Все элементы данного типа документов могут быть выстроены в иерархическую структуру наподобие генеалогического дерева с единственным предком наверху и многочисленными потомками (в основном, элементами, содержащими #PCDATA) внизу. Это сильное упрощение оказывает неожиданно эффективным в большом числе случаев. Однако, оно не адекватно общей сложности реальных текстовых структур. В частности, оно не справляется со случаем более или менее свободно плавающих элементов, которые могут встретиться почти на любом уровне иерархии в структуре. Так же оно не справляется со случаем, когда различные элементы перекрываются друг с другом, или когда в одном и том же документе могут быть идентифицированы различные деревья. Для первого случая SGML предоставляет механизм *исключений* (exceptions); для второго — позволяет определять «альтернативные» структуры документа.

6.1 Исключения в модели содержимого

В большинстве документов существуют некоторые элементы, которые могут встречаться на любом уровне их структуры. Аннотации, например, могут относиться к стихотворению в целом, к строфе, к строке строфы или к отдельному слову в ней. В критическом издании то же самое может относиться к вариантам изложения. В таком простом случае сложность добавления элемента аннотации как необязательного компонента всех моделей содержимого еще не очень обременительна; в более реалистичной сложной модели, содержащей, возможно, десять–двенадцать уровней, такой подход будет намного более сложным.

Чтобы с этим справиться, SGML позволяет любую модель содержимого модифицировать списком *исключений*. Есть два типа исключений: *включающие* (inclusions), то есть, дополнительные элементы, которые могут включаться в любом месте в группе модели или в любом из входящих в нее элементов; и *исключающие* (exclusions), то есть, элементы, которые не могут включаться в текущей модели.

Чтобы расширить наши описания, разрешив аннотации и вариантное написание, которые, будем считать, могут появляться где угодно по тексту стихотворения, сначала добавим описания этих двух элементов:

```
<!ELEMENT (note | variant) - - (#PCDATA)>
```

Оба элемента, аннотация (note) и вариант (variant), должны иметь и открывающую, и закрывающую метки, так как они могут появляться где угодно. Чтобы не добавлять их к моделям содержимого каждого типа стихотворений, мы можем добавить их в форме списка включений к элементу стихотворения, который теперь выглядит так:

```
<!ELEMENT poem - O (title?, (stanza+ | couplet+ | line+) )
                                          +(note | variant) >
```

Знак плюс перед списком названий **(NOTE | VARIANT)** означает, что это «включающее» исключение. С таким дополнением аннотации или варианты могут появляться в любом месте внутри содержимого элемента стихотворения — даже в тех элементах (как, например, **<title>**), которые объявлены с моделью содержимого #PCDATA. Они, таким образом, могут также появляться внутри аннотаций или вариантов!

Если мы по каким–то причинам хотим предотвратить появление аннотаций или вариантов внутри заголовков, мы можем добавить исключающее исключение к описанию **<title>**:

```
<!ELEMENT title  - O  (#PCDATA)  -(note | variant) >
```

Знак минус перед списком названий **(NOTE | VARIANT)** означает, что это «исключающее» исключение. Теперь аннотациям и вариантам запрещено появляться внутри заголовков, невзирая на их потенциальное включение, подразумеваемое предыдущей добавкой к модели содержимого для **<poem>**.

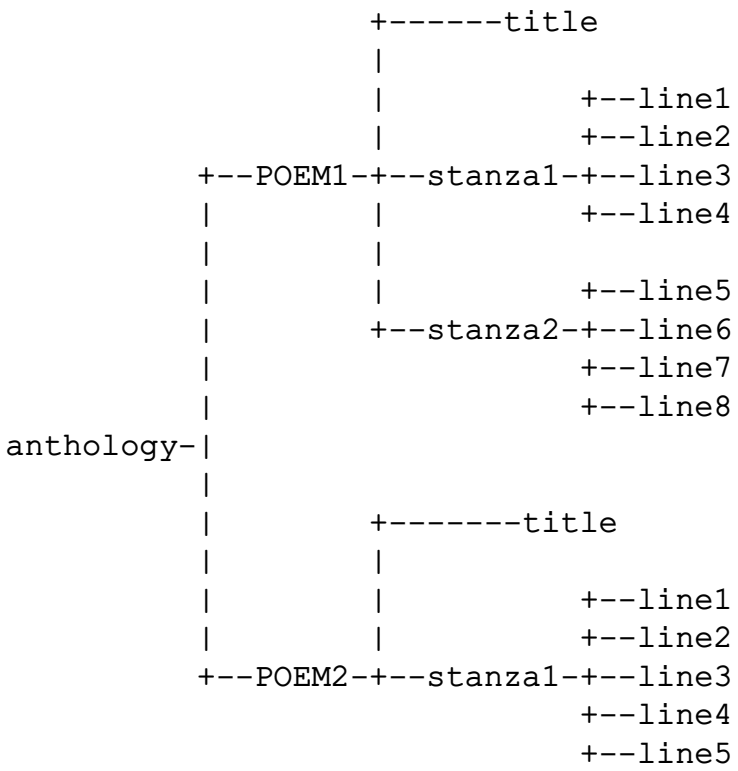
Аналогичным образом мы можем избежать вложенности аннотаций и вариантов самих в себя, модифицировав вышеприведенное их описание таким образом:

```
<!ELEMENT (note | variant) - - (#PCDATA)  -(note | variant) >
```

Дотошный читатель заметит, что это запрещает и варианты внутри аннотаций, и аннотации внутри вариантов. При использовании исключений следует быть осторожным, так как их последствия могут быть неочевидными.

6.2 Альтернативные структуры

Все структуры, которые мы до сих пор обсуждали, были просто иерархическими: то есть, на каждом уровне дерева каждый узел целиком содержался в родительском узле. Следующая иллюстрация представляет структуру документа, удовлетворяющего простому DTD, которое мы к этому моменту определили в виде дерева (для экономии места нарисованного на боку). Вы уже видели как стихотворение Уильяма Блейка можно разделить на заголовок и две строфы, по четыре строки каждая. На этой диаграмме мы добавим второе стихотворение, состоящее из одной строфы и заголовка, чтобы создать экземпляр антологии:



Очевидно, что для описания структуры этой и других антологий можно нарисовать много таких деревьев. Некоторые из них можно представить как дальнейшее ветвление этого дерева: например, мы можем разделить строки на отдельные слова, поскольку слова не пересекают границ строк. Но не менее очевидно, что можно нарисовать множество иных деревьев, которые *не* поместятся в этом дереве. Мы, например, можем быть заинтересованы в синтаксических структурах — которые редко согласуются с формальными границами стихов. Или, например, нас может интересовать разбиение на страницы различных редакций одного текста.

Одним из способов достичь этого может быть группирование строк и заголовков нашей модели в страницы. Описание такого элемента достаточно просто:

```
<!ELEMENT page - - ((title?, line+)+)   >
```

То есть, страница состоит из одной или более безымянных групп, каждая из которых содержит необязательный заголовок, за которым идет последовательность строк. (Заметим, что, по случайности, эта модель запрещает заголовку встречаться внизу страницы). Однако, просто вставить элемент **<page>** в уже определенную иерархию не так просто, как это могло бы показаться. Некоторые стихотворения длиннее, чем одна страница, и некоторые страницы содержат больше одного стихотворения. Мы, следовательно, не можем вставить элемент **<page>** в иерархию ни между **<anthology>** и **<poem>**, ни между **<poem>** и **<stanza>**, ни в обоих местах сразу! То, что нам нужно, — это возможность создать независимую иерархию, с теми же элементами внизу (строфы, строки и заголовки), но скомбинированными в другую структуру. Эту возможность предоставляет в SGML свойство CONCUR.

Для каждого иерархического дерева, в которое отображается текст, должно быть

создано отдельное определение типа документа. Определение, которое мы до сих пор выстроили для антологии, полностью выглядит так[1]:

```
<!DOCTYPE anthology [
<!ELEMENT anthology      - -  (poem+)             >
<!ELEMENT poem           - -  (title?, stanza+)   >
<!ELEMENT stanza         - O  (line+)             >
<!ELEMENT (title | line) - O  (#PCDATA)           >
]>
```

Как показывает этот пример, название типа документа всегда должно совпадать с названием самого верхнего элемента иерархии. Давайте теперь добавим к этому описанию второе определение для параллельного типа документа, который мы назовем постраничной антологией или **<p.anth>**[2]:

```
<!DOCTYPE p.anth [
<!ELEMENT p.anth         - -  (page+)               >
<!ELEMENT page           - -  ((title?, line+)+)    >
<!ELEMENT (title|line)   - O  (#PCDATA)             >
]>
```

Теперь мы определили два разных способа смотреть на один базовый текст — компоненты #PCDATA, группируемые обоими определениями типа документа в строки или заголовки. С одной точки зрения строки собираются в строфы и стихотворения; с другой — они собираются только в страницы. Заметим, что с обоих точек мы смотрим на тот же самый текст: две иерархии просто позволяют нам структурировать его двумя разными способами.

Чтобы разметить оба представления одновременно, необходимо будет обозначить, к какой иерархии принадлежит каждый элемент. Это делается с помощью указания имени типа документа (представления) в круглых скобках непосредственно перед идентификатором, как в открывающей, так и в закрывающей метках. Так, страницы (видимые только в типе документа **<p.anth>**) должны размечаться меткой **<(p.anth)page>** в начале и **</(p.anth)page>** в конце. Таким же образом, так как стихотворения и строфы появляются только в типе документов **<anthology>**, они должны размечаться с использованием меток **<(anthology)poem>** и **<(anthology)stanza>**, соответственно. Однако, для элементов строк и заголовков, появляющихся в обеих иерархиях, не нужно задавать тип документа: каждая метка, содержащая только название, подразумевается отмечающей элемент, присутствующий во всех типах документов.

В качестве простого пример, представим, что стихотворение Блейка встречается в некоторой антологии, с границей страницы, приходящейся посередине первой строфы. Стихотворение может тогда быть размечено так:

```
<(anthology)anthology>
<(p.anth)p.anth>
```

[1]Используемый синтаксис далее обсуждается в разделе 10.2.

[2]Точка в имени элемента служит, как правило, для структурирования пространства имен и никакого специального значения не имеет. (прим. переводчика)

```
<(p.anth)page>

<!--      тут прочие заголовки и строки этой страницы  -->

     <(anthology)poem><title>The SICK ROSE
     <(anthology)stanza>
          <line>O Rose thou art sick.
          <line>The invisible worm,
</(p.anth)page>
<(p.anth)page>
          <line>That flies in the night
          <line>In the howling storm:
     <(anthology)stanza>
          <line>Has found out thy bed
          <line>Of crimson joy:
          <line>And his dark secret love
          <line>Does thy life destroy.
     </(anthology)poem>

<!--      тут остаток материала на этой странице    -->
</(p.anth)page>

</(p.anth)p.anth)
</(anthology)anthology>
```

Теперь возможно выбирать только элементы, относящиеся к определенному представлению текста, несмотря на то, что оба они представлены в разметке. Приложение, интересующееся только разбиением на страницы, будет видеть только элементы, чьи метки включают спецификацию **P.ANTH** или не включают спецификации вообще. Приложение, интересующееся только представлением **ANTHOLOGY**, не будет видеть границы страниц. А приложение, интересующееся взаимосвязью обоих представлений, может это делать без неоднозначности.

Следует предупредить: CONCUR является необязательным свойством в SGML, не все существующие программные системы SGML его поддерживают, и не все из поддерживающих делают это в соответствии с буквой стандарта. Поэтому, когда данное Руководство определяет потенциальную применимость CONCUR, оно также обязательно рекомендует альтернативные методы. Для более полного обсуждения этих вопросов см. главу **??**.

Заметим, что мы не можем ввести новый элемент, например, номер страницы, в тип документа **<p.auth>**, так как в типе документа **<anthology>** нет данных, которые могли бы быть в него помещены. Один из способов добавления такой дополнительной информации рассматривается в следующем разделе.

7 Атрибуты

В контексте SGML, слово *атрибут* (attribute), подобно другим, имеет строгий технический смысл. Оно используется для описания информации, являющейся в каком–либо смысле описательной для конкретного появления элемента, но не являющейся частью его содержимого. Например, можно добавить атрибут *status* к экземплярам некоторых элементов для обозначения степени их достоверности, или добавить атрибут *identifier*, так что можно будет ссылаться на конкретное появление элемента из других мест документа. Атрибуты полезны именно в таких случаях.

Хотя разные элементы могут иметь атрибуты с одинаковыми названиями (например, в схеме TEI каждый элемент определяется имеющим атрибут *id*), эти атрибуты всегда считаются различными и могут иметь различные присваиваемые им значения. Если элемент определен имеющим атрибуты, значения атрибутов задаются в документе как *пары атрибут–значение* внутри открывающей метки экземпляра элемента. Закрывающая метка не может содержать спецификаций атрибут–значение, так как это было бы излишним.

Например:

```
<poem id=P1 status="draft"> ... </poem>
```

Элемент **<poem>** определен имеющим два атрибута: *id* и *status*. Для экземпляра **<poem>** в этом примере, представленного многоточием, атрибут *id* имеет значение **P1**, а атрибут *status* — значение **draft**. SGML–процессор может использовать значения атрибутов произвольным образом; например, форматировщик может печатать элементы стихотворения, имеющие значением атрибута *status* **draft** и **revised** по–разному; другой процессор может использовать тот же атрибут, чтобы определить, нужно ли вообще обрабатывать соответствующие элементы стихотворения. Атрибут *id* является особым случаем, так как, по соглашению, он всегда используется для снабжения конкретного экземпляра элемента уникальным значением, которое может использоваться для перекрестных ссылок, как обсуждается ниже.

Как и элементы, атрибуты описываются при объявлении типа документа, с использование довольно похожего синтаксиса. Кроме спецификации названия атрибута и элемента, к которому он относится, можно указать (в некоторых границах), какие виды значений приемлемы для атрибута, и каково его значение по умолчанию.

Для определения двух атрибутов, которые мы указывали выше у элемента **<poem>**, можно использовать следующие объявления:

```
<!ATTLIST poem
          id       ID                                #IMPLIED
          status   (draft | revised | published)   draft        >
```

Объявление начинается с символа **ATTLIST**, который открывает *спецификацию списка атрибутов* (attribute list specification). Первая его часть указывает рассматриваемый элемент (или элементы). В нашем примере атрибуты объявляются только для элемента **<poem>**. Если несколько элементов имеют одни и те же атрибуты, все они могут быть определены в одном объявлении. Для этого, точно так же, как и в объявлениях

элементов, нужно задавать в круглых скобках список из нескольких названий. За названием (или списком) следует набор строк, где объявляются по одному атрибуты; каждое объявление содержит три части. Они задают название атрибута, тип принимаемых им значений и значение по умолчанию, соответственно.

Названия атрибутов (*id* и *status* в этом примере) подпадают под те же ограничения, что и другие идентификаторы в SGML; однако, они должны быть уникальны не в пределах всего DTD, а только в пределах списка атрибутов данного элемента.

Вторая часть спецификации атрибута может принимать одну из двух форм, обе из которых показаны выше. В первом случае используется одно из нескольких ключевых слов, объявляя тип значений, принимаемых атрибутом. В примере фигурирует специальное ключевое слово **ID**, показывающее, что атрибут *ID* будет использоваться для указания уникального идентифицирующего значения для каждого стихотворения (см. дальнейшее обсуждение ниже). Среди прочих ключевых слов SGML:

CDATA Значение атрибута может содержать любые разрешенные символьные данные; в значение могут входить метки, но они не будут распознаны синтаксическим анализатором SGML и не будут обрабатываться как обычно.

IDREF Значение атрибута должно содержать указатель на некоторый другой элемент (см. дальнейшее обсуждение **ID**).

NMTOKEN Значение атрибута является *идентификатором* (name token), то есть (более или менее) произвольной строкой алфавитно–цифровых символов.

NUMBER Значение атрибута состоит только из цифр.

В предыдущем примере был задан список возможных значений атрибута *status*. Это значит, что синтаксический анализатор может убедиться, что в документе нет **<poem>** со значением атрибута *status* не из числа **draft**, **revised** или **published**. Либо, если значение объявлено как CDATA или NAME, анализатор будет принимать почти любую строку символов (**status=awful** или **status=12345678**, если значение было **NMTOKEN**; **status="сойдет что попало"**

или **status="ну, ПОЧТИ что попало"**

, если оно было **CDATA**). Конечно, иногда набор возможных значений нельзя определить заранее. Когда это возможно, то лучше так и сделать.

Последняя часть информации в каждом определении атрибута указывает, как анализатор должен интерпретировать отсутствие рассматриваемого атрибута. Это делается с помощью указания одного из нижеперечисленных ключевых слов или (как в данном случае) конкретного значения, которое трактуется как значение этого атрибута по умолчанию. В нашем примере, если стихотворение просто размечено как **<poem>**, анализатор будет трактовать его в точности как если бы оно было размечено как **<poem status='draft'>**. В ином случае, значение по умолчанию для атрибута может быть указано при помощи одного из следующих ключевых слов:

#REQUIRED Значение должно быть указано.

#IMPLIED Значение не обязано быть указано (как в случае *ID* выше).

#CURRENT Если в данном появлении элемента не указано значения, использовать последнее указанное.

Например, если вышеприведенные определения атрибутов переписать в виде

```
<!ATTLIST poem
          id       ID                                #IMPLIED
          status   (draft | revised | published)  #CURRENT      >
```

то стихотворения, обозначенные в антологии просто как **<poem>**, будут трактоваться как если бы они имели тот же статус, что и предыдущее стихотворение. Если вместо **#CURRENT** употребить ключевое слово **#REQUIRED**, то анализатор будет считать такие стихотворения ошибочно размеченными, так же, как если значение будет отличаться от одного из **draft, published,** или **revised**. Использование **#CURRENT** подразумевает, что значение, указанное для этого атрибута у первого стихотворения, будет применяться ко всем следующим, пока не будет изменено новым значением. Следовательно, если статусы всех стихотворений одинаковы, то его нужно указывать только у первого.

Иногда необходимо из одного элемента сослаться на другой, очевидными примерами чего служат «см. замечание 6» или «как указано в главе 5». Во время подготовки текста действительные номера, соответствующие замечаниям или главам могут быть не определены. Если мы используем описательную разметку, такие вещи, как номера страниц или глав, будучи целиком и полностью вопросом представления, ни в коем случае не могут присутствовать в размеченном тексте: они будут присвоены приложением, обрабатывающим текст (и могут на самом деле отличаться в разных приложениях). SGML предоставляет специальный механизм, с помощью которого экземпляру любого элемента можно присвоить специальный идентификатор, вроде метки, который может использоваться для ссылок на него из любого другого места в этом же тексте. Перекрестная ссылка сама по себе является элементом специального вида, который так же должен быть объявлен в DTD. В любом случае идентифицирующая метка (которая может быть произвольной) указывается в качестве значения специального атрибута.

Предположим, например, что мы хотим в аннотацию к одному стихотворению включить ссылку на другое. Сначала нам нужно предоставить способ присоединять метку к каждому стихотворению: это делается определением атрибута для элемента **<poem>**, как рекомендовано выше.

```
<!ATTLIST poem
          id       ID     #IMPLIED >
```

Здесь мы определяем атрибут *id*, значение которого должно быть типа **ID**. Название *id* для атрибутов типа **ID** не является обязательным; однако, оно является удобным соглашением, которому практически всегда следуют. Заметим, что не каждое стихотворение должно нести атрибут *id*, и анализатор свободно игнорирует его отсутствие. Только те стихотворения, на которые мы собираемся ссылаться, нуждаются в использовании этого атрибута; в открывающую метку каждого такого стихотворения мы теперь должны включить некоторый уникальный идентификатор, например:

```
<POEM ID=Rose>
   Текст стихотворения с идентификатором 'ROSE'
</POEM>
```

```
<POEM ID=P40>
   Текст стихотворения с идентификатором 'P40'
</POEM>

<POEM>
   У этого стихотворения нет идентификатора
</POEM>
```

Далее мы определим новый элемент для собственно перекрестных ссылок. Он не будет иметь содержимого — являясь только указателем — но будет иметь атрибут, значением которого будет идентификатор указываемого элемента. Это достигается следующим объявлением:

```
<!ELEMENT poemref - O EMPTY                  >
<!ATTLIST poemref     target IDREF #REQUIRED >
```

Элемент **<poemref>** не нуждается в закрывающей метке, так как у него нет содержимого[1]. У него единственный атрибут *target*. Значение этого атрибута должно быть типа **IDREF** (ключевое слово, используемое для перекрестных указателей такого типа) и обязано быть указанным.

С использованием этих объявлений мы теперь можем закодировать ссылку на поэму с *id* **Rose** следующим образом:

```
Стихотворение Блейка про больную розу <POEMREF TARGET=Rose> ...
```

Когда анализатор SGML дойдет до этого пустого элемента, он просто проверит существование элемента с идентификатором **Rose**. Различные SGML–приложения могут предпринимать разнообразные дополнительные действия: форматировщик может сконструировать точную ссылку на номер страницы и строки стихотворения в текущем документе и вставить ее, или просто процитировать заголовок или первые строки стихотворения. Гипертекстовый процессор может использовать этот элемент как сигнал активизации ссылки на стихотворение. Назначением SGML разметки является просто показать, что такая ссылка существует: она не определяет, что приложение будет с ней делать.

8 Объекты SGML

Обсуждавшиеся до сих пор аспекты SGML все имели отношение к разметке структурных элементов документа. SGML также предоставляет простой и гибкий метод кодирования и наименования произвольных частей действительного содержимого документа переносимым образом. В SGML слово *объект* (entity) несет специальный смысл: оно означает именованную часть размеченного документа, безотносительно ко всяческим соображениями структуры. Объектом может быть строка символов или целый файл текста. Для включения его в документ используется конструкция, известная как *ссылка на объект* (entity reference). Например, следующее объявление

[1]Что указывается в модели с помощью ключевого слова EMPTY (пустой). (прим. переводчика)

```
<!ENTITY tei "Text Encoding Initiative">
```

определяет объект, называющийся **tei** и значением которой является строка «Text Encoding Initiative».[1]

Это был пример *объявления объекта* (entity declaration), которое объявляет *внутренний объект* (internal entity). Следующее объявление, напротив, объявляет *системный объект* (system entity):

```
<!ENTITY ChapTwo SYSTEM "sgmlmkup.txt">
```

Это определяет системный объект, называющийся **ChapTwo**, значением которого является текст, ассоциированный с системным идентификатором — в этом случае, системный идентификатор — имя файла операционной системы и текст замещения объекта является содержимым файла.

После того, как объект объявлен, на него можно ссылаться в любом месте документа. Это делается путем использования его названия, перед которым ставится символ «&», а после которого — точка с запятой. Точка с запятой может быть опущена, если за ссылкой на объект следует пробел или конец записи.

Когда SGML анализатор встречает такую *ссылку на объект*, он немедленно подставляет вместо названия объекта объявленное значение. Так, фраза «Работа &tei только начата» будет интерпретирована SGML процессором в точности как если бы она была записана как «Работа Text Encoding Initiative только начата». В случае системного объекта, понятно, подставляется содержимое файла операционной системы, так что фраза «Следующий текст был опущен: &ChapTwo;» будет раскрыта, чтобы включать целиком все, что система найдет в файле **sgmlmkup.txt**.[2]

Это, очевидно, экономит нажатия на клавиши и упрощает задачу поддержки непротиворечивости набора документов. Если печать сложного документа происходит во многих местах, само тело документа может использовать ссылку на объект, такую как **`&site;`**, там, где требуется название места. Затем в разных местах можно добавить разные объявления объектов для подстановки вместо этой ссылки соответствующего названия, без необходимости менять текст самого документа.

Этот механизм *строковой подстановки* имеет много других применений. Его можно использовать для обхода известного неравенства множества компьютерных систем в представлении полного диапазона символов, необходимых для показа современного английского (не говоря уже о требованиях других современных алфавитов или древних языков). Так называемые «специальные символы», напрямую не доступные с клавиатуры (или, если доступные, неправильно транслируемые при передаче) могут представляться

[1] По соглашению названия объектов являются чувствительными к регистру, в отличие от названий элементов.

[2] Строго говоря, SGML не требует, чтобы системные объекты были файлами; они могут в принципе быть любыми источниками данных, доступными SGML процессору: файлами, результатами запросов к базе данных, результатами вызовов системных функций — чем угодно. Однако, при изучении SGML проще представлять себе системные объекты ссылающимися на файлы, и настоящее обсуждение далее игнорирует прочие возможности. Все существующие SGML процессоры поддерживают использование системных объектов для ссылок на файлы; немногие поддерживают другие возможные использования системных объектов.

ссылкой на объект.

Предположим, например, что мы ходим закодировать использование лигатур в старых печатных текстах. «Лигированная» форма *ct* должна отличаться от не лигированной путем кодирования ее в виде **&ctlig;**, а не в виде **ct**. Прочие специальные типографские элементы вроде линеек или leafstops точно так же можно представить в тексте в виде мнемонических ссылок на объекты. При обработке таких текстов будет добавлено объявление объектов, давая желаемое представление для подобных элементов текста. Если, например, лигированные буквы интереса не представляют, мы можем просто добавить объявление вида

```
<!ENTITY ctlig "ct" >
```

и различие, представленное в исходном документе, будет снято. Если, с другой стороны, используется форматирующая программа, способная представить лигированные символы, мы можем заменить объявление объекта так, чтобы дать ту последовательность символов, которую требует такая программа.

Список объявлений объектов известен как *набор объектов* (entity set). Стандартные наборы объектов, доступные для большинства SGML процессоров, обычно включают названия, взятые из списков таких имен, опубликованных как приложение к стандарту SGML и в других местах.

Значения подстановок, заданные в объявлениях объектов, конечно, сильно системно –зависимы. Если используемые в них символы нельзя набрать напрямую, SGML предоставляет механизм спецификации символов по их численным значениям, известный как *ссылки на символы* (character references). Ссылка на символ отличается от остальных символов в строках подстановки тем, что она начинается со специального символа, обычно — последовательности *&#* и заканчивается обычной точкой с запятой. Например, если используемый форматировщик представляет лигированную форму ct по символам c и t, предваряемыми символом с десятичным значением 102, объявление объекта может выглядеть так:

```
<!ENTITY ctlig "&#102;ct" >
```

Заметим, что ссылки на символы обычно не имеет смысла переносить между программными или аппаратными платформами, поэтому их использование рекомендуется только в ситуациях, подобных описанной.

Хотя механизм ссылок на объекты и полезен для редких отклонений от ожидаемого набора символов, никто не будет пользоваться им для кодирования длинных предложений, например, цитат на греческом или русском в английском тексте. В подобных случаях нужно использовать иные механизмы, обсуждаемые в других частях этого Руководства (см. главу **??**).

В объявлениях SGML разметки может использоваться специальная форма объектов, *параметрические объекты* (paremeter entities); они отличаются от ранее обсуждавшихся объектов (которые называются *обычными объектами* (general entities) двумя свойствами:

- Параметрические объекты используются *только* внутри объявлений SGML раз-

метки; за некоторыми не рассматриваемыми сейчас исключениями они обычно не встречаются в собственно документе.
- Параметрические объекты ограничиваются знаком процента и точкой с запятой, а не знаком «&» и точкой с запятой.

Объявления параметрических объектов имеют тот же вид, что и объявления обычных, но между ключевым словом **ENTITY** и названием объекта вставляется знак процента, с обоих сторон отделенный пустыми местами (пробелами, табуляциями или переводами строк). Так могут быть объявлены внутренний объект **TEI.prose**, раскрывающийся в **INCLUDE**, и внешний параметрический объект **TEI.extensions.dtd**, ссылающийся на системный файл **mystuff.dtd**:[1]

```
<!ENTITY % TEI.prose 'INCLUDE'>
<!ENTITY % TEI.extensions.dtd SYSTEM 'mystuff.dtd'>
```

Определение типа документов TEI широко использует параметрические объекты для управления выбором различных наборов элементов и для облегчения модификации TEI DTD. Множество примеров их использования можно найти в главе **??**.

9 Отмеченные секции

Иногда бывает удобно отметить некоторые части текста так, чтобы SGML анализатор относился к ним по особому. Например, некоторые части юридических шаблонов должны регулярно включаться или опускаться, в зависимости от штата или страны, для которой готовится документ. (Так, предложение «Ответственность ограничена $50,000» должно включаться в штате Delaware, но опускаться в Maryland.) Технические руководства для связанных продуктов могут разделять множество информации, но отличаться в некоторых деталях; может быть удобно держать всю информацию о целом наборе сходных продуктов в одном документе, выбирая в момент показа или печати только те его части, что относятся к конкретному продукту. (Скажем, описание того, как заменять моторное масло может использовать один и тот же текст для большинства шагов, но содержать разные советы по удалению карбюратора, зависящие от модели двигателя.)

Для применения в подобных случаях SGML предоставляет конструкцию *отмеченных секций* (marked sections). В общем, как следует из приведенных примеров, она очевидно более удобна в производстве новых текстов, а не в кодировании уже существующих. Большинство пользователей схемы кодирования TEI не встретится с необходимостью использования отмеченных секций и может пропустить остаток этого описания. Однако, TEI DTD широко использует отмеченные секции, и этот раздел должен быть внимательно прочитан и понят каждым, кто хочет детально следить за изложением в главе **??**.

«Специальная обработка», предлагаемая в SGML для отмеченных секций, может быть нескольких типов, каждый из которых ассоциируется с одним из следующих ключевых слов:

[1] Подобные объявления объектов используются для расширения базового набора элементов TEI для прозы объявлениями, находящимися в **mystuff.dtd**.

INCLUDE Отмеченная секция должна быть включена в документ и нормально обработана.

IGNORE Отмеченная секция должна быть целиком проигнорирована; если SGML приложение генерирует данные на основании документа, отмеченная секция будет из них исключена.

CDATA Отмеченная секция может содержать строки символов, которые выглядят как метки SGML или ссылки на объекты, но которые не должны распознаваться в качестве таковых SGML анализатором. (Это Руководство использует такие отмеченные секции CDATA для включения примеров SGML разметки.)

RCDATA Отмеченная секция может содержать строки символов, которые выглядят как метки SGML, но которые не должны распознаваться в качестве таковых SGML анализатором. Ссылки на объекты, с другой стороны, могут присутствовать и будут распознаны и заменены как обычно.

TEMP Предложения, включенные в отмеченную секцию, являются временной частью документа; отмеченная секция в основном используется для обозначения их положения, так, чтобы они могли быть позже удалены или пересмотрены.

Когда в тексте появляется отмеченная секция, ей предшествует строка *открытия отмеченной секции* (marked–section start), содержащей одно или несколько ключевых слов из списка выше; ее конец отмечается строкой *закрытия отмеченной секции* (marked–section close). Вторая и последняя строки следующего примера являются открытием и закрытием игнорируемой отмеченной секции:

```
В таких случаях банк возместит клиенту все потери.
<![ IGNORE [
Ответственность ограничена $50,000.
]]>
```

Из ключевых слов наиболее важным для понимания TEI DTD являются **INCLUDE** и **IGNORE**; они используются для выборочного включения и исключения частей документа — или DTD — для подгонки его под конкретные условия (например, чтобы дать пользователю возможность выбирать части DTD, подходящие конкретному документу).

Однако, буквальное применение ключевых слов **INCLUDE** и **IGNORE** не очень помогает в настройке DTD по требованиям пользователя. (Например, чтобы вышеприведенный текст изменить так, чтобы включить исключаемую фразу, пользователю нужно будет вручную редактировать текст, сменив **IGNORE** на **INCLUDE**. С тем же успехом можно просто убирать или добавлять фразу в текст.) Но ключевые слова не обязаны быть представлены буквальными значениями; они могут быть ссылками на параметрические объекты. Скажем, в документе со многими фразами, которые должны быть включены в текст только в Maryland, каждая такая фраза может быть заключена в отмеченную секцию, чье ключевое слово представлено ссылкой на параметрический объект под названием **Maryland**. Предыдущий пример будет выглядеть так:

```
В таких случаях банк возместит клиенту все потери.
<![ %Maryland; [
Ответственность ограничена $50,000.
]]>
```

Когда параметрические объекты таким образом используются для управления отмеченными секциями DTD, внешний файл DTD обычно содержит объявления по умолчанию. Если пользователь хочет изменить умолчания (например, включив секции для Maryland), то для этого достаточно добавить соответствующее объявление в подмножество DTD.[1]

Теперь можно лучше понять примеры объявлений параметрических объектов, приведенные в конце предыдущего раздела. Объявления

```
<!ENTITY % TEI.prose 'INCLUDE'>
<!ENTITY % TEI.extensions.dtd SYSTEM 'mystuff.dtd'>
```

имеют эффект *включения* в DTD всех секций, отмеченных как относящиеся к прозе, так как во внешнем файле DTD все такие секции являются отмеченными и управляются параметрическим объектом **TEI.prose**. Они также изменяют объявленное по умолчанию значение объекта **TEI.extensions.dtd** (которое обычно объявлено пустой строкой) так, что он включает в DTD файл **mystuff.dtd**.

10 Картина в целом

Удовлетворяющий стандарту SGML документ состоит из нескольких частей, не все из которых рассмотрены в этой главе, и многие из которых пользователь этого Руководства может игнорировать. Для полноты картины может быть полезно рассмотреть взаимосвязь этих частей.

SGML документ состоит из SGML *пролога* (prolog) и *тела документа* (document instance). Пролог содержит *объявление SGML* (SGML declaration), описываемое ниже, и *определение типа документа*, содержащее объявления элементов и объектов, как было описано выше. Разные программные системы могут иметь различные способы ассоциирования тела документа с прологом; в некоторых случаях, например, пролог может быть «зашит» в используемую программу и невидим для пользователя.

10.1 Объявление SGML

Объявление SGML указывает основные данные об используемом диалекте SGML, такие, как набор символов, коды ограничителей SGML, длину идентификаторов и т.д. Его содержимое для типов документов, совместимых с TEI, излагается в главах **??** и **??**. Обычно объявление SGML существует в виде скомпилированных таблиц в SGML процессоре и пользователю не видимо.

[1]Это потому, что объявления в подмножестве DTD читаются раньше тех, что содержатся в файле DTD, а силу имеет объявление, считанное первым. Это вкратце описано в разделе 8.

10.2 DTD

Определение типа документа задает тип документа, которым проверяется экземпляр документа. Так же, как и объявление SGML, оно может существовать внутри SGML процессора, или быть ассоциировано с ним не видимым для пользователя образом, или требовать только задания названия типа документа для проверки документа.

В самом простом виде определение типа документа состоит просто из определения базового типа документа (возможно, еще одного или нескольких определений параллельных типов документов), которое предваряет тело документа. Например:

```
<!DOCTYPE my.dtd [
    <!-- здесь находятся все объявления для MY.DTD  -->
    ...
]>
<my.dtd>
    Это тело документа типа MY.DTD
</my.dtd>
```

Обычно определение типа документа содержится в отдельном файле и активизируется ссылкой на него, как в этом примере:

```
<!DOCTYPE tei.2 system "tei2.dtd" [
]>
<tei.2>
   Это пример документа типа не модифицированного TEI.
</tei.2>
```

Здесь текст определения типа документа TEI.2 не задается в явном виде, а SGML процессору указывается прочитать его из файла с системным идентификатором, заключенным в кавычки. Как показано в примере, квадратные скобки можно указывать, даже если в них ничего не заключено.

Часть, заключаемая в квадратные скобки, известна как *подмножество объявления типа документа* (document type declaration subset) или «подмножество DTD». Ее назначение — указать изменения, производимые в указанном DTD:

```
<!DOCTYPE tei.2 SYSTEM "tei2.dtd" [
     <!ENTITY tla "ТрехБуквенное Сокращение">
     <!ELEMENT my.tag - - (#PCDATA)>
     <!-- тут идут прочие специальные объявления или
          переопределения -->
]>
<tei.2>
   Это пример документа типа модифицированного TEI.2, который
   может содержать <my.tag>специальные элементы</my.tag> и
   ссылаться на обычные объекты типа &tla;.
</tei.2>
```

В этом случае активное определение типа документа включает сначала содержимое

подмножества DTD, и затем — содержимое файла, указанного после ключевого слова **SYSTEM**. Этот порядок важен, так как SGML учитывает только первое объявление объекта. Следовательно, в этом примере объявление объекта **tla** в подмножестве DTD будет иметь приоритет над любым объявлением того же объекта в файле **tei2.dtd**. Совершенно нормально для SGML иметь несколько объявлений объекта; это — обычный метод разрешить пользователю модифицировать DTD. (Элементы, напротив, не могут объявляться более чем один раз; если бы в файле **tei2.dtd** содержалось объявление элемента **<my.tag>**, SGML анализатор объявил бы об ошибке.) Комбинирование и расширение определений типа документа TEI обсуждается далее в главе **??**.

10.3 Тело документа

Телом документа является собственно содержание документа. Оно содержит только текст, разметку и ссылки на обычные объекты, и, таким образом, не может содержать новых объявлений. Удобным способом построения больших документов в модульном виде может быть объявление в подмножестве DTD объектов для отдельных частей или модулей таким образом:

```
<!DOCTYPE tei.2 [
     <!ENTITY chap1 system "chap1.txt">
     <!ENTITY chap2 system "chap2.txt">
     <!ENTITY chap3 "-- еще не написано --">
]>
<tei.2>
<teiHeader> ... </teiHeader>
<text>
    <front> ... </front>
    <body>
     &chap1;
     &chap2;
     &chap3;
     ...
</body>
</text>
</tei.2>
```

В этом примере DTD, содержащийся в файле **tei2.dtd**, расширен объявлениями объектов для каждой части работы. Первые два являются системными объектами, ссылающимися на файлы, в которых находятся тексты соответствующих частей; третий — пустой, показывающий, что текст еще не написан (или можно использовать объект с пустым значением). В теле документа ссылки на объекты &chap1; и т.п. будут разрешены анализатором, давая требуемое содержимое. Понятно, что файлы с главами не должны содержать никаких объявлений элементов, списков атрибутов или объектов — только размеченный текст.

11 Использование SGML

Существует множество программных средств для создания, проверки и обработки SGML документов. Здесь описываются только некоторое из них. Сердцем большинства продуктов служит *синтаксический анализатор* SGML, то есть, программный модуль, принимающий определение типа документа и генерирующий систему проверки любого документа с этим DTD. Выходом анализатора, в простейшем случае, является просто «да» (экземпляр документа правилен) или «нет». Большинство анализаторов, кроме того, выдают новую версию документа в *канонической форме* (canonic form), обычно с добавленными закрывающими метками и разрешенными ссылками на объекты) или сформатированную в соответствии с заданием пользователя. Эта форма может быть затем использована другими частями программной системы (более или менее тесно связанными с анализатором) для выполнения дополнительных функций, таких, как структурное редактирование, форматирование или управление базой данных.

Структурный редактор (structured editor) является, в своем роде, интеллектуальным текстовым процессором. Он может использовать информацию, выделенную из обработанного DTD, для того, чтобы подсказывать пользователю, какие элементы требуются в разных местах документах по мере редактирования. Он также может сильно упростить задачу подготовки документа, например, автоматически вставляя метки.

Форматировщик (formatter) использует размеченный экземпляр документа для генерации его печатной формы. Многие типографские различия набора, такие, как использование различных начертаний шрифта или его размера, тесно связаны со структурными различиями текста, так что форматировщики могут использовать знания, заложенные в описательной разметке. Можно, кроме того, определять структуру разметки для форматирующей программы при помощи параллельной структуры документа.

Системы управления текст–ориентированными базами данных обычно используют инвертированные индексы, указывающие в документ или его подразделения. Можно организовывать поиск вхождений некоторого слова или образца внутри документа или его части. Осмысленное подразделение входных документов будет, безусловно, тесно связано с делением, заданным описательной разметкой. Таким образом, системам текстовых баз данных несложно использовать размеченные в SGML документы. Большое количество исследовательской работы в настоящее время в области расширения возможностей существующих (не текстовых) систем управления базами данных для использования преимуществ, даваемых явной SGML разметкой структурированной информации.

Гипертекстовые (hypertext) системы поддерживают ассоциативные связи внутри и между документами. Снова строительные блоки, необходимые такой системе, являются также и базовыми блоками SGML: возможность идентифицировать и связывать отдельные элементы документа вытекает из методов работы SGML. Явно размечая связи вместо использования закрытого программного обеспечения, создатели гипертекстов могут быть уверены, что создаваемые ими ресурсы надолго будут доступны. Все, что требуется для загрузки SGML документа в гипертекстовую систему, — это процессор, умеющий корректно интерпретировать метки SGML, например, описанный в главе **??**.