УДК 004.624

ТЕХНОЛОГИЯ АВТОМАТИЗИРОВАННОГО СОЗДАНИЯ ЭЛЕКТРОННЫХ ДОКУМЕНТОВ OOXML-ФОРМАТА

Дунько Ю.С., Поточняк Я.В.
Одесский национальный политехнический университет, Украина.

Рассматриваются вопросы автоматизированного создания электронных документов. Предлагаются подходы к генерации электронных документов на основании xml-шаблонов. Рассматриваются алгоритмы создания документов на основе данного подхода. Сравнения с другими подходами и выделение данного на их фоне.

Введение

Деятельность любого предприятия сопровождается созданием множества документов: ежедневно создается и подписывается множество приказов и распоряжений, договоров и протоколов, писем и факсов. Многие из этих документов - однотипные по своей форме, но при этом приходится создавать их снова. Поэтому создаются типовые заготовки, бланки и формы для тех видов документов, которые приходится создавать чаще всего. При необходимости можно открыть такие документы в текстовом редакторе, внести нужные изменения и сохранить новую копию. На создание однотипных электронных документов уходит много времени. Особенно это проявляется при формировании документа на основании стороннего источника данных с многочисленным количеством записей. Поэтому автоматизация создания электронных документов с последующей интеграции с другими системами является актуальной задачей.

Ранее авторами была разработана система переноса содержимого таблиц документов DOC-формата в реляционную БД. В тоже время, обработка документов, структура которых создана самим пользователем, обладает высокой вероятностью возможных ошибок, уменьшение которой возможно лишь при автоматизированном создании заготовок документов для будущего заполнения пользователями [1].

В настоящий момент существует множество решений по автоматизации. Такую автоматизацию можно осуществить даже на уровне развитого текстового процессора, но часто он привязан к определённому текстовому процессору [2]. К примеру, на уровне библиотеки, или подключаемого модуля – это «*livedocx*», но это решение доступно только в Windows, привязано к Visual Studio[3]. Подобные решения могут быть встроены в другие программные комплексы, но они также узко ориентированы [4], т. к. не учитывают: извлечение данных из третьих систем или из баз данных (БД), инерционность данных, дублирование части документа, к примеру, строки с данными в таблице. Поэтому целью нашей работы стало развитие выше указанных решений за счет обеспечения кроссплатформенности и упрощения процесса создания документов и сокращения его ресурсоёмкости.

1 Общее представление решения

Предлагается модуль для автоматизации генерации электронных документов, в частности, решить задачу автоматической генерации текстовых документов, в которых основная часть данных формируется из БД.

Для формализации описания документа предложена следующая структура его шаблона (рис. 1).

Определены следующие связи между компонентами данной структуры: шаблон (1) состоит из документа (2) и метаданных (3):

$$t = \langle d, m \rangle, \tag{1}$$

$$d = \{p\}, \tag{2}$$

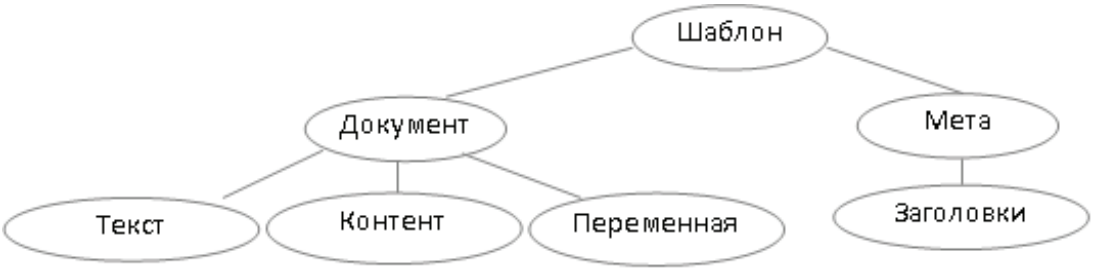

Рисунок 1. Пример структуры шаблона

$$m = \langle z, r \rangle, \tag{3}$$

Метаданные включают в себя заголовки (4) к создаваемому документу и некоторые свойства «r». Документ может включать в себя: текст «t», контент «c», и переменные «v». Для программного и математического обобщения данные структуры были обобщены в параграф (5), который, в свою очередь, может представлять одну из перечисленных структур. Но контент, в отличие от текста и переменных, является сложной и иерархической структурой данных, поэтому он может быть представлен следующим образом (6).

$$z = \{d\}, \tag{4}$$

$$\{p\} \in \{\langle t, c, v \rangle\}, \tag{5}$$

$$c \in \{p\}, \tag{6}$$

Текст в шаблоне является статической частью, которую необходимо включить в создаваемый документ. Контент представляет собой источник данных и методы для управления данными. Также он служит некоторым пространством имен для данных, находящихся внутри него. Но необходимо манипулировать данными, поэтому нужны переменные. Исходя из этого, переменные разбиты на следующие группы: переменные, которые хранят в себе выбор пользователя (необходимы для вставки изменяемых данных внутрь документа); переменные управления контентом.

Контент является двойным именованным массивом, к элементам которого можно обращаться не только по индексу, но и по имени, по итерации. Для переменных контент представляет следующий список услуг:

1. Обращение к контенту верхнего уровня, то есть к тому внутри которого он находиться, но обратной связи не предусмотрено.
2. Перейти к следующей итерационную строку
3. Узнать: можно ли сделать следующий шаг
4. Взять значение из текущей строки по имени (при работе с базой данных это название колонки)
5. Взять следующее значение из строки
6. Узнать есть ли следующее значение в строке
7. Взять последнее созданное значение
8. Узнать номер итерации строки.

В дальнейшем появилась необходимость записывать несколько операций в одной переменной, поэтому был внедрён механизм работы с контекстом на уровни внутренних регулярных выражений. Для этого текст внутри переменной должен начинаться с символа «$», а текст будет рассматриваться как набор инструкций, где символ «#» — взятие последнего элемента из контента; «?» — взятие следующего значения из текущей строки; «-» — обращение к контенту верхнего уровня. Этот же механизм был внедрён в обработчик *sql* запросов. Это было необходимо для формирования *sql* запросов, с использованием текущего значение верхнего уровня. К примеру, нам необходимо вывести список студентов, сгруппированным по группам и факультетам. Для этого контентом верхнего уровня будет список факультетов, а групп контент нижнего уровня. Но чтобы получить список групп, нам необходимо знать значение факультета.

Но данных только из БД недостаточно, поэтому контент характеризуется типом получения данных. В настоящий момент это могут быть данные из БД или данные, установленные пользователями. Шаблон представлен в виде *xml*-документа. Рассмотрим пример простого шаблона:

```
<template>
  <data type="content">
    <properties>
      <property name="content-type">user-settings</property>
      <property name="inject">departments</property>
    </properties>
    <data><text>заголовок перед определением факультета</text></data>
    <data type="var">
      <properties><property name = "var-type">settings</property></properties>
       <text>$faculty</text>
    </data>
    <data><text>заголовок пере списком студентов</text></data>
    <data type="content">
      <properties>
        <property name = "content-type">database</property>
           <property name = "sql">select * from students where gr_name='-#'</
property>
      </properties>
      <data type="var">
          <properties><property name = "var-type">content</property></properties>
          <text>iteration</text>
      </data>
      <data type="var">
        <properties><property name = "var-type">content</property></properties>
        <text>next</text>
      </data>
      <data><text><![cdata[some text ...]]></text></data>
      <data type="var">
        <properties><property name = "var-type">content</property></properties>
        <text>step</text>
      </data>
    </data>
    <data type=»var»>
      <properties><property name = «var-type»>content</property></properties>
      <text>step</text>
    </data>
  </data>
</template>
```

Предлагается алгоритм для создания документа со следующими шагами:

1. Создание обработчика параграфов с глобальными данными: все пользовательские переменные, контент пользователя и соединение с БД, для обработки БД.
2. Обработка каждого параграфа в соответствии с типом данного параграфа:
   – Если это контент, то он создаёт точно такой же обработчик, но с данным необходимыми для текущего контента. Результаты обработки добавляют в итоговый документ.
   – Если это переменная, то происходит проверка типа переменной. При типе управления контентом, производиться манипуляция с контентом в соответствии с указаниями в переменной. Если же эта переменная хранит выбор пользователя, то в итоговый документ добавляется соответствующее значение из пользовательских переменных.
   – Если это текст, то это значение добавляется в итоговый документ.

Таким образом, мы получили простое в управлении, быстрое и легко интегрируемое решение для автоматизированного создания текстовых документов. Также было принято решения для расширения возможностей создаваемых документов, поскольку текстовый формат имеет крайне узкое

направления для работы. Поэтому его необходимо расширить для работы с офисными документами. В первую очередь это документы формата «*doc*», но сложность работы с ними заключается в том, что все библиотеки для записи данных внутрь документа платные и ограниченны в рамках *Windows* решений. Поэтому мы остановились на новом формате «*docx*».

2 Расширение решения на офисные форматы документов

«*Open Document Format*» (*OOXML*) является открытым форматом документа и поддерживается всеми текстовыми редакторами, а так же является *zip*-архивом с набором *xml*-файлов, что позволяет работать с данным документом без сложных библиотек. Внутри данного формата уже заложены элементы управления содержимым данного документа. Одним из таки является «умная вставка» данный элемент характеризуется: идентификатором ресурсов, названием элемента, набором пользовательских настроек. Он вставляется в документ на уровне текста.

Следующим элементом управления является «собственная разметка». Данный элемент характеризуется теми же свойствами что и «умная вставка», но допускает дочерние элементы.

Анализ данного формата начинается с определения файлов в архиве: контент документа, его дополнительные свойства: наборы стилей, таблицы шрифтов, и т.д. В начале обработки данного формата анализируются файлы описания, которые всегда присутствуют в архиве, и определяем какие именно файлы нам необходимо изменить.

В следующем шаге трансформируется входной *xml* так, чтобы его можно было передать на обработку обработчику текстового формата: все теги и их содержимое меняются на текстовый параграф за исключением тегов разметки. Тег «собственной разметки» превращается в описание контента, а тег «умной вставки» превращается в переменную. Текстовый обработчик создает *xml*-файл, который заменит файлы во входном *docx*-архиве. Т.к. работа выполняется с текстом, данный алгоритм быстро обрабатывает документ, а требуемые ресурсы незначительны, т. к. созданная оболочка занимает больше ресурсов, чем сам процесс создания документа.

Для упрощения процесса создани разметки предложено создавать документ с условными обозначениями:

1. «#»имя контента»{» — начало контента
2. «#}»имя контента»» — конец контента
3. «$»имя переменной»{» — объявление переменной

Предлагается хранить *xml* описание в файле «*MetaInf.xml*». Но в данном случае переменная в шаблоне связана с множеством реальных переменных. Вначале необходимо найти в тексте данные условные обозначения. После этого, необходимо вставить в *xml* файлы описание пользовательской разметки в соответствии с данными из «*MetaInf.xml*». Для описания контекста в «MetaInf.xml» необходимо хранить информацию от том, перед каким элементом в *xml* файле необходимо открывать контент. Это происходит из-за того что мы определяем описатель на самом нижнем уровне — тексте, а он может описывать строку в которой находиться.

«*MetaInf.xml*» представлен в виде *xml*-документа. Рассмотрим простой пример:

```
<meta-inf>
  <source>
    <content type = «database»>
     <properties><property name=»sql»>SELECT * FROM students WHERE gr _ name='-#'</
property></properties>
    </content>
    <content type = «user-settings»>
      <properties><property name = «inject»>departmens</property></properties>
    </content>
  </source>
  <controls>
    <vars name = «some-name»>
      <var format = «%s» type = «text»>
```

```
        <properties><property name = «var-type»>content</property></properties>
        <text>iteration</text>
      </var>
      <var>
        <properties><property name = «var-type»>content</property></properties>
        <text>next</text>
      </var>
    </vars>
  </controls>
</meta-inf>
```

Выводы

На основе продолженных структур данных и алгоритмов разработана программа генерации текстовых документов. Были проведены эксперименты при создании документов, использующихся в деканате студентов: ведомость контроля успеваемости и итоговая ведомость. В результате было продемонстрировано, что предложенное решение сократило время на подготовку ведомостей (более 500) и сократило ресурсы на печать, т. к. печать выполнялась на матричном АЦПУ-принтере.

Литература

[1] Дунько Ю.С., Марулин С.Ю. Структурно-синтаксический анализ электронных документов с табличными структурами // Труды I всеукраинской научно-технической конференции студентов, аспирантов и молодых учёных «Информационные управляющие системы и компьютерный мониторинг». г. Донецк, 19-21 мая, 2010.

[2] С начала было слово (Word). Материалы с официального ресурса. Электронный ресурс. Режим доступа: http://www.delcomp.ru/005_5.html

[3] Text Control GmbH2 Inc.. Материалы с официального ресурса. Электронный ресурс. Режим доступа: http://www.livedocx.com/

[4] Генерация документов в системе управления требованиями Borland/Microfocus CaliberRM. Материалы с официального ресурса. Электронный ресурс. Режим доступа: http://www.interface.ru/home.asp?artId=24184