View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

Модел. и анализ информ. систем. Т. **18**, № **3** (2011) 125–143

УДК 004.91

Модульная модель мультимедийного документа и особенности ее программной реализации

Январев В.И.

МГУ им. М.В. Ломоносова

e-mail: janvarev@gmail.com

получена 2 сентября 2010

Ключевые слова: мультимедийный документ, архитектура программного обеспечения, система управления сайтом (CMS), модель мультимедийного документа, HTML

Предлагается подход к созданию модульной модели мультимедийного документа, сочетающей расширяемость списка доступных редактору объектов с возможностью их свободного комбинирования. Для предложенной модели предлагаются методы сравнения версий мультимедийных документов и осуществления поиска по их коллекциям. Рассматривается архитектура соответствующей программной системы управления мультимедийными документами с акцентами на переносимость в другие программно-аппаратные платформы и на обеспечение обратной совместимости. Описываются две программные системы, созданные с использованием разработанной архитектуры: система управления мультимедийными сайтами Fuzzle CMS и редактор бизнес-диаграмм FLAscheme.

1. Введение

Среди многочисленных проблем, связанных с машинной обработкой информации, можно выделить такую важную задачу, как построение систем создания и управления документами.

Существенной характеристикой документа является вид представленной в нем информации. К базовым видам представления информации можно отнести текст, графику, звук, видео. Гомогенные документы состоят преимущественно (или исключительно) из данных одного базового вида. Соответственно можно говорить о текстовых, аудио-, видеодокументах. Более сложным является **мультимедийный документ**, который создается путем композиции разных видов представления информации. Современный мультимедийный документ обладает также таким свойством, как **интерактивность**, т.е. возможность изменения внешнего вида документа пользователем в процессе просмотра, например возможность увеличения фотографии или изменения типа графика, визуализирующего статистические данные.

Примерами мультимедийных документов могут служить веб-сайты, интерактивные модели естественнонаучных явлений, интерактивные презентации и пр.

Рассмотрим основные проблемы, представленные в научных работах, посвященных исследованию мультимедийных документов.

Существенное место занимает проблема описания временной последовательности отображения фрагментов документа. Решаемые практические задачи могут формулироваться, например, следующим образом: в какой последовательности демонстрировать текстовую, графическую информацию, сопровождающую видео, с учетом того, что пользователь может перейти в любую часть этого видео. Для решения этой проблемы разрабатываются языки, базирующиеся на стандарте HyTime [3] и использующие другие научные разработки, например ITL (Interval Temporal Logic) [17]. В работах по данной тематике рассматриваются принципы расширения исходного стандарта или языка описания документа. Например, в работе [15] предпринята попытка описать возможный язык спецификаций на базе ITL, который позволил бы осуществлять композицию элементов мультимедийного документа не в императивном, а в декларативном стиле. В работе [12] подобную задачу планируется решать с помощью введения отношения типа «следование» (в оригинальной работе — «follow») между элементами мультимедийного документа. В работе [14] рассматривается созданный комплекс программ Madeus, специально построенный для упрощения проектирования мультимедийного документа, обладающего значительной протяженностью во времени.

Другой класс научных работ фокусируется на задачах поиска в мультимедийном документе. В работе [16] рассматривается архитектура системы управления содержимым мультимедийного документа, позволяющая осуществлять поиск. Авторы такого рода работ часто исходят из допущения, что документ состоит только из текстовой информации и изображений. Например, в работе [16] считается, что каждый элемент мультимедийного документа содержит текстовую метаинформацию, по которой и осуществляется поиск. В диссертации [13] мультимедийный документ рассматривается как совокупность текста и изображений. Отдельно рассматривается вопрос об интеллектуальном анализе этих изображений с получением их характеристик, на основании которых будет осуществляться поиск.

Небольшой класс работ — обычно это работы 1995—2000 гг., а также стандарты ISO, специфичные для мультимедийных документов, такие как ODA [4] и HyTIME [3], — посвящены вопросам, касающимся проблем стандартизации представления видео- и аудиоданных, в частности вопросам использования стандартизированных аудиокодеков и видеокодеков для кодирования элементов мультимедийного документа. Например, работа [11] в значительной своей части посвящена проблеме адекватной степени сжатия данных в процессе подготовки мультимедийного документа для последующей передачи его по сети.

При анализе работ заметен возрастающий интерес к проблематике адаптации мультимедийного документа под разные условия презентации; например, в работах [18, 10] рассматривается проблема построения модели мультимедийного документа, который может адекватно отображаться как на обычном компьютере, так и на мобильном устройстве с пониженным быстродействием и разрешением экрана.

В центре внимания настоящей статьи находится разработка структуры (модели) мультимедийного документа, оптимальной с точки зрения минимизации временных и интеллектуальных затрат, необходимых пользователям для создания и редактирования документов. Особенность современного положения дел заключается в том, что ответственными за подготовку и обновление мультимедийных документов, как правило, являются специалисты в некой области деятельности, не связанной с программированием: преподаватели, секретари, литературные редакторы, администраторы сайтов и т.д. Соответственно система управления документами должна предоставлять этим группам пользователей возможность конструировать документ из различных мультимедийных элементов, не предъявляя при этом требований к знанию языка программирования.

Множество рассматриваемых мультимедийных документов ограничено документами со свободным просмотром, т.е. такими, которые состоят в основном из распределенных на странице объектов, слабо взаимодействующих между собой. Пользователи могут свободно перемещаться по страницам документов и взаимодействовать с любым из расположенных на них мультимедийных объектов.

В статье будут рассмотрены основные подходы к построению модели мультимедийного документа со свободным просмотром, выявлены их достоинства и недостатки с точки зрения построения системы, наиболее удобной в работе для непрограммирующего пользователя. Предложен новый подход, объединяющий два важных свойства рассмотренных подходов: возможность неограниченно расширять документ новыми видами мультимедийных объектов и возможность гибко конструировать документ из объектов различных видов.

Также в статье будут рассмотрены основные архитектурные решения, необходимые для реализации предложенной модели в программной системе и обеспечивающие переносимость системы на другие программно-аппаратные комплексы и обратную совместимость версий ядра системы. Рассмотрены проблемы реализации поиска и сравнения версий мультимедийного документа. Предложенные архитектурные решения были использованы при разработке системы управления мультимедийными сайтами Fuzzle CMS [1] и редактора бизнес-диаграмм FLAscheme [2].

2. Требования к системам управления мультимедийными документами

Существенные ограничения на структуру и функциональность системы управления мультимедийными документами накладывает используемая в ней **модель документа**, под которой мы будем подразумевать формальное описание объектов, которые могут быть включены в мультимедийный документ, правила их композиции и визуализации. Примером модели мультимедийного документа может служить стандарт языка HTML [5]. В нем четко указано, какие теги могут входить в состав документа, какие у них могут быть атрибуты (параметры), какие из тегов могут вкладываться друг в друга и как следует такое включение визуализировать (правила композиции и визуализации).

Введем основные понятия, связанные с разработкой и использованием программ-

ных средств, реализующих модель мультимедийного документа. Программные средства можно разделить на системы визуализации и системы редактирования документов. Эти системы могут быть как совмещенными (как, например, в редакторе Microsoft Word), так и раздельными (в случае HTML). Будем рассматривать только совмещенные системы, поскольку при определенных усилиях со стороны программистов две раздельные системы можно объединить в одну совмещенную. Аналогично совмещенную систему можно превратить в две раздельных. Соответствующую совмещенную систему будем называть **системой управления мультимедийными документами**.

В случае, если система допускает доопределение новых видов объектов путем интеграции дополнительного программного кода для их визуализации и редактирования, такую систему будем называть **модульной**, а соответствующие дополнительные программные средства — **модулями**. Основную часть такой системы естественно назвать **ядром**. Для написания программного кода модулей могут привлекаться сторонние разработчики.

Для модульных систем управления мультимедийными документами можно выделить три основные группы пользователей.

1. Конечные пользователи, в задачи которых входит просмотр мультимедийного документа. Одним из важных требований этой группы является возможность поиска по содержимому мультимедийного документа и по коллекциям документов.

2. Пользователи-редакторы — лица, ответственные за подготовку и обновление мультимедийного документа. Как правило, это специалисты в некой области деятельности, не связанной с программированием — например, это могут быть преподаватели, секретари, литературные редакторы, администраторы сайтов и т.д. Данная группа заинтересована в:

   (a) возможности композиции объектов мультимедийного документа и настройки их внешнего вида в визуальном режиме;

   (b) отсутствии необходимости в программировании;

   (c) возможности самостоятельного добавления новых модулей в систему;

   (d) возможности коллективной работы над документом (как правило, такая работа не осуществляется в режиме «реального времени», когда несколько редакторов одновременно правят один документ, однако возможность визуализации последовательных правок путем сравнения версий была бы весьма полезной).

3. Разработчики модулей, создающие подключаемые модули, которые в дальнейшем могут использоваться пользователями-редакторами. Требованиями разработчиков являются:

   (a) обратная совместимость — сохранение работоспособности всех модулей после обновления ядра системы (необходимого для устранения ошибок и добавления новых возможностей).

(b) отсутствие необходимости обрабатывать типовые (общие для всех модулей) метаданные объектов (такие как положение на странице, стандартные настройки стилей); их обработка должна осуществляться на уровне ядра системы.

Кроме того, должна быть предусмотрена возможность использования программного кода системы в различных программно-аппаратных комплексах (требование кроссплатформенности) и ее интеграции в более крупные системы документооборота (требование внедряемости).

3. Подходы к созданию моделей мультимедийного документа

Можно выделить два основных подхода к построению модели мультимедийного документа. Первый из них можно назвать **«вложенные объекты»**. Модель здесь описывается как неограниченное вложение друг в друга объектов-контейнеров. Примерами такого рода моделей могут служить: HTML-модель, MXML-язык проектирования программных интерфейсов в IDE Adobe Flex Builder [7], модель документов Adobe Flash.

Для HTML-документа, как это следует из описания стандарта, визуализируемый объект описывается одним тегом. Например, для визуализации изображения используется тег <img />, таблицы — <table>...</table> и т.д. В случае, когда тег включает в себя другие теги, для каждого из них также создается объект, местоположение которого зависит от местоположения родительского объекта. Визуализируемыми данными объекта являются данные, содержащиеся внутри тега, а также частично атрибуты тега (например, объект-ссылка описывается тегом <a> с атрибутом href). Принципы визуализации тегов описаны в стандарте языка.

Схожим образом устроена модель документов Adobe Flash. Объектом в таком документе является либо некая базовая фигура, такая как форма («shape»), кривая Безье, линия, растровое изображение («bitmap») и т.д., либо составной объект (объект-символ), который включает в себя несколько базовых фигур или других символов. Данными объекта для базовых фигур будут являться параметры их отображения (количество опорных точек для построения, цвет, прозрачность и пр.). Дополнительные классы символов могут быть созданы путем группировки нескольких символов или базовых фигур — например, может быть создан класс символов «рисованное лицо» путем объединения нескольких кривых, овала и пр.

Отметим общие черты рассмотренных моделей. Количество типов объектов в этом случае ограничено (в случае Adobe Flash, несмотря на расширяемость, все они должны сводиться к базовым фигурам). Соответственно пользователь-редактор не может расширять подобную систему дополнительными типами объектов с помощью модулей сторонних разработчиков. Для описания любого дополнительного интерактивного взаимодействия редактору потребуется знание языка программирования (JavaScript для HTML-документов, ActionScript для Flash). Таким образом, пользователь, не владеющий навыками программирования, не сможет в рамках этих

моделей создать достаточно интерактивный документ, который сможет нетривиально реагировать на действия пользователя (под тривиальным взаимодействием мы подразумеваем, например, переход по HTML-ссылке).

Второй подход основан на отделении содержания мультимедийного документа от его внешнего вида. В английском языке подход к построению подобного рода систем обычно имеет название «model — view» (**данные — визуализация**). Этот подход используется в CMS и предполагает модульную структуру системы управления мультимедийным документом. Каждый программный модуль обычно соответствует одному типу объектов.

Например, в каждой CMS существует модуль визуализации текстовых страниц. Его данными являются содержимое текстовой страницы, ее название, дата последнего изменения и т.п. За визуализацию отвечает программная часть модуля, которая объединяет все перечисленные элементы в единую структуру (например, в фрагмент HTML-документа) и визуализирует ее. Другим примером может быть модуль «лента новостей». Здесь визуализируемая часть формируется из последовательности однотипных объектов типа «новость», каждый из которых, в свою очередь, включает стандартный набор данных: заголовок, текст новости, дату публикации, источник, фотоиллюстрации и т.п.

Поскольку структура данных у объектов одного типа одинакова, для их хранения часто используются реляционные СУБД. При инсталляции нового модуля создается таблица базы данных, предназначенная для хранения списков данных соответствующих объектов. Также программист модуля может разработать специальный пользовательский интерфейс редактирования, позволяющий пользователю-редактору добавлять новые объекты в список и изменять данные уже существующих.

Подход «данные — визуализация» предполагает, что в каждый момент времени активен только один модуль, соответственно визуализирован только один объект или группа объектов одного типа. Произвольное комбинирование объектов разного типа невозможно. Чтобы преодолеть это ограничение, на практике используются следующие подходы:

- Для каждого объекта предусматривается максимальное количество настраиваемых свойств. Например, для каждой новости модуля «лента новостей» должны быть определены: текст новости, дата, маленькая картинка, большая картинка, правила выравнивания для маленькой картинки, для большой картинки, правила отображения даты и т.д. Такой подход ведет к увеличению программного кода модуля, снижению прозрачности структуры визуализируемых данных.

- В качестве параметров модуля используются данные не базовых типов (текстовые, изображения), а код, написанный на некотором мета-языке (например, на языке HTML или в wiki-разметке). Такой подход ведет к усложнению программного кода модуля, который должен включать интерпретатор мета-языка, и необходимости специальной подготовки пользователя-редактора (знание языка HTML или wiki-форматирования).

В рамках настоящей работы предлагается **смешанный подход к построению модели документа**. Его основные принципы заключаются в следующем:

1. Каждая страница документа представляет собой одноуровневый список объектов; страницы в нашем представлении независимы и вводятся исключительно для удобства навигации в документе так же, как существуют страницы HTML или страницы Word-документа.

2. Объекты, расположенные на странице, могут свободно позиционироваться (у них могут изменяться координаты, ширина, высота). Также у объектов может изменяться ряд стандартных свойств, например, цвет фона, обрамления.

3. Каждый объект характеризуется (кроме координат и общих свойств) типом и данными, предназначенными для визуализации. Объекты одинакового типа визуализируются с помощью вызова одного и того же программного компонента визуализации, подключаемого к программной системе (модуля); в качестве параметров визуализации передаются данные объекта.

4. Редактирование данных происходит путем вызова компонента редактирования, специфичного для данного типа объекта и также содержащегося в программном коде модуля.

5. Список типов объектов может пополняться путем подключения пользователем-непрограммистом дополнительных модулей к системе управления документами.

Предложенная модель и построенная на ее основе система управления мультимедийными документами объединяют два основных преимущества подходов «вложенные объекты» и «данные — визуализация»: 1) конструирование документа путем свободного позиционирования различных мультимедийных объектов на странице документа; 2) расширение системы за счет добавления в нее объектов новых типов (подключения новых модулей). Отметим, что в том и в другом случае от пользователя-редактора не требуется знание языка программирования.

4. Архитектурные решения и методы работы с мультимедийным документом

Рассмотрим реализацию предложенной модели в системе управления мультимедийными сайтами Fuzzle CMS. Система Fuzzle CMS реализована как клиент-серверное приложение. Клиентская часть системы является основной — она написана на языке ActionScript 3 для исполнения внутри виртуальной машины Flash. Выбор языка реализации был обусловлен в том числе тем, что виртуальная машина Flash доступна для многих программно-аппаратных комплексов. Серверная часть системы является вспомогательной и предназначена для хранения данных, таких как авторизационные данные пользователей, содержимое страниц, файлы подключаемых модулей, конфигурационные параметры системы и т.д.

В этом разделе будет показан способ реализации предложенной модели в программной системе, в том числе описан формат представления мультимедийных данных и структура визуализирующих их модулей.

Также в разделе будет рассмотрен ряд архитектурных решений, направленных на обеспечение долгосрочного стабильного функционирования системы. Эти решения включают, в первую очередь, меры по обеспечению:

- переносимости системы на другие программно-аппаратные платформы;
- обратной совместимости версий ядра системы.

Для формального описания предложенной модели в системе Fuzzle CMS используется подмножество языка XML. Страница в ней описывается как одноуровневый список тегов <block>, каждый из которых соответствует одному объекту. Каждый тег, в свою очередь, содержит следующие универсальные атрибуты:

- `id` — уникальный идентификатор объекта;
- `type` — тип объекта;
- `x, y, width, height` — координаты и размеры объекта;
- универсальные параметры стиля: фон, цвет и толщина границ и др.

Данные объекта, предназначенные для визуализации, записываются между открывающим и закрывающим элементами конструкции <block>...</block>. Они представляют собой корректный XML-код, который подается на вход программному компоненту визуализации данных соответствующего модуля. Таким образом, XML-код страницы формируют несколько записанных последовательно одно- или разнотипных тегов <block>. Ниже приведен пример XML-кода страницы с размещенными на ней двумя объектами типа «текстовая надпись»:

```
<page>
  <block id="1" type="com.fuzzle.standart.BlockText"
         x="90" y="401" width="195" height="211">
    <text>Методы параллельного программирования.</text>
  </block>
  <block id="2" type="com.fuzzle.standart.BlockText"
         x="500" y="401" width="195" height="211">
    <text>Обработка текстовой информации.</text>
  </block>
</page>
```

Модули, визуализирующие те или иные виды данных, могут быть свободно подключены к ядру системы управления. Существует предустановленная библиотека базовых видов объектов, таких как форматированный текст, изображение, кнопка и др.

Каждый модуль включает два файла со скомпилированным байт-кодом визуальных компонентов *визуализации* и *редактирования данных*. Компонент визуализации используется, когда документ открыт пользователем-читателем в режиме просмотра, и служит для отображения данных. Он может включать интерактивные инструменты навигации по данным, выбора одного из нескольких режимов их отображения и т.п. Компонент редактирования используется, когда пользователь-редактор вносит изменения в состав данных в момент создания документа или его последующего обновления. За переключение между режимами и непосредственное изменение месторасположения объектов и настроек их визуальных стилей отвечают компоненты библиотеки визуализации и редактирования, входящие в состав ядра системы. О них будет сказано чуть ниже.

Компонент визуализации данных каждого модуля реализует программный интерфейс IBlock, компонент редактирования — программный интерфейс IBlockEditor (см. рис. 1). Эти интерфейсы идентичны, однако характер их использования несколько различен. В программном интерфейсе IBlock команда loadFromXML заставляет компонент визуализировать данные, сохраненные в XML-коде страницы, а команда saveToXML возвращает текущие данные объекта, расположенного на странице. В компоненте редактирования команда loadFromXML также визуализирует данные объекта, сохраненные в XML-коде страницы, однако делает это в пользовательском интерфейсе редактирования, где в данные могут быть внесены изменения. Команда saveToXML выполняется после того, как пользователь подтвердит совершенные изменения. Она записывает текущие данные объекта в XML-код страницы. Команда «Отмена» в пользовательском интерфейсе редактирования позволяет возвратиться к неизмененному варианту данных.

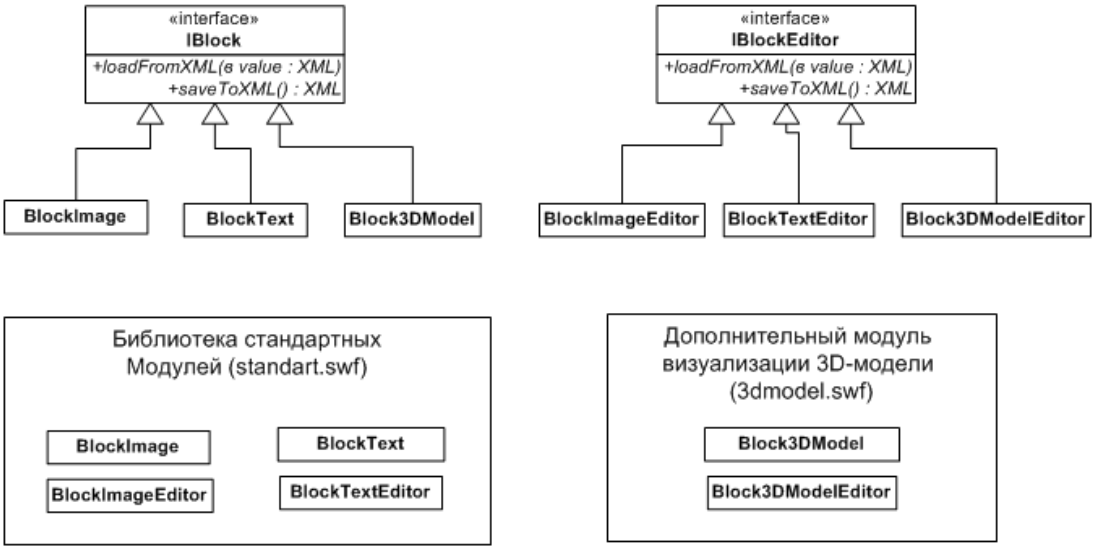

Рис. 1. Диаграмма классов подключаемых модулей

Компонент редактирования, входящий в состав модуля, предназначен исключительно для обновления содержательных данных объекта. За настройку таких

универсальных метаданных, как координаты, ширина, высота объекта, стандартные настройки стилей (фон, границы и др.), отвечают компоненты библиотеки визуализации и редактирования, входящей в ядро системы. Это позволяет снять с разработчика модуля задачу обработки универсальных метаданных и тем самым убыстрить и упростить разработку новых модулей.

Библиотека визуализации и редактирования страниц обеспечивает отображение и редактирование содержимого страниц документа. Она включает в себя следующие программные классы:

- визуальный контейнер «редактируемое полотно» (EditableCanvas);
- визуальную панель настройки свойств редактируемого полотна (ECPanel);
- универсальный контейнер, в который загружаются компоненты визуализации каждого из мультимедийных объектов, расположенных на странице (BlockAll);
- всплывающий контейнер, в который загружается компонент редактирования активного (выделенного пользователем) мультимедийного объекта (PopupGlobalEditor);
- управляющий класс (ECBlockController).

Управляющий класс ECBlockController реализован по шаблону проектирования Singleton (это означает, что в системе может существовать единственный экземпляр этого класса, который будет доступен из любой точки системы). Класс ECBlockController обладает следующей функциональностью:

- загружает модули в соответствии с содержанием XML-кода активной страницы;
- возвращает список наименований всех объектов и их текстовых идентификаторов;
- создает визуальные компоненты для всех расположенных на странице объектов по их идентификаторам (функция getObject(name:String));
- создает визуальный компонент редактирования для активного объекта по его идентификатору (функция getEditor(name:String)).

Взаимоотношения между этими классами легче всего выявить при описании процесса визуализации страницы.

1. У компонента EditableCanvas вызывается функция loadFromXML, которая считывает из хранилища данных XML-код страницы.
2. При вызове функций для каждого тега <block> из XML-кода страницы создается объект типа BlockAll и добавляется в контейнер EditableCanvas, после чего:

(a) по параметрам x, y, width, height устанавливается ширина и высота этого контейнера;

(b) по универсальным параметрам стиля у него задаются цвет фона, границ и т.д.;

(c) по текстовому идентификатору создается компонент визуализации, загруженный из соответствующего модуля с помощью вызова функции getObject объекта ECBlockController;

(d) созданный компонент визуализации добавляется в контейнер BlockAll и заполняет собой все его пространство;

(e) у созданного компонента визуализации вызывается функция loadFromXML (которая доступна, поскольку компонент реализует интерфейс IBlock), заставляющая компонент визуализировать содержательные данные объекта;

(f) в случае, если для блока данного вида задана анимация появления на странице, запускается процедура анимирования контейнера BlockAll.

Процесс редактирования страницы имеет ряд сходных черт с процессом визуализации, главным образом потому, что при редактировании страницы размещение мультимедийных объектов на ней происходит в визуальном режиме: пользователь перетаскивает объекты мышью, меняет их размеры, потянув за «горячие точки» в углах рамки объекта. Таким образом, он сразу может оценить конечный результат своей работы. Если пользователь имеет права на изменение страницы, система выводит в правом нижнем углу страницы кнопку «Редактировать», после нажатия на которую вызывается функция перехода полотна EditableCanvas в состояние редактирования toEditMode. В процессе исполнения этой функции:

1. Создается панель настройки свойств редактируемого полотна ECPanel. Панель ECPanel визуализируется поверх страницы и может свободно перемещаться по ней (плавающая панель). На панели расположены функции вставки/удаления/копирования объектов, доступных для добавления в документ. Их список получается от объекта ECBlockController.

2. Вокруг всех контейнеров BlockAll, расположенных на полотне, отображается рамка с «горячими точками» в углах, потянув за которые можно поменять расположение и размеры объекта. В состоянии редактирования интерактивное взаимодействие компонентов визуализации с "мышью" блокируется.

Когда пользователь хочет отредактировать данные или стандартные настройки стилей того или иного объекта, он выделяет этот объект на полотне и вызывает соответствующий ему компонент редактирования (в Fuzzle CMS для этого необходимо дважды щелкнуть мышью по объекту). После этого:

1. Создается всплывающий компонент редактирования активного объекта PopupGlobalEditor, который выводится как модальное окно, блокируя взаимодействие с остальной частью экрана.

2. Компонент BlockAll собирает полную информацию о себе, объединяя содержательные данные, полученные от своего компонента визуализации данных (через функцию saveToXML), и параметры стандартных настроек стиля (фона, границ и др.).

3. Собранные данные передаются компоненту PopupGlobalEditor через функцию loadFromXML.

4. На основании сведений о типе редактируемого объекта компонент PopupGlobalEditor через функцию объекта ECBlockController создает соответствующий компонент редактирования, который помещается на первую вкладку модального окна PopupGlobalEditor. На вход ему через функцию loadFromXML интерфейса IBlockEditor передается текущее содержимое редактируемого объекта.

5. Параметры стандартных настроек стиля передаются в расположенные на вспомогательных вкладках модального окна PopupGlobalEditor универсальные для всех видов объектов компоненты редактирования настроек стиля.

По окончании редактирования происходит обратный сбор информации в XML — как содержательных данных, полученных от компонента редактирования через функцию saveToXML, так и настроек стиля отображения объекта. После сбора этой информации она передается в текущий компонент BlockAll, который производит визуализацию.

Когда пользователь решает окончить редактирование полотна, он отдает команду о сохранении страницы (в Fuzzle CMS для этого в правом нижнем углу страницы предусмотрена кнопка «Сохранить»). Полотно EditableCanvas обходит все содержащиеся в нем компоненты BlockAll, собирает от каждого из них в виде XML содержательные данные и метаданные о расположении на странице и настройках стилей, формирует из них результирующий XML-код страницы. После чего полотно EditableCanvas переходит в режим визуализации.

Таким образом, предложенная модель была реализована в программной системе Fuzzle CMS. Изложенные принципы организации библиотеки визуализации и редактирования страниц позволяют решить одновременно несколько задач:

1. Обеспечить модульность системы управления документами (каждый новый вид мультимедийных данных — новый модуль).

2. Стандартизировать процесс редактирования различных видов мультимедийных объектов.

3. Выполнить часть настроек, касающихся позиционирования объекта, в визуальном режиме, наиболее удобном для пользователя-редактора.

4. Упростить процесс разработки модулей, обеспечив обработку универсальных метаданных объектов (размеров, расположения, оформления) средствами библиотеки визуализации.

При разработке программной системы Fuzzle CMS был принят ряд архитектурных решений, направленных на обеспечение долгосрочного стабильного функционирования системы. Эти решения включают меры по обеспечению:

Переносимости системы. Важным качеством системы управления мультимедийными документами является принципиальная возможность ее интеграции в состав более крупных систем документооборота, корректное функционирование на различных программно-аппаратных платформах.

Обратной совместимости версий ядра системы. Жизненный цикл программной системы в перспективе долгосрочного функционирования предусматривает ее регулярное обновление: устранение ошибок, расширение функциональности. Кроме того, особенностью системы, работающей с мультимедийными документами, является непрерывное расширение числа подключаемых модулей, визуализирующих различные виды данных, по мере развития соответствующих технологий. Для разработки подключаемых модулей могут привлекаться сторонние разработчики. Меры, направленные на поддержание обратной совместимости, обеспечивают корректное функционирование всех модулей после обновления версии программного ядра системы. В противном случае неизбежны дополнительные временные и финансовые затраты на обновление (или даже полную замену) отдельных модулей.

Архитектурные решения, направленные на решение указанных задач, касаются преимущественно клиентской части системы Fuzzle CMS. Они предполагают выделение следующих дополнительных библиотек программных компонентов:

- библиотеки сервисных платформозависимых и платформонезависимых компонентов;
- библиотеки прокси-классов.

Библиотека сервисных компонентов. Для корректной работы некоторых мультимедийных объектов требуется обращение к процедурам и функциям используемой платформы, например к файловой системе или к базе данных. Поскольку рассмотренная выше библиотека визуализации страниц должна оставаться платформонезависимой, обеспечение связи с платформой берут на себя специальные *сервисные платформозависимые компоненты*, такие, например, как файловый менеджер, редакторы меню и медиа-галерей (обращающиеся к используемой базе данных), модуль интернет-магазина (включающий каталоги товаров, форму заказа, корзину) и некоторые другие. Эти компоненты требуют модификации при переходе с одной платформы на другую.

В библиотеку сервисных компонентов входят также *платформонезависимые компоненты*. В основном к ним относятся стандартные визуальные компоненты: кнопки, картинки, текстовые надписи, — которые используются разработчиками при создании модулей, визуализирующих новые виды мультимедийных данных. Необходимость реализации таких компонентов на уровне ядра системы обусловлена тем, что все они должны выглядеть и управляться одинаково, вне зависимости от того, в каком подключаемом модуле они будут использоваться. Набор сервисных платформонезависимых компонентов может расширяться по мере развития системы.

Библиотека прокси-классов. Прокси-классы предназначены для изоляции программного кода модулей от ядра системы управления, что позволяет обновлять

ядро, не нарушая работоспособности системы в целом. Они представляют собой программные интерфейсы ко всем вышеперечисленным сервисным компонентам (платформозависимым и платформонезависимым), которые внедряются в тело модуля и устанавливают динамическую связь с соответствующими компонентами ядра на этапе выполнения. Структура прокси-компонента остается неизменной, она задается один раз в момент инсталляции соответствующего компонента ядра и не зависит от его текущей версии. Это позволяет обновлять ядро системы без перекомпиляции модулей, что является несомненным преимуществом как для разработчиков, так и для пользователей-редакторов мультимедийного документа. Кроме того, все важные проверки (соответствие типов, согласование имен функций), могут быть выполнены еще на этапе компиляции модуля, чем обеспечивается стабильность работы всей системы.

Общая архитектура системы отражена на рис. 2.

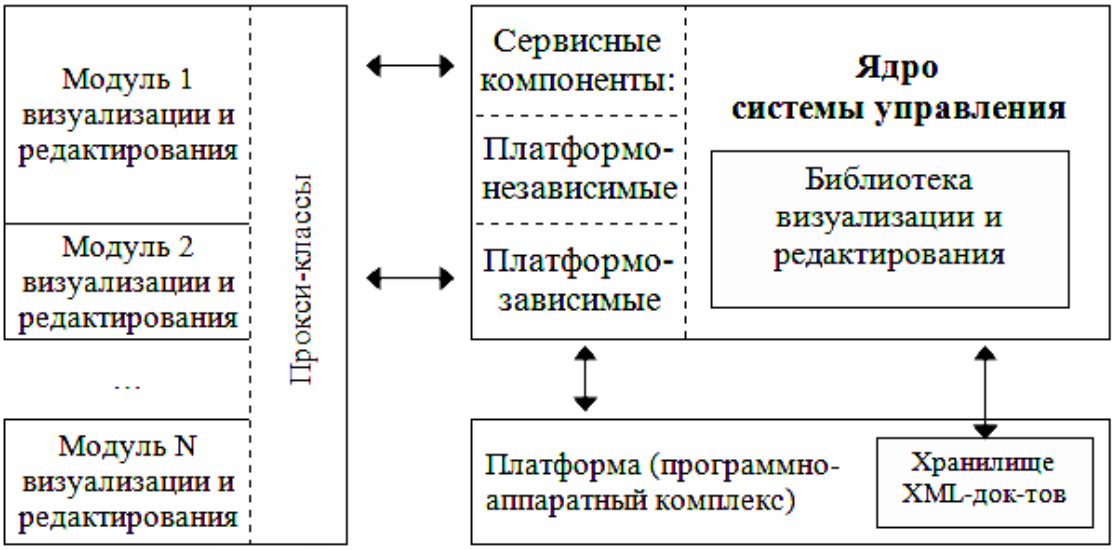

Рис. 2. Архитектура системы управления мультимедийными документами

В заключение раздела рассмотрим разработанные методы решения следующих задач обработки мультимедийных данных, важных для конечных пользователей и авторов документов:

- Задачи сравнения версий мультимедийного документа.
- Задачи осуществления поиска по мультимедийному документу и коллекциям документов.

Важным качеством системы управления мультимедийным документом является поддержка коллективной работы над документом. Это подразумевает, в том числе, возможность **сравнения версий документа**, выполненных в различное время различными пользователями, и управления версиями. В отличие от текстовых данных, для которых алгоритмы сравнения версий в настоящее время разработаны и успешно применяются на практике, для мультимедийных данных таких алгоритмов

не существует. Отдельной задачей является визуализация результатов сравнения документа, так чтобы она была легка для восприятия.

Основную сложность при реализации метода сравнения версий мультимедийного документа представляет то обстоятельство, что список различных видов данных, визуализируемых в мультимедийном документе, принципиально открыт: он может пополняться новыми видами мультимедийных данных по мере развития системы и мультимедийных технологий. Поэтому для реализации метода сравнения версий документа необходимо разграничить *уникальные* (обусловленные видом визуализируемых данных) и *универсальные*(общие для всех) свойства расположенных на странице мультимедийных объектов, и обрабатывать каждые из них по отдельности.

Для этого каждому объекту на странице должен быть присвоен уникальный идентификатор, неизменный во всех версиях документа. При сравнении двух версий страницы алгоритм проходит по всем объектам страницы, и они выводится на экран в специальном форматировании:

- если объект присутствует в старой версии страницы, но отсутствует в новой, он выводится с пометкой «Удален»;
- если объект отсутствует в старой версии страницы, и присутствует в новой, он выводится с пометкой «Создан»;
- если объект присутствует и в старой, и в новой версии, то
  - если были изменены размеры и местоположение объекта на странице, он выводится с пометкой «Перемещен»;
  - если были изменены данные объекта, он выводится с пометкой «Данные изменены» и возможностью переключения между старой и новой версиями данных. Если данный тип объекта поддерживает специальный программный интерфейс «Сравнение», то вызывается соответствующая функция визуализации, принимающая на вход обе версии данных объекта.

Еще одной важной задачей является реализация **поиска по мультимедийным данным**. В случае размещения коллекции мультимедийных документов в сети Интернет должна быть предусмотрена возможность ее индексации поисковыми машинами.

В настоящее время не существует эффективных поисковых алгоритмов, позволяющих осуществлять поиск информации в нетекстовых источниках [16, 13]. В указанных работах предлагается снабжать все входящие в состав документа мультимедийные объекты текстовыми описаниями, а затем осуществлять поиск по этим описаниям. Однако ручное заполнение таких текстовых описаний сопряжено со значительными временными затратами, поэтому необходимо проработать дополнительные автоматические способы получения релевантной информации для поиска. В системе Fuzzle CMS предусмотрены следующие способы ручного и полу-/автоматического формирования текстовых описаний:

- в случае, если часть данных мультимедийного объекта имеет текстовую форму (подписи к фотографиям, графикам, названия аудиотреков, товаров в магазине, географические названия на карте и т.п.), эти данные автоматически формируют текстовое описание (при необходимости разработчик определяет процедуры фильтрации/преобразования данных, чтобы в текстовое описание попадали только наиболее релевантные для поиска сведения);

- каждому виду мультимедийных объектов разработчик определяет ключевые слова, описывающие характер представленных данных, например: «3D-панорама», «калькулятор стоимости»; эти ключевые слова используются по умолчанию, но в случае необходимости могут быть скорректированы пользователем;

- наконец, в пользовательских интерфейсах редактирования у всех объектов предусмотрено стандартное поле «пользовательское текстовое описание», которое заполняется пользователем вручную.

Ручное заполнение текстовых описаний сопряжено со значительными временными затратами, поэтому предполагается, что основная часть работ по их формированию должна выполняться в автоматическом режиме, при этом текстовые описания, заполненные пользователем вручную, должны обладать наибольшей релевантностью при поиске.

Поскольку система Fuzzle CMS работает в сети Интернет, в ней была также создана подсистема преобразования содержимого мультимедийного документа в набор HTML-страниц, что делает возможным поиск по документу с помощью популярных поисковых систем, таких как Яндекс, Google, Рамблер. Каждая HTML-страница соответствует странице исходного документа и автоматически составляется из текстовых описаний всех расположенных на ней объектов, оказываясь, таким образом, пригодной для индексации поисковыми машинами.

5. Программная реализация модели

Модель мультимедийного документа и предложенные архитектурные решения были реализованы в двух программных системах: системе управления мультимедийными сайтами Fuzzle CMS [1] и встраиваемом редакторе бизнес-диаграмм FLAscheme [2].

Поскольку все архитектурные решения рассматривались в статье на примере их реализации в системе Fuzzle CMS, сразу перейдем к рассмотрению особенностей реализации модели в системе FLAscheme.

Система FLAscheme, так же как и Fuzzle CMS, представляет собой клиент-серверное приложение. Она предназначена для создания и редактирования интерактивных бизнес-диаграмм внутри страниц независимых систем управления сайтами, таких как WackoWiki, WikkaWiki, MediaWiki. В данном случае используемые CMS (системы управления сайтами) рассматриваются как платформы, под которые каждый раз происходит адаптация встраиваемого редактора FLAscheme.

Клиентская часть системы FLAscheme также использует библиотеку визуализации и редактирования страниц. Важным отличием системы FLAscheme от Fuzzle

CMS является фиксация набора доступных модулей, встроенных в клиентскую часть системы. Отказ от динамического подключения дополнительных модулей вызван позиционированием системы как редактора бизнес-диаграмм. Набор доступных объектов ограничен здесь тремя видами: текстовыми блоками, стрелками и достаточно большим по объему комплектом иконок. Все перечисленные объекты достаточно просты, поэтому здесь появилась возможность отказаться от использования прокси-компонентов и платформозависимых программных компонентов. Следует, однако, отметить, что в случае необходимости технических сложностей с добавлением новых модулей не возникло бы.

Еще одной особенностью FLAscheme была необходимость адаптации серверной части (формата хранения данных о диаграмме) к стандартам используемой CMS, поскольку каждая из них по-своему хранит данные о содержимом страниц.

Таким образом, опыт внедрения показал, что предложенная модель документа и архитектурные решения могут быть использованы не только для создания полнофункциональных систем управления мультимедийными документами, но и для реализации специализированных встраиваемых редакторов с ограниченной функциональностью.

6. Заключение

В статье был предложен и обоснован новый подход к построению модели мультимедийного документа. Он сочетает в себе основные достоинства наиболее распространенных в настоящее время подходов «вложенные объекты» (возможность комбинирования объектов в пространстве страницы документа) и «данные — визуализация» (возможность добавлять новые типы объектов в систему управления документом путем подключения дополнительных программных модулей).

Предложенная модель была реализована в программной системе. Рекомендуемые архитектурные решения по ее реализации также явились предметом настоящей статьи. Рассмотрены меры по обеспечению переносимости системы на другие программно-аппаратные комплексы и обратной совместимости версий ядра системы. Предложены методы обработки мультимедийных данных: метод сравнения версий мультимедийного документа и метод создания его текстовой версии, пригодной для поиска и индексации поисковыми системами.

На базе принципов, предложенных в работе, были созданы две программные системы: система управления мультимедийными сайтами Fuzzle CMS и встраиваемый редактор бизнес-диаграмм FLAscheme. Обе системы были реализованы с использованием одного набора библиотек, что свидетельствует о гибкости предложенной архитектуры. С момента разработки системы Fuzzle CMS (март 2009 г.) на ней было создано более 25 сайтов.

Список литературы

1. Описание и документация Fuzzle CMS. URL: http://fuzzle-cms.ru/.

2. Описание и документация FLAscheme. URL: http://flascheme.com/.

3. ISO 10744, Information Technology - Hypermedia/Time-based Structuring Language (Hy-Time), 1992.

4. ISO 8613, Information Technology - Open Document Architecture (ODA) and interchange format, 1989–1996.

5. RFC 1866 Hypertext Markup Language - 2.0, 1995.
URL: http://tools.ietf.org/html/rfc1866

6. Adobe Flash CS4. Официальный учебный курс. М.: Эксмо, 2009.

7. Кейзоун Ч., Лотт Дж. Программирование с использованием Adobe Flex. СПб.: Питер, 2009.

8. Boll S., Klas W., Westermann U. A Comparison of Multimedia Document Models Concerning Advanced Requirements // Technical Report - Ulmer Informatik-Berichte. No. 99-01

9. Boll S., Klas W., Westermann U. Multimedia Document Formats - Sealed Fate or Setting Out for New Shores In Multimedia? // Tools and Applications, 2000. N 11. Issue 3.

10. Boll S. MM4U - A framework for creating personalized multimedia content. 2003.

11. Gehne R., Jesshope C. Interactive Multimedia for Dummies. 2008.

12. Hauser J. Realization of an Extensible Multimedia Document Model // Proceedings of Eurographics Multimedia, 1999.

13. Jeong Ki Tai. A common representation for multimedia documents // Dissertation Prepared for the degree of doctor of philosophy. University of North Texas, 2002.

14. Jourdan M., Layaida N., Roisin C., Sabry-ismail L., Tardif L. Madeus, an Authoring Environment for Interactive Multimedia Documents // ACM Multimedia, 1998.

15. King P. R., Modelling multimedia documents. Electronic publishing,1995. V. 8(2 & 3). P. 95–110.

16. Matellanes A., May A., Villegas P., Snijder F., Kobzhev A., Dijk E.O. An architecture for multimedia content management // Integration of Knowledge, Semantics and Digital Media Technology, 2005. EWIMT 2005. The 2nd European Workshop on the (Ref. No. 2005/11099).

17. Moszkowski B. Executing Temporal Logic Programs. Cambridge University Press, 1986.

18. Pellan B., Concolato C. Adaptation of scalable multimedia documents // Proceeding of the eighth ACM symposium on Document engineering (DocEng '08). ACM, New York, NY, USA, 2008. P. 32-41.

Module-based Multimedia Document Model: Basic Principles and Software Architecture

Janvarev V.I.

Keywords: multimedia document model, document models, multimedia, HTML, Flash, software architecture

This paper describes an approach to multimedia documents modeling. The approach allows a non-programming user to compose pages from objects of different types (blocks) and to install new types of objects (modules) made by third-party programmers. We propose original methods for document version comparison and multimedia document indexing and searching. The paper describes an architecture of a Content Management System for the model proposed; the focus is to provide software portability and backward compatibility. We elaborated two software products using this architecture: Fuzzle CMS, multimedia CMS for Flash websites, and FLAscheme, the embeddable business diagram editor.

Сведения об авторе:
Январев Владислав Игоревич,
МГУ им. М.В. Ломоносова, факультет ВМК,
аспирант