Федеральное агенство по образованию Российской Федерации
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет информатики
Кафедра прикладной информатики

УДК 681.03

ДОПУСТИТЬ К ЗАЩИТЕ В ГАК
Зав. кафедрой, проф., д.т.н.
_____ С.П.Сущенко
“__”_____2005 г.

Кудрявцев Иван Александрович

ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ ПЕРЕНОСИМОЙ СРЕДЫ ВЫПОЛНЕНИЯ ПРИЛОЖЕНИЙ ВИЗУАЛЬНОЙ ДИНАМИЧЕСКОЙ ОБЪЕКТНОЙ МОДЕЛИ (V.D.O.M.)

Дипломная работа

Научный руководитель — Ерохин А.Е.

Исполнитель, студ.гр.1401 — Кудрявцев И.А.

Томск — 2005

Реферат

Сведения об отчете: дипломная работа 46 с., 19 рис., 2 табл., 12 использованных источников литературы.

БАЗЫ ДАННЫХ, СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ, ВСТРАИВАЕМЫЕ СИСТЕМЫ, INTERNET, WEB, СИСТЕМЫ УПРАВЛЕНИЯ СОДЕРЖИМЫМ.

Цель работы: Проектирование и реализация переносимой среды выполнения приложений визуальной динамической объектной модели (V.D.O.M.)

Метод исследования: Получение информации о предметной области и ее анализ.

Результаты работы: Разработан программный комплекс, позволяющий выполнять приложения, реализованные в рамках модели V.D.O.M., на различных программно-аппаратных платформах. Реализована утилита, осуществляющая преобразование данных из модели данных V.D.O.M. в формат, поддерживаемый переносимой средой выполнения. Предложен новый подход к разработке web-приложений на основе специализированных макро-объектов.

Содержание

1 Введение

Современные пользователи предъявляют повышенные требования к средствам разработки документов. Сложность верстки заставляет производителей искать новые пути интуитивно-понятных способов создания сложных документов. Факты таковы, что если десять лет назад пользователя устраивал обычный текстовый редактор, в котором форматирование можно было задавать только отступами и разделяющими элементами, то в настоящее время электронный документ имеет настолько сложный формат, что для формирования документов требуются специализированные приложения. В настоящее время отдельное место занимают документы, публикуемые в сети интернет. Особенность таких документов — высокая степень интерактивности, мультимедийность, неоднородность, поддержка обратной связи.

Создание документов для публикации в интернет производится при помощи специализированных языков разметки. Особенность этих языков разметки — достаточно простая структура, что позволяет создавать такие документы вручную при помощи тэгов разметки. С другой стороны, структура документа может быть комплексной, что затрудняет их создание и поддержку. Кроме того, для создания документа, поддерживающего рекомендуемые W3C[1][10] стандарты, требуются специализированные знания. Чем сложнее требуемая структура и дизайнерские решения, необходимые для документа, тем более сложным становится его создание для обычного пользователя. Для создания качественных стандартных документов привлекаются специалисты по Web-дизайну и информационному проектированию[9].

Многие игроки IT-индустрии предлагают решения, которые позволяют обычному пользователю создавать документы для публикации в интернет без применения языков разметки. Обычно, пользователю предоставляется мастер или редактор в стиле WYSIWYG (What You See Is What You Get), который позволяет создать публикуемый документ без использования специализированных знаний.

WYSIWYG-редакторы делятся на два больших класса: специализировнные локальные приложения и редакторы, встраиваемые в приложения Web. Первые обладают более широким набором свойств и возможностей нежели вторые, но главное преимущество вторых — отсутствие привязки к машине пользователя и мгновенная публикация на сайте. Вы можете модифицировать, добавлять и удалять документы без использования специализированного ПО, пользуясь только браузером. Очевидно,

[1] World Wide Web Consortium

что для мобильных пользователей эта особенность представляет удобство.

Еще один класс систем, который следует упомянуть — системы управления содержимым (Content Management Systems, CMS). Далее, слова "содержимое"и "контент"считаются эквивалентными. Такие системы ориентированы на документы определенных типов. Ключевой характеристикой документа является его тип. Документ описывается набором полей и имеет четкую структуру. Каждый документ проходит фазы создания, редактирования, публикации. Публикация документа осуществляется по некоторой форме. Системы управления контентом, в общем случае, позволяют создавать произвольные типы документов.

Задача вышеперечисленных приложений — снизить стоимость поддержки сайта и дать возможность обычному потребителю услуг самостоятельно осуществлять публикации.

Система V.D.O.M. (Visual Dynamic Object Model / Визуальная Динамическая Объектная Модель) — клиент-серверная система, предназначенная для создания и управления сайтом при помощи специализированных визуальных средств и инструментария. Центральную роль в системе занимает объект пользовательского интерфейса. Система имеет несколько десятков объектов, которые позволяют строить полнофункциональные web-сайты. Система V.D.O.M. разработана CyberTronique inc. с целью помочь организациям и пользователям разрабатывать приложения для Web без знания языков разметки и навыков программирования.

Основным недостатком V.D.O.M. является несовместимость с популярными платформами интернет. Платформа выполнения V.D.O.M. — Microsoft Windows NT 5.0 и выше. Для использования технологии необходим MS SQL Server и MS Internet Informational Services с поддержкой ASP. Суммарная стоимость этого решения высока для большинства пользователей услуг интернет. Кроме того, поскольку основная платформа Web — Unix, то для большинства пользователей использование V.D.O.M. недопустимо в силу существующих архитектурных и программных решений. В этих условиях расширение рынка затруднительно, что привело к необходимости создания переносимой среды выполнения (V.D.O.M. CPE). Под переносимостью понимается возможность использования среды и приложений без необходимости изменения под конкретную аппаратно-программную платформу.

Основная задача — не дублировать всю функциональность, а создать среду выполнения приложений с эмуляцией вызовов исходной среды разработки. Среда выполнения должна функционировать самостоятельно, без использования внешних СУБД и Web-сервера.

Одно из требований проекта — открытость разработки, привлека-

тельность для организаций, ориентированных на использование открытых технологий (Open Source).

2 Технология V.D.O.M.

Технология V.D.O.M. представляет собой объектную систему построения Web приложений. Ключевые возможности технологии реализуются посредством набора предопределенных объектов, из которых конструируется документ. Для конструирования используется специализированное приложение, которое позволяет осуществлять визуальное размещение объектов на странице документа. Каждый элемент документа является объектом некоторого предопределенного типа.

В настоящее время V.D.O.M. не позволяет определять пользовательские объекты, а существующий набор объектов достаточен для построения "типичного" Web приложения для индивидуального пользователя и малого бизнеса.

Невозможность создания пользовательских объектов ограничивает сферу применения V.D.O.M. Например, для приложений электронной торговли существует классический набор компонентов. Компонент понимается как комплексный объект, возможно агрегированный. В условиях невозможности определения пользовательских объектов и компонентов клиент оказывается в ситуации, когда ему становится невыгодно использовать технологию V.D.O.M. для реализации приложения.

Другим сдерживающим фактором является зависимость от программно-аппаратной платформы (системы выполнения приложений). В настоящее время V.D.O.M. предоставляется клиентам не в индивидуальное пользование, а как услуга. Программно-аппаратные компоненты используются многими пользователями одновременно. При интенсивной нагрузке на сервер производительность приложения падает, даже если к этому приложению обращается малое число пользователей. Для улучшения качества обслуживания требуется наращивать мощь сервера или распределять нагрузку по нескольким серверам. В общем, пользователь зависит от поставщика услуг V.D.O.M. и не может получить приложение в личное пользование без больших материальных затрат. Другим недостатком V.D.O.M. является ориентированность на MS Internet Explorer, что не позволяет использовать V.D.O.M. клиентам, работающим с другими браузерами.

3 Обзор системы

Далее мы рассмотрим систему V.D.O.M. На рисунке 1 приведена схема системы.

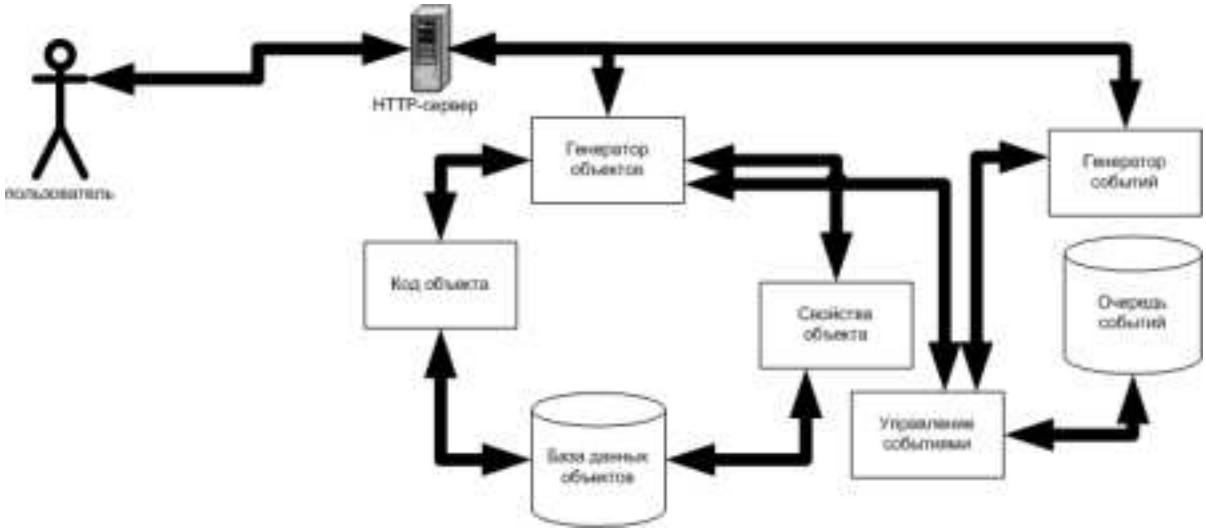

Рисунок 1 — архитектура V.D.O.M.

На рисунке 1 изображена концептуальная схема системы, менее важные части были опущены. Основной элемент системы — база данных объектов, которая содержит всю необходимую для работы системы информацию. В базе для каждого объекта приложения содержится два типа записей: запись о свойствах объекта и запись о коде объекта. На основе этой информации генератор объектов создает экземпляр объекта с заданными свойствами и функциональностью.

Другая важная подсистема — управление событиями. Благодаря ей возможно создавать “асинхронный web-интерфейс”, что актуально для web-приложений с высоким уровнем интерактивности и наличием обратной связи.

3.1 Система с точки зрения пользователя

Для создания сайта в системе V.D.O.M. пользователю достаточно браузера MS Internet Explorer 5.x и выше. Все операции выполняются в специализированном приложении V.D.O.M. — дизайнере. Дизайнер охватывает все стадии создания приложения, начиная от проектирования навигационной схемы приложения и заканчивая встраиванием сценариев в страницы. Основное назначение дизайнера — проектирование

web-страниц в режиме OnLine. Такая модель разработки удобна для мобильных пользователей. Одной из выдающихся особенностей дизайнера является асинхронный режим работы. В этом режиме потери времени пользователя минимальны. Клиент-серверное взаимодействие осуществляется параллельно действиям пользователя, не влияя на его работу.

Из других возможностей системы наиболее интересной является подсистема создания резервных копий и восстановления по требованию. Наличие такой функции позволяет легко переносить приложения между различными средами выполнения V.D.O.M., а также защищает данные пользователя от повреждения. Подсистема предоставляет несколько уровней резервного копирования, которые различаются по объему сохраняемой информации.

3.2 Система с точки зрения программиста

Анализ показывает, что в системе V.D.O.M. существует пять основных типов сущностей: приложения, объекты, страницы, сценарии, события. На них построена основная функциональность системы. Далее, каждая сущность будет рассмотрена детально.

3.2.1 Приложение

Система V.D.O.M. представляет собой многопользовательскую систему. Под термином "приложение" понимается логически организованная совокупность страниц (см. далее), предназначенная для решения некоторой задачи и принадлежащая некоторому пользователю. Приложение используется для разделения рабочих пространств пользователей. Каждое приложение обладает набором атрибутов, который задает свободу действий пользователя. Наиболее существенные атрибуты приложений — максимальное количество объектов и страниц. Эта сущность не имеет значения для V.D.O.M. CPE [2] и поэтому далее не рассматривается.

3.2.2 Объект

Объект — основная строительная единица приложения. Каждый объект принадлежит к определенному типу, который обладает свойствами, определяющими представление объекта и алгоритмом, который определяет параметрический (относительно значений свойств) вид объекта.

[2] в силу того, что V.D.O.M. CPE предназначена для выполнения какого-то одного приложения

Свойства объекта хранятся в специализированной базе данных объектов. При запросе объекта они загружается в память приложения. Для использования объекта необходим контейнер, который называется "страницей" (см. далее). Для программного управления свойствами объекта используются "сценарии" (см. далее).

Алгоритм объекта — код на языке программирования Visual Basic, который загружает объект из БД по идентификатору и формирует фрагмент DHTML, который соответсвует визуальному представлению объекта. Объект может транслироваться в HTML или DHTML, в зависимости от режима выполнения приложения. DHTML-режим необходим для создания приложений, поддерживающих подсистему управления событиями (см. далее).

В процессе исследования объектов были замечены определенные недостатки, которые ограничивают сферу применения. Основной недостаток — наличие единственной копии объекта для всего приложения. При изменении свойства объекта, происходит соответсвующее изменение БД. Таким образом, все пользователи видят измененный объект. Очевидно, что в многопользовательских системах должны существовать объекты время жизни которых ограничено пользовательской сессией. Копия такого объекта должна создаваться для каждой пользовательской сессии и изменение свойств объекта должно происходить в рамках сессии и не оказывать влияние на объект в БД (рис. 2).

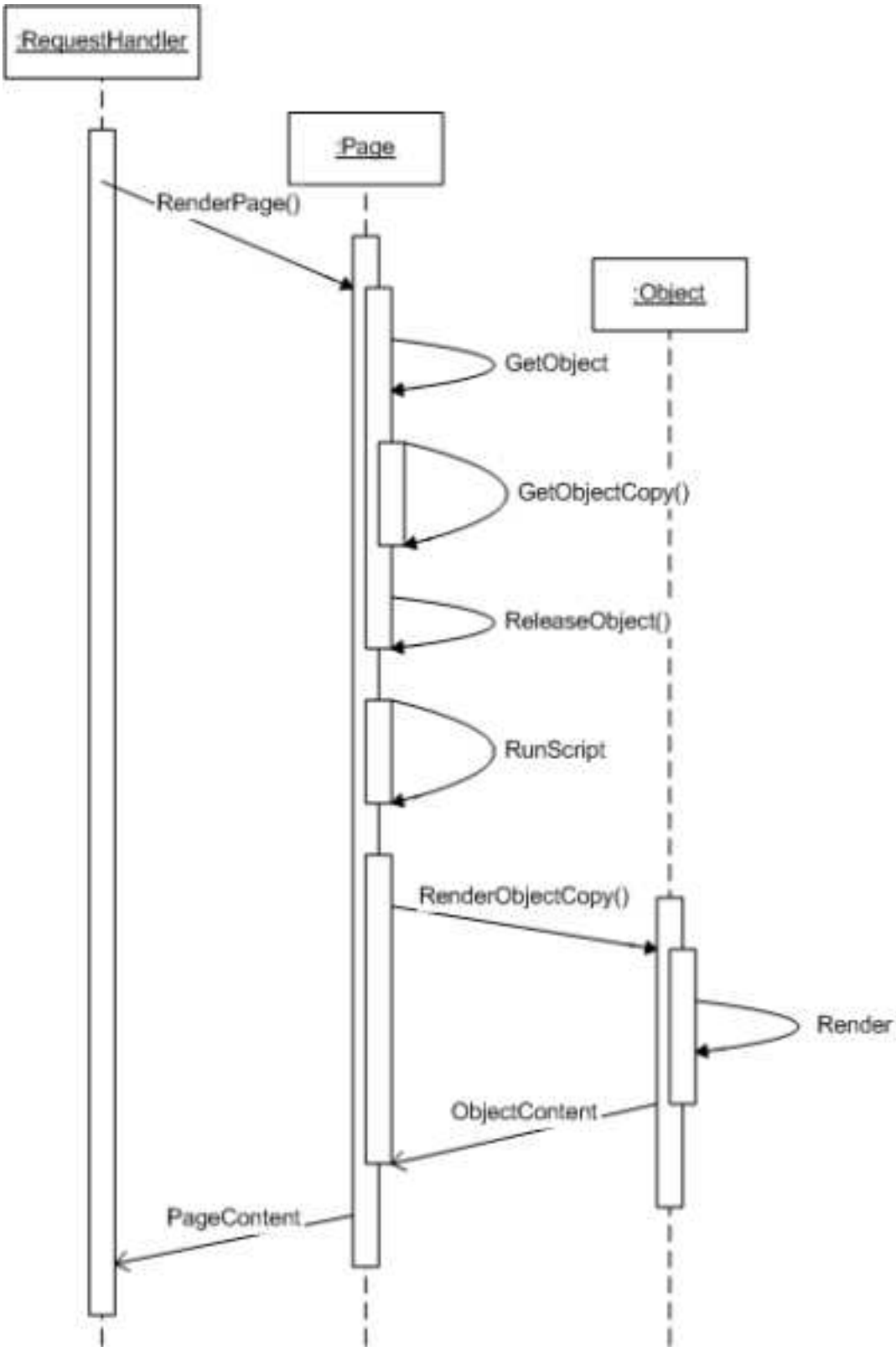
:RequestHandler
:Page
:Object
RenderPage()
GetObject
GetObjectCopy()
ReleaseObject()
RunScript
RenderObjectCopy()
Render
ObjectContent
PageContent

Рисунок 2 — Генерация страницы по запросу.

3.2.3 Страница

Объекты размещаются на страницах, каждая страница — контейнер, который содержит следующие части: объекты, сценарии управления объектами (см. далее).

Внутреннее представление страницы — сценарий на языке ASP, позволяющий сгенерировать документ.

Действия системы по генерации изображены ниже (рис. 3)

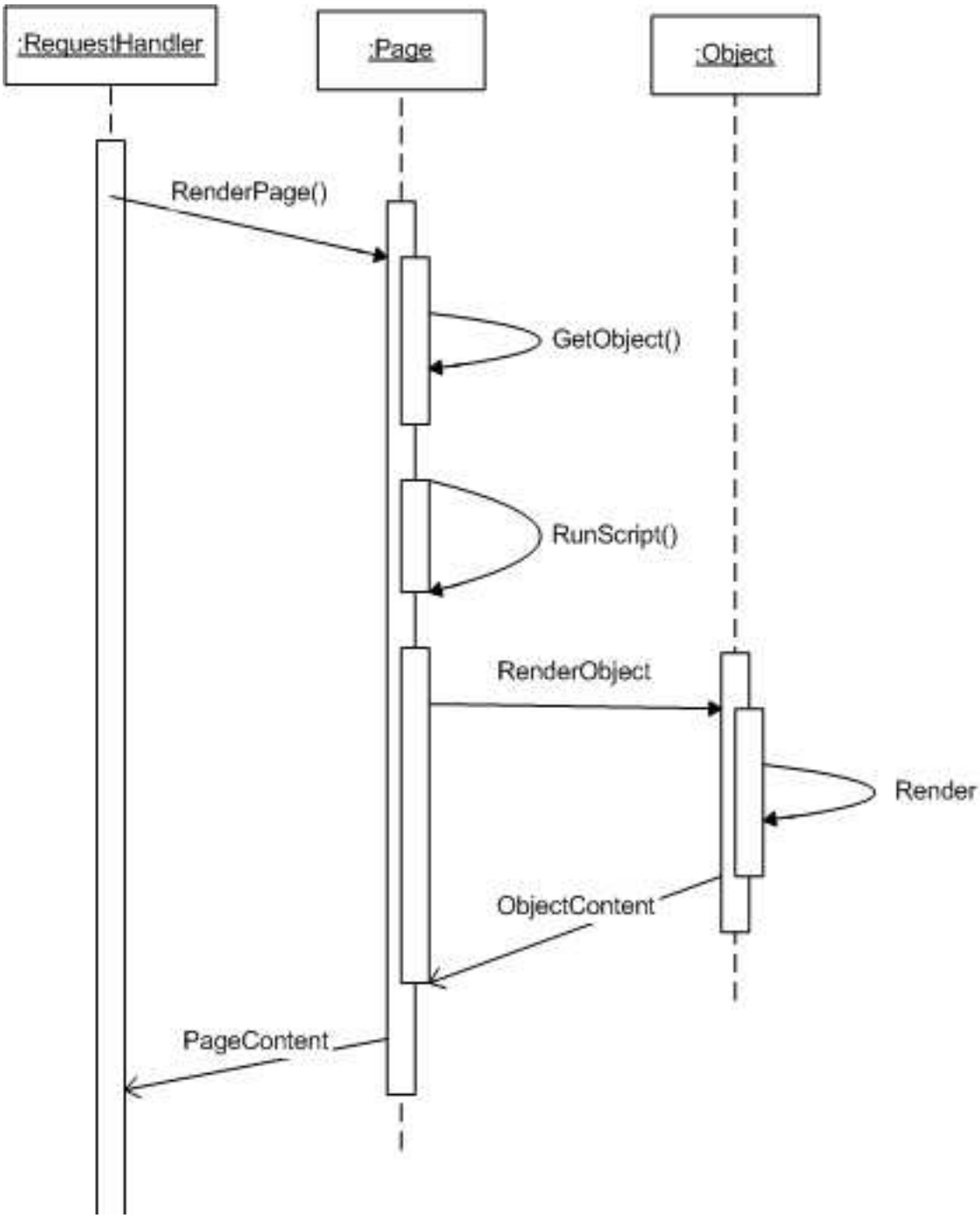

Рисунок 3 — Генерация страницы по запросу.

Общая структура страницы на псевдоязыке выглядит следующим образом:

```
<!- Object declaration ->
<%
object1 = Query(id1)
object2 = Query(id2)
object3 = Query(id3)
%>
<!- Script section ->
<%
object1.color("red")
object2.width("100%")
%>
<!- Presentation section ->
<html>
<head>
<title>Sample page</title>
</head>
<body>
<!- Render section ->
<%
object1.run()
object2.run()
%>
</body>
</html>
```

Из примера видно, что в документе кроме областей статической разметки имеются области выполнения встроенного кода, которые позволяют задать "поведение" страницы. В начале страницы размещается секция декларации (*Object declaration*) объектов. В этой секции объекты запрашиваются из базы данных. После этого становятся доступными операции над свойствами объектов. Операции над объектами должны быть произведены до отображения объектов, поэтому область сценария страницы (*Script section*) размещается во второй секции. Последняя секция (*Render section*) содержит процедуры отображения объектов.

3.2.4 Сценарий

Динамические документы создаются при помощи специализированных систем программирования (например, ASP). Наиболее важная задача любой технологии создания динамических страниц — обработка действий пользователя. Возможность реализации обработчиков формирует предпосылки для создания web-приложений. Система V.D.O.M. позво-

ляет встраивать специализированные сценарии в страницы, что позволяет создавать динамические документы произвольной сложности. Сценарии V.D.O.M. реализуются на языке программирования Visual Basic и оформляются как фрагменты ASP.

Один из существующих недостатков V.D.O.M. — искусственное ограничение функциональности. Данное ограничение введено для того, чтобы обеспечить безопасное функционирование ядра V.D.O.M. в агрессивной многопользовательской среде. При отсутствии таких ограничений могла бы возникнуть опасность нарушения целостности системы при работе с небезопасными объектами Visual Basic.

3.2.5 Событие

При использовании подсистемы событий изменяется структура статических секций страницы и код генерации компонент. Ключевая возможность подсистемы событий — управление объектом без перезагрузки документа в браузера пользователя. Благодаря этому возможно создавать клиент-серверные приложения аналогичные приложениям, выполняющимся в однородной клиентской среде, без использования java-апплетов и flash-компонентов. Вся функциональность реализуется посредством DHTML. Потенциально, это позволяет обеспечить большую совместимость между клиентскими платформами. При использовании подсистемы событий схема взаимодействия клиента и сервера выглядит следующим образом (рис. 4):

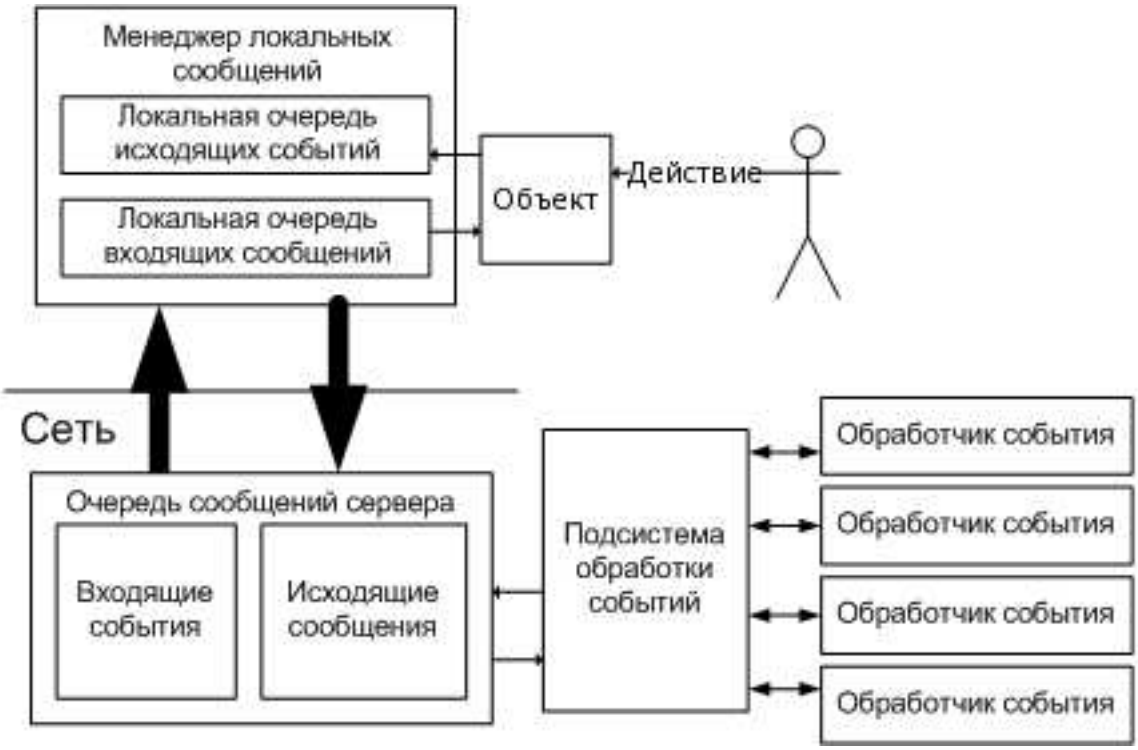

Рисунок 4 — подсистема управления событиями

Событийно-ориентированные приложения (далее СО-приложения) ориентированы на работу с DOM [10], что позволяет более эффективно программировать интерфейс приложения. В СО-приложении объекты представляются JavaScript-кодом, который позволяет генерировать и обрабатывать события, отображать объект и управлять его поведением. Для реализации схемы доставки, приема событий каждый документ состоит из трех частей (рис. 5):

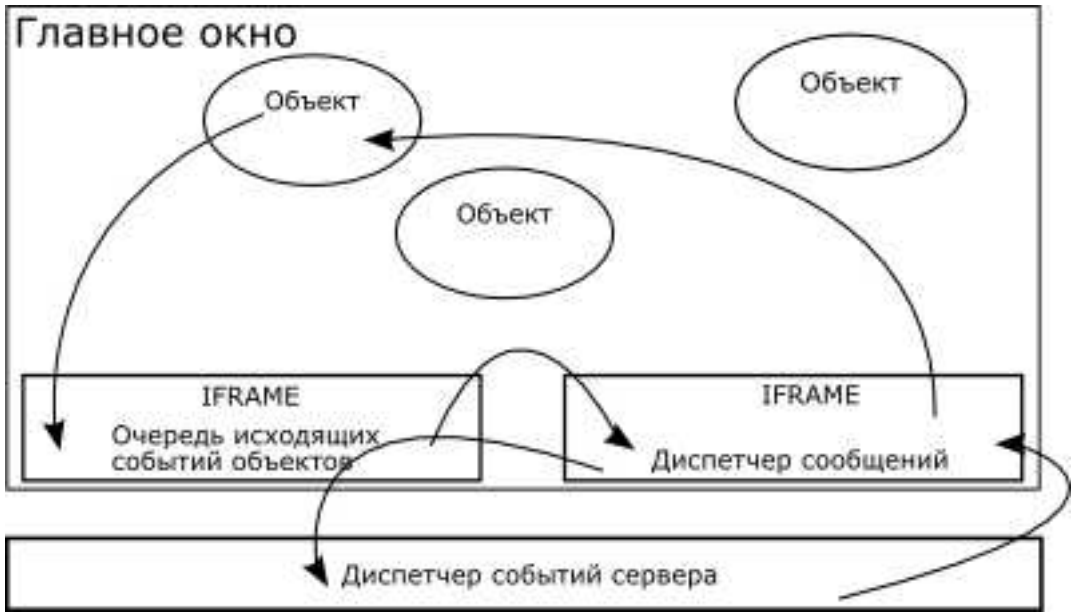

Рисунок 5 — документ в СО-приложении

СО-приложение оперирует не документами, а событиями. В документе находтся объекты, которые, реагируя на действия пользователя, генерируют события. События помещаются в очередь исходящих событий объектов. Из этой очереди события передаются диспетчеру сообщений, который отправляет события на сервер или обрабатывает локально, если имеется такая возможность. События передаются на сервер в виде сообщений, обрабатываются и возвращаются в диспетчер сообщений в виде кода на языке JavaScript. Далее, этот код выполняется родительским окном, что влечет за собой изменение документа полной перегрузки страницы.

3.3 Резюме

В этом разделе дано упрощенное описание технологии V.D.O.M. Далее будут сформулированы задачи дипломной работы и описаны технические решения.

4 Подготовительный этап разработки

Поскольку для предоставления полной функциональности исходной системы необходимо полностью повторить все решения в целевой разработке, первой фазой являлось ознакомление с функциональностью V.D.O.M. с точки зрения пользователя. На данном этапе представитель Cybertronique inc. Nicolas Kourbulevsky — автор технологии V.D.O.M. представлял функционирование системы при помощи схем и презентации. По окончании недельного ознакомительного семинара была выяснена граница проектируемой системы, что позволило более точно описать модель переносимой среды выполнения. В ходе этого этапа было определено назначение всех наиболее важных подсистем и интерфейсов между ними.

По договоренности с партнерами развитие системы должно происходить в несколько этапов. Каждый этап должен завершаться выпуском функционирующей системы с ограниченной функциональностью. На следующем этапе к текущей системе должна добавляться дополняющая функциональность и выпускаться следующая версия.

Ключевым моментом разработки является полное тестирование подсистем. Для каждой подсистемы пишется набор тестов, которые определяют способы использования данной подсистемы и ее интерфейсы к другим компонентам приложения. Набор тестов составляется для каждого модуля[3].

Система тестирования была специально разработана для выполнения тестов модуля. Каждый модуль содержит класс TEST с определенным интерфейсом. Класс TEST включает в себя набор тестов для модуля, в котором определен данный класс. На практике это выглядит следующим образом:

example/module.py

```
class C1:
    # some code here
class C2:
    # some code here
#------------------
# Test environment
class TEST:
    #Test environment for example.module
    def init(self):
```

[3]Модуль — минимальная распространяемая единица языка Python [3]

```
    Nothing here
    pass
#-----------------
# receive all tests
def tests(self):
    return [self.test1,self.test2,self.test3,]
#-----------------
# test1
def test1(self):
    # test code here
def test2(self):
    # test code here
```

Для централизованного выполнения всех тестов применяется специализированное приложение, результатом выполнения которого является страница тестирования, в которой отображаются результаты выполнения всех тестов модулей, модули, в которых отсутствуют тесты, успехи и неудачи тестирования.

Централизованное тестирование призвано упростить поиск ошибок при интеграции новых подсистем в существующую версию. Отдельно выделяется фаза интеграции, во время которой производится централизованное тестирование с целью выяснения нарушений функциональности.

4.1 Соглашение по программным средствам

Основное требование к системе — переносимость между широким спектром аппаратно-программных платформ. Удачный выбор инструментария позволяет расширить список поддерживаемых платформ и уменьшить время разработки приложения.

В качестве средства разработки был выбран язык Python. В пользу данного решения говорят следующие факторы: широкий список поддерживаемых платформ, ортогональный синтаксис и развитые возможности, встроенная система документирования. Приложение разрабатывается на Python 2.3.

Поскольку в проекте задействовано несколько разработчиков, то для хранения исходных кодов приложения и сопровождающих файлов используется централизованный репозиторий. Репозиторий построен на основе системы контроля версий CVS. Для мониторинга используется система CVS Stat, отображающая информацию о состоянии проекта в Web.

Для тестирования приложения используется пять сред: Microsoft Windows XP SP1, Microsoft Windows 2003 Enterprise Edition, GNU/Linux SuSE 9.0, GNU/Linux ALT Linux 2.4 Master, FreeBSD 4.10-STABLE. Данные платформы позволяют осуществлять тестирование приложения в различных условиях.

Для тестирования производительности приложения используется утилита Apache Bench из дистрибуции Web сервера Apache 2.x.

5 Архитектура приложения

5.1 Сервер приложений

Основа системы — http-сервер, выполняющий приложение V.D.O.M. CPE. Сервер должен обладать высокой производительностью, чтобы обеспечить функционирование асинхронной подсистемы событий, но в то же время он должен быть переносимым между различными программно-аппаратными платформами.

Сервер реализован при помощи стандартных классов языка Python ThreadingTCPServer и SimpleHTTPRequestHandler. Использование этих стандартных классов позволяет значительно упростить процесс разработки. Сервер полностью эмулирует среду выполнения CGI-приложения.

Цель эмуляции — обеспечить функционирование ядра V.D.O.M. CPE как в среде CGI, так и в качестве модуля HTTP сервера. Сервер предоставляет приложению V.D.O.M. CPE среду выполнения. Назначение среды выполнения — обеспечения интерфейса между http-сервером и приложением V.D.O.M. CPE. Так же сервер предоставляет интерфейс управления заголовками HTTP протокола.

Архитектура сервера представлена на диаграммах пакетов (рис. 6) и классов (рис. 7).

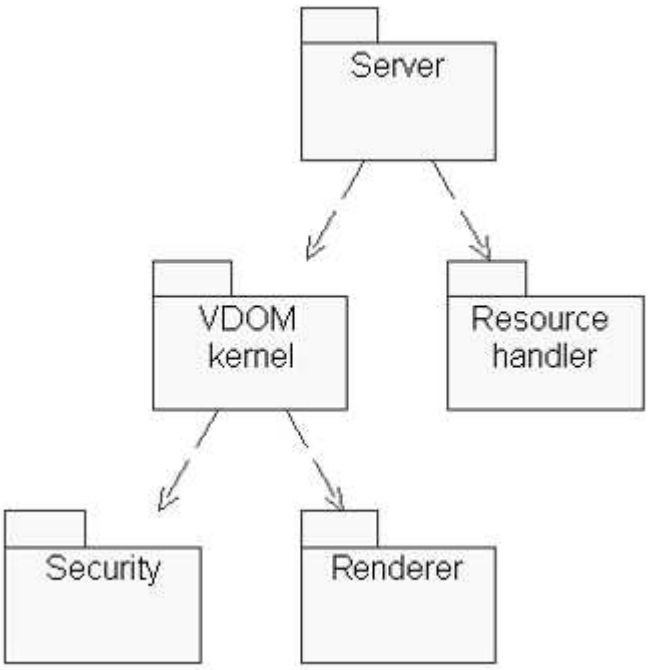

Рисунок 6 — Архитектура подсистем сервера.

Сервер построен при помощи стандартных классов языка Python, которые реализуют многопоточный http-сервер и обработчик запросов. Наследованием от данных классов получены специализированные сущности, предназначенные для построения многопоточного http-сервера. Сервер может запускать множество обработчиков запросов пользователя (по одному на запрос). Обработчик запроса создает сущность $VDOM\_kernel$, которая предназначена для предоставления пользовательского API управления сервером. Объект сессии (см. далее) разделяется между всеми обработчиками запросов сервера.

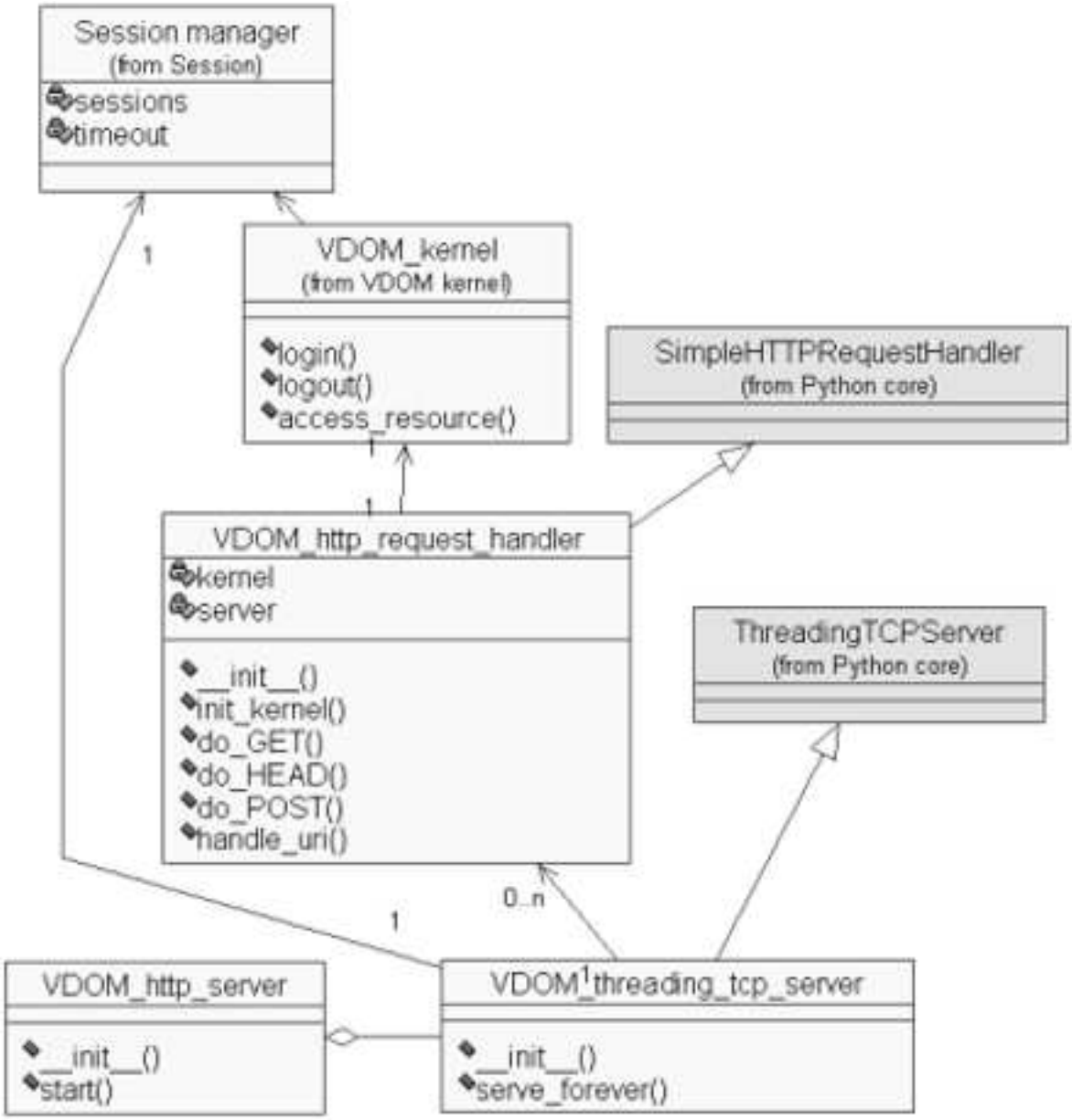

Рисунок 7 — Архитектура сервера.

5.2 Сессии и привилегии

Особое значение в играет подсистема привилегий и отслеживания сессий. Подсистема привилегий организована по принципу разрешающего приоритета (для доступа к определенному ресурсу запрашиваемый приоритет ресурса должен быть выше текущего приоритета пользовате-

ля). В этой подсистеме присутствует два типа объектов: пользователь системы и ресурс системы (рис. 8).

Пользователь системы — владелец текущей сессии. Каждому пользователю администратор выдает приоритет, который определяет множество ресурсов, к которым пользователь имеет доступ. Ресурс системы — URI, запрашиваемый пользователем системы. Для URI выставляется минимальный уровень доступа.

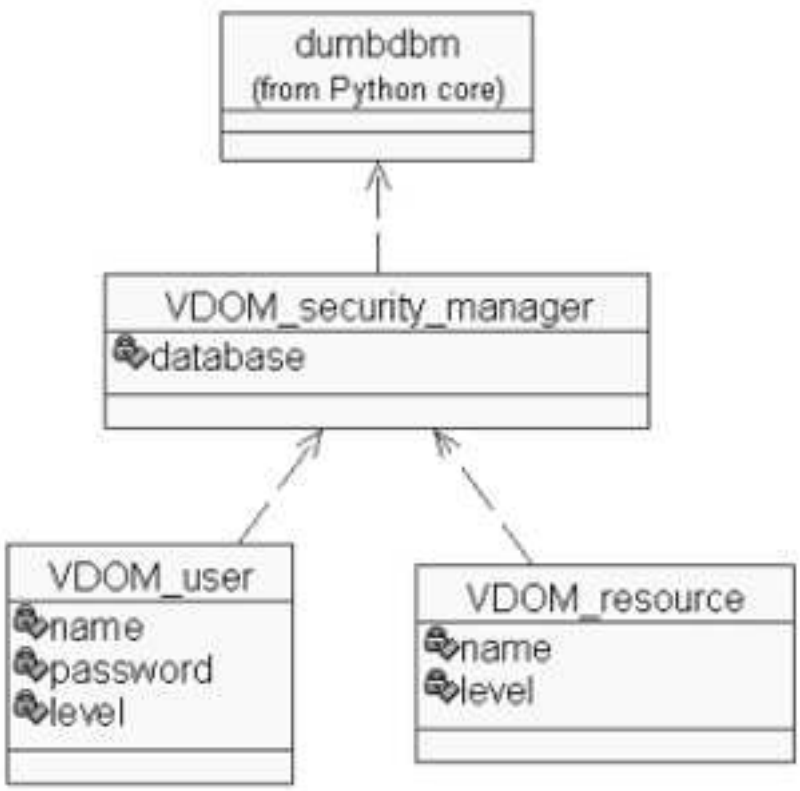

Рисунок 8 — Архитектура подсистемы безопасности.

При обращении пользователя к ресурсу происходит поиск ресурса в БД ресурсов. Далее для ресурса проверяется доступ. Алгоритм проверки доступа выглядит следующим образом:

```
resource_uri = None
error_resource = new VDOM_resource("/error.py")
if resource.acl <= user.acl:
   # access to the page is granted
   resource_uri = resource.uri
else:
```

```
   # access denied
   resource_uri = error_resource.uri
# Operate with resource
```

Информация о пользователе хранится в сессии[6]. Поэтому особое значение играет подсистема сессий, в которой сохраняются параметры пользовательских соединений. Сессия представляет собой специальную область памяти, в которой хранятся переменные окружения пользователя. На стороне клиента сессия представляется специализированной записью cookie, в которой хранится идентификатор сессии. Необходимость сессии возникает в силу того, что протокол HTTP не ориентирован на постоянное соединения клиента и сервера, что не позволяет идентифицировать принадлежность соединения к определенному клиенту.

5.3 Кэш объектов

В системе V.D.O.M. все объекты хранились в централизованной БД, которой управляла СУБД общего назначения MS SQL. В переносимой системе нет возможности использовать выделенную СУБД для хранения объектов, поэтому объекты хранятся в файловой системе. Очевидно, что нецелесообразно постоянно обращаться к файловой системе, когда требуется получить определенный объект; в то же время множество объектов хранимых в оперативной памяти могут занять все ресурсы вычислителя, что вызовет уменьшение производительности других приложений. В качестве компромисса между этими подходами был выбран такой алгоритм, который позволяет хранить в памяти лишь ограниченное количество актуальных объектов, вытесняя их на диск, когда требуется загрузить отсутствующий в памяти объект.

Решением является подсистема кэширования с вытеснением. Принцип вытеснения — LRU. Использование объекта характеризуется количеством запросов к кэшу с целью получения данного объекта. Для отслеживания запросов к объекту существует специальная очередь, которая определяет объекты для вытеснения. Схематически система кэширования организована в виде таблицы загруженных объектов (рис. 9).

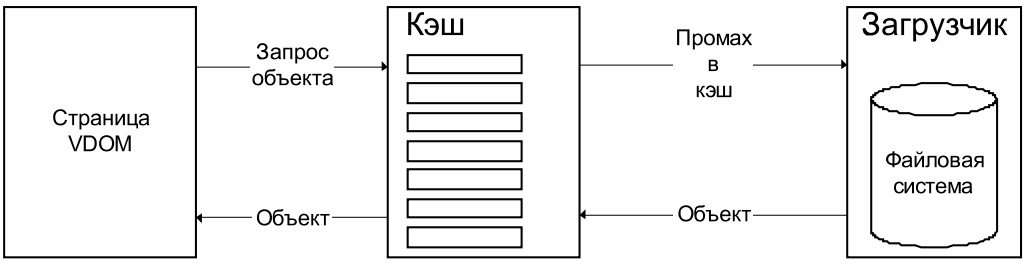

Рисунок 9 — Кэш объектов

Основа подсистемы кэширования (рис. 10) — класс-сериализатор (Pickle) языка Python. Данный класс используется классом $VDOM\_loader$ для загрузки и сохранения объектов. Приоритеты объектов вычисляются по нахождению объекта в очереди $VDOM\_queue$

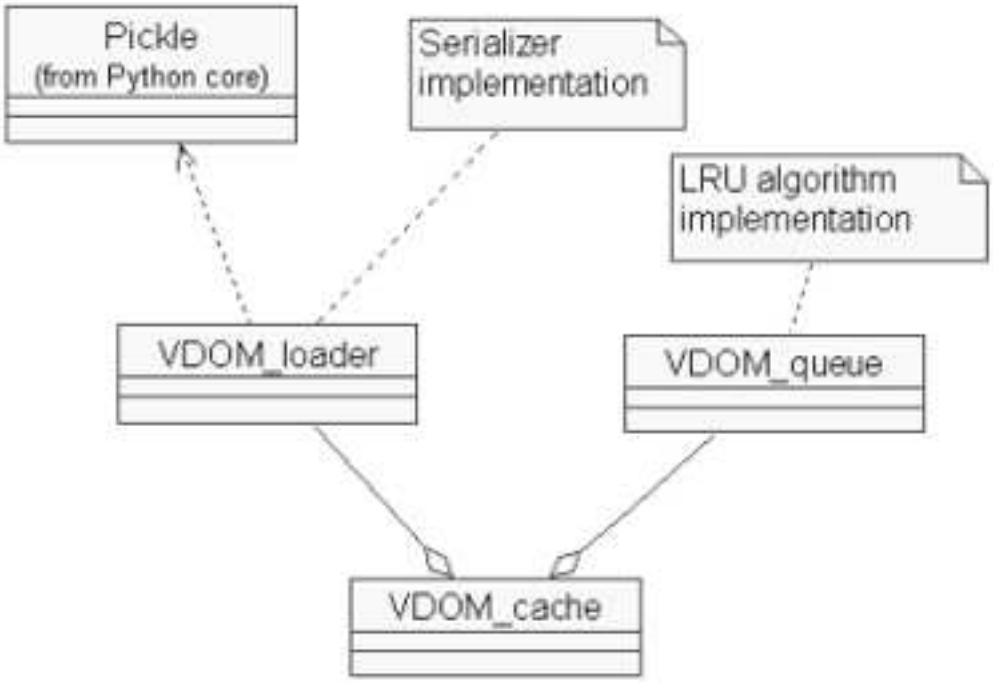

Рисунок 10 — Архитектура подсистемы кэширования

5.4 Архитектура подсистемы страниц

Каждая страница в системе V.D.O.M. CPE состоит из трех частей: сценарий страницы, объекты на странице, служебная информация. Страница является одной из основных сущностей V.D.O.M. Страница преобразуется при помощи класса $VDOM\_renderer$ в выполняемый код

Python (рис. 11). Кэширование страниц организовано аналогично кэшированию объектов и реализовано на тех же классах.

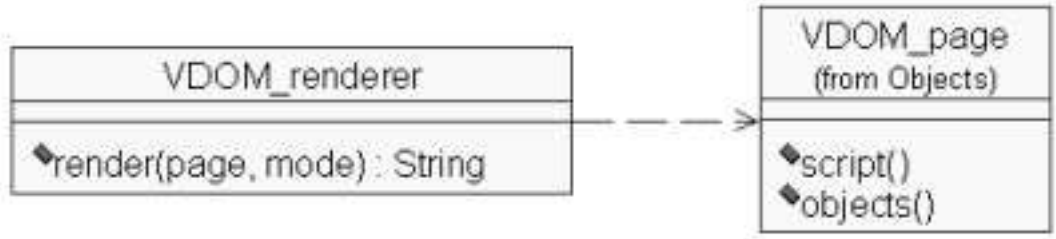

Рисунок 11 — Генератор python-кода

5.5 Ресурсы системы

Другой важной подсистемой является подсистема обработки пользовательского запроса. Основная функция данной подсистемы — определение типа запрашиваемого ресурса и корректное предоставление ресурса пользователю. Ресурс — файл хранящийся на сервере. Сервер распознает ресурс по расширению запрашиваемого файла и выдает поток байт в соответствии с типом потока для данного расширения файла. Поскольку некоторые типы ресурсов требуют дополнительной обработки то в сервере предусмотрено предоставление обработчика для каждого типа файла. Это позволяет обеспечить единообразный интерфейс предоставления ресурсов. Таблица обработчиков выглядит следующим образом(Таблица 1):

Расширение	Модуль	Класс-обработчик
.jpe?g	VDOM.kernel.resource.handler	VDOM_jpeg_handler
.png	VDOM.kernel.resource.handler	VDOM_png_handler
...	...	...

Таблица 1 — обработчики

Ключи в таблице — регулярные выражения, определяющие тип ресурса. Значения таблицы — модуль, где хранится обработчик и класс обработчика. Все обработчики наследуются от интерфейса VDOM_handler (рис. 12).

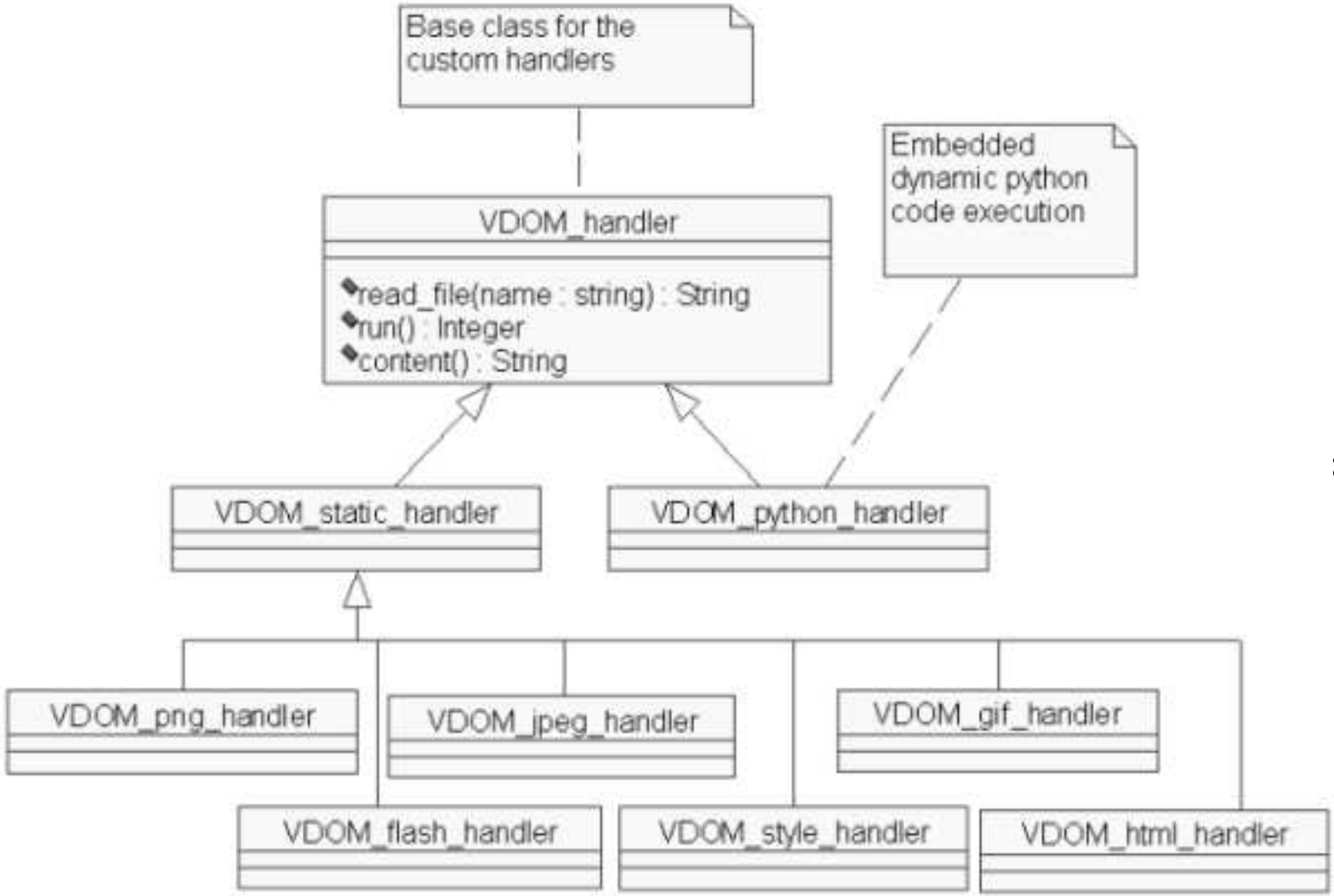
Base class for the
custom handlers
Embedded
dynamic python
code execution
VDOM_handler
read_file(name : string) : String
run() : Integer
content() : String
VDOM_static_handler
VDOM_python_handler
VDOM_png_handler
VDOM_jpeg_handler
VDOM_gif_handler
VDOM_flash_handler
VDOM_style_handler
VDOM_html_handler

Рисунок 12 — Архитектура обработчиков типов ресурсов системы

5.6 Средства синхронизации

Поскольку объекты разделяются несколькими потоками, то возникает необходимость синхронизации доступа к ресурсам. Основная проблема, относящаяся к проблеме синхронизации — локальность средств синхронизации. Например, семафор — объект класса синхронизации. У семафора отсутствует имя. Для того, чтобы семафор был виден двум функциям необходимо объявить его в глобальной для этих двух функций области имен. Это не совсем удобно, так как в нашем случае имеется множество объектов, с каждым из которых должен быть ассоциирован бинарный семафор. Было решено реализовать специализированную прослойку, цель которой — именование всех объектов синхронизации. Все объекты синхронизации хранятся в глобальной таблице синхронизации и могут запрашиваться и освобождаться по имени. Аналогичным способом были определены критические секции, которые позволяют осуществлять монопольное использование участков кода несколькими потоками.

5.7 Встраиваемая СУБД

С точки зрения прикладного значения важным компонентом приложения является подсистема встроенных таблиц данных клиента. Клиент может определить отношения и представлять данные из этих отношений в документах. Схема представления выглядит следующим образом(Рисунок 13):

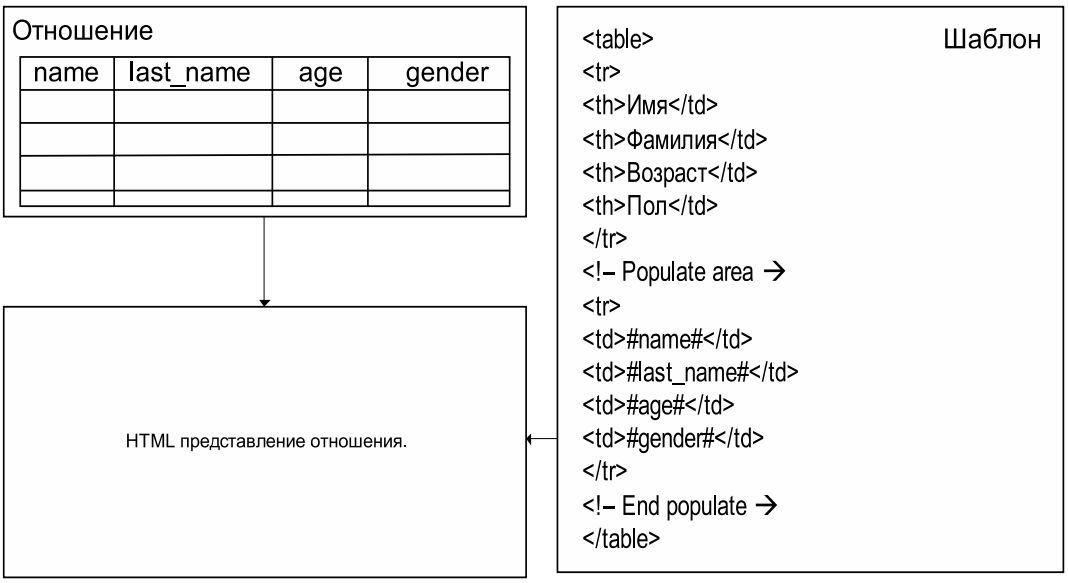

Рисунок 13 — Отношение и представление по шаблону

Существует основанный на SQL механизм, позволяющий осуществлять более гибкие операции с БД. Необходимо заметить, что отношения могут объединяться в цепочки связей 1:1. Это позволяет оптимизировать доступ только к необходимым данным, не применяя запросов, а только при помощи шаблонов.

Реализовать встраиваемую СУБД, позволяющую оперировать эффективно с большими объемами данных, не представляется возможным. Кроме того, требуется обеспечить высокую эффективность операций выборки, обновления и добавления новых строк в отношения. Данная проблема может быть эффективно решена, если исходить из условия локальности данных. Это условие означает, что в текущий момент операции происходят с данными, которые располагаются рядом, в соответствии с системной индексацией. Предположение о локальности данных можно сделать на основании механизма отображения данных в документе. Отношение отображается страничным способом по N строк на странице. Знание этого факта позволяет нам организовать отношения в виде страничных файлов. Каждый файл должен быть достаточно маленьким, чтобы можно было запрашивать все локальные данные без загрузки лишних строк таблицы и достаточно большим для того, чтобы место в файловой системе не тратилось напрасно. Страничная структура позволяет легко проводить операции обновления, удаления и вставки новых элементов в отношения. При этих операциях целиком перезаписывается тот страничный файл, на котором находится данная строка. Для умень-

шения количества операций с файловой системой страницы отношения хранятся в кэше с алгоритмом LRU, перезапись происходит при вытеснении и выполнении условия “dirty page”. Поскольку для отношения не определяется тип полей, то его строки могут иметь произвольную длину. Отношение представляется набором строк, метаинформации и имеет имя (рис. 14). Все отношения доступны через специализированный менеджер баз данных, который обеспечивает синхронизацию с файловой системой и спецификационные операции.

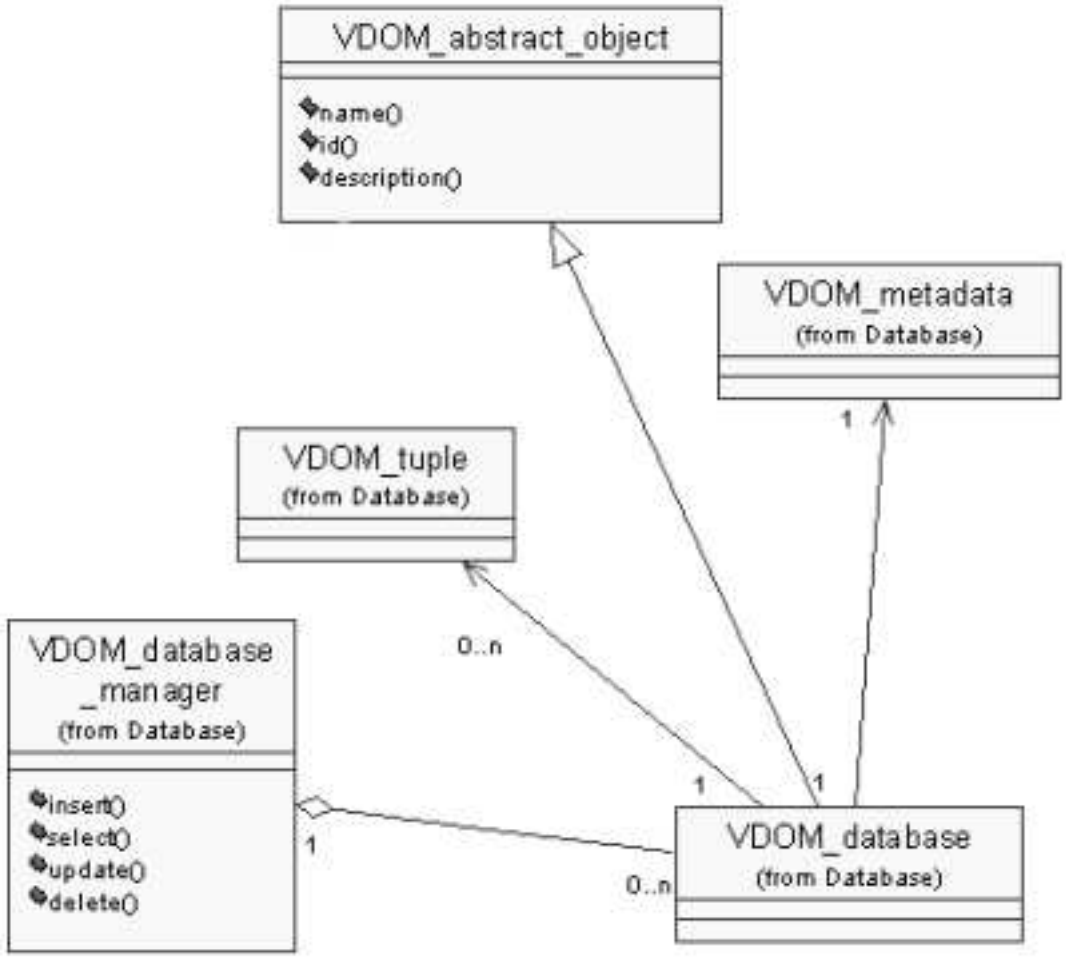

Рисунок 14 — Структура отношения

5.8 Исполнение динамических сценариев

Для выполнения динамических сценариев в сервер был добавлен специализированный тип ресурсов с расширением ".py". Этот ресурс

представляет собой HTML-код с вставками Python-кода. Выглядит это следующим образом:

```
<html>
<head>
<title>VDOM HTTP server management area</title>
</head>
<body bgcolor="blue»
<%
import thread
request = kernel.request().request()
cookies = kernel.cookies().cookies()
env = kernel.environment().environment()
if kernel.access_resource("/manage/stop.py") and
request.has_key("command") and request["command"][0] == "stop":
     kernel.server().access_logger().info(
     "VDOM_http_server", thread.get_ident(),
     "Termination command recieved from %s, sid = %s",
env["REMOTE_ADDR"], cookies["sid"])
     kernel.server().stop()
     print '<center>'
     print '<h3>Server stopped</h3>'
     print '</center>'
else:
     code = 401
%>

</body>
</html>
```

Предоставление данного ресурса происходит в два этапа: на первом этапе происходит трансляция всего кода в сценарий языка Python, а на втором этапе происходит исполнение кода и генерация HTTP ответа. Поскольку время трансляции негативно влияет на скорость предоставления ресурса, то можно включить промежуточную стадию кэширования кода. В этом случае первая стадия происходит только при изменении исходного ресурса.

6 Утилита преобразования БД V.D.O.M. в V.D.O.M. CPE

Данные V.D.O.M. не могут использоваться в V.D.O.M. CPE. Данное ограничение следует из того факта, что в V.D.O.M. объектами управляет СУБД MS SQL Server, а в V.D.O.M. CPE объекты хрянятся в файловой системе. Преобразование данных из МД V.D.O.M. в V.D.O.M CPE может быть осуществлено двумя способами:

1. Просмотр БД MS SQL Server и построение МД V.D.O.M CPE;
2. Построение МД V.D.O.M. CPE по сохраненной резервной копии приложения V.D.O.M.

Для реализации утилиты преобразования был выбран второй подход в силу ряда преимуществ:

1. возможность локального преобразования;
2. независимость от целевой платформы;
3. простота реализации;
4. независимость от среды выполнения V.D.O.M.

Резервная копия приложения хранится в виде набора XML файлов, в которых описывается структура приложения, страницы,объекты и события. При помощи анализа XML файлов генерируются структуры данных, необходимые для функционирования V.D.O.M. CPE.

Анализ XML файлов производится при помощи встроенных классов Python. Для генерирования сущностей МД V.D.O.M CPE используется подсистема кэширования.

Утилита состоит из двух подсистем: анализатора и генератора. Анализатор разбирает XML файлы и строит дерево разбора в памяти. Генератор обходит дерево разбора и при помощи подсистемы кэширования создает целевые объекты в файловой системе.

Для работы некоторых объектов необходимы статические ресурсы. Статические ресурсы так же хранятся в резервной копии и копируются в целевую МД при создании объектов.

7 Компонентно-ориентированная модель Web-приложений

В данной главе предлагается новый подход к разработке приложений web. Подход является развитием традиционного способа разработки[1] и предназначен для повышения качества программных решений, уменьшения стоимости и времени разработки за счет повторно используемых стандартных компонентов. Новизна подхода заключается в том, что компоненты реализуются без использования сторонних расширений браузера и обладают максимально широкой переносимостью.

7.1 Введение

Современные web-приложения[1] представляют собой комплексные системы с развитым интерфейсом пользователя и комплексной логической частью.

Специфика web-приложений заключается в том, что логическая часть системы сосредоточена на сервере, а результаты предоставляются клиенту на удаленный узел посредством специализированной программы, называемой браузером. Основная единица интерфейса пользователя — документ. Каждое действие пользователя заставляет сервер формировать документ, в котором содержится ответ на действие.

Документоориентированность представляет собой существенное ограничение web-интерфейса (далее WebGUI) и вытекает из особенностей протокола HTTP(S)[12]. Если рассмотреть графический интерфейс пользователя, предоставляемый операционной системой (далее GUI), то основной единицей интерфейса является окно, что позволяет отражать изменения непосредственно. Каждое окно может функционировать независимо от остальных элементов интерфейса и имеет свои собственные обработчики команд. Данная особенность интерфейса позволяет легко реализовывать компоненты на базе окон (примеры таких компонент можно найти в VCL Delphi). Поскольку из накопленного опыта известно, что компоненты упрощают разработку приложений, то, очевидно, что создание компонентов для WebGUI, может оказать существенное влияние на разработку web-приложений.

В данной работе предлагается создавать приложение из структурных объектов, обладающих функциональностью и представлением, с определенным поведением и состоянием, выполняющих определенный набор функций, характерных для типа, к которому относится объект. Для краткой ссылки на такие структурные объекты далее будет использован термин "компонент". Каждый компонент относится к определенному "типу компонента", который характеризует функциональность и задачи компонента. Основное отличие компонента от других логических объектов приложения — самодостаточность, что означает возможность применения компонента без использования дополнительного кода.

В качестве примера компонента можно рассмотреть табличный отчет по SQL запросу, другой пример — компонент, позволяющий осуществлять загрузку файлов на сервер с ограничениями на загружаемые файлы.

При проектировании компонента должно учитываться его влияние на другие компоненты, размещенные в том же документе. Очевидно, что компонент должен оказывать влияние на другие компоненты только в

том случае, если другие компоненты "хотят"подвергаться его влиянию. Например, если в одном и том же документе располагаются каталог товаров и корзина покупателя, то каталог должен влиять на корзину при выборе некоторого продукта из каталога для покупки. Корзина же, в свою очередь, также должна влиять на каталог, в случае, когда из корзины удаляется продукт. Если в этом же документе располагается компонент "отчет о погоде за текущую неделю", то манипуляции с корзиной и каталогом не должны оказывать какое-то влияние на этот компонент.

7.2 Обзор традиционного подхода

Под традиционным подходом понимается метод разработки приложений web, который характеризуется наличием отдельных обработчиков команд пользователя, каждый из которых порождает документ. Так как приложения web ориентированы на выполнение в режиме "вопрос-ответ", то приложение представляется набором реакций на действия пользователя. Реакция на определенное действие называется режимом выполнения (run mode). Наличие режимов выполнения характерно для всех систем с высокой степенью интерактивности.

В любом запросе и ответе клиенту содержится информация, которая состоит из двух частей:

1. управляющая информация;
2. информация, необходимая бизнес-процессу приложения в качестве входных данных;

Будем считать, что с каждым приложением связана Глобальная База Данных (ГБД). Состоянием приложения назовем состояние ГБД. Состояние ГБД — управляющая информация приложения, полученная из предшествующих запросов и ответов. Каждый режим выполнения формирует некоторое состояние ГБД. Каждый режим выполнения — функция, которая преобразует данные пользователя в некоторый результат:

$$Y = R(X)$$

Где,
X — Данные пользователя (запрос).
R — Режим выполнения.
Y — Ответ сервера на запрос пользователя.

С другой стороны, приложение web можно рассматривать как конечный автомат, состояниями которого являются состояния ГБД, а переход между состояниями осуществляется посредством управляющей информации. Очевидно, что без потери общности можно считать эквивалентными понятиями состояние ГБД и режим выполнения. В дальнейшем эти термины будут употребляться как взаимозаменяемые. Управляющая информация является неотъемлемым атрибутом каждого запроса, что предусмотрено в рамках протокола HTTP(S). В управляющем информационном блоке передается режим выполнения запроса.

Единственное несоответствие приложения конечному автомату может проявляться в силу отсутствия н ачального и конечного состояний (конечный автомат должен иметь начальные и конечные состояния).

С точки зрения автоматного языка приложение выглядит следующим образом:

$S = \{S1, S2, \ldots, Sn\}$ — состояния (режимы выполнения) приложения;

W — Множество управляющих команд приложения (язык).

Правила перехода между состояниями описывают работу пользователя с приложением:

$(Command)(StateI) -> (StateJ)$

Рассмотрим пример приложения, предназначенного для просмотра сообщений как конечного автомата (рис. 15).

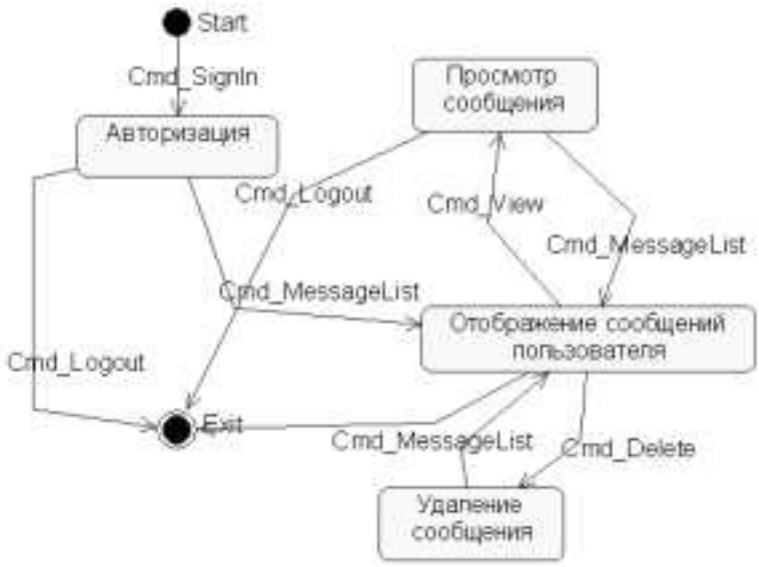

Рисунок 15 — Граф приложения

Таким образом, в традиционном подходе к разработке приложение представляется в виде графа режимов выполнения, переход между которыми осуществляется при помощи управляющих команд. Вышеописанный подход к разработке приложений web будет расширен для построения модели приложений, основанной на компонентах.

7.3 Построение компонентно-ориентированной модели

Определим компонент как конечный автомат с множеством состояний, которое определяет функциональное назначение компонента. Состояние компонента вводится аналогично состоянию приложения — состояние ГБД компонента (ГБДК). В КО-модели режимы выполнения определяются не только для приложения, но и для каждого компонента в отдельности, иными словами, не только приложение является конечным автоматом, но и каждый из компонентов — конечный автомат относительно собственных режимов выполнения (состояний ГБДК). Компоненты проектируются и реализуются таким образом, чтобы не влиять при своем выполнении на другие компоненты, расположенные в том же документе.

Документ используется как контейнер для компонентов. В то же время в документе может присутствовать бизнес-логика, не относящаяся к какому либо компоненту. В данной работе мы будем считать, что режимы выполнения приложения реализуются как динамические документы, которые содержат или не содержат компоненты.

Компонент состоит из алфавита, множества состояний и грамматики. Алфавит — множество управляющих команд. Состояние — режим выполнения компонента. Грамматика — набор продукций, определяющий правила работы компонента.

Бизнес-логика (как компонентная, так и некомпонентная) внедряется в документ при помощи встроенного языка (например, PHP или ASP) либо при помощи других специализированных средств языков программирования (например, шаблоны разметки или XML/XSLT). В качестве примера компонента можно рассмотреть компонент табличного отчета по SQL запросу (рис. 16).

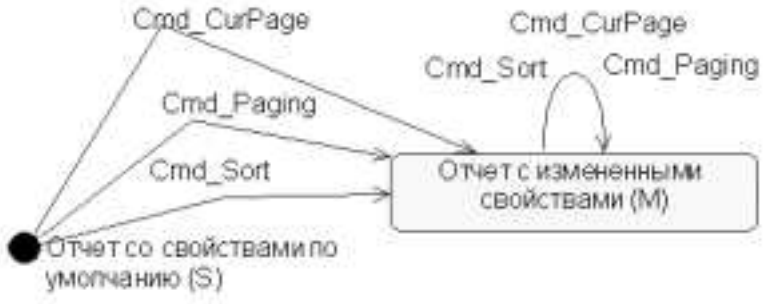

Рисунок 16 — Граф компонента табличного отчета

Компонент табличного отчета предназначен для отображения та-

бличных данных в страничной форме с возможностью изменения количества строк на странице и режима сортировки данных. Пример компонента табличного отчета (рис. 17):

Рисунок 17 — Компонент табличного отчета

В компоненте допустимы следующие действия:

1. изменение порядка сортировки строк в отчете;
2. изменение количества строк на странице;
3. изменение текущей страницы отчета.

Алфавит компонента состоит из следующих команд:
$\{Cmd\_Paging, Cmd\_Sort, Cmd\_CurPage\}$.

Состояния компонента:
S — отображение компонента со свойствами, заданными по умолчанию;
M — отображение отчета с измененной ГБДК.

Правила языка:
$Cmd\_PagingS \mid Cmd\_PagingM$ -> M (по команде Cmd_Paging изменить количество отображаемых строк на таблице отчета)
$Cmd\_SortS \mid Cmd\_SortM$ -> M (по команде Cmd_Sort изменить количество отображаемых строк на таблице отчета)
$Cmd\_CurPageS \mid Cmd\_CurPageM$ -> M (по команде Cmd_CurPage изменить текущую отображаемую страницу)

7.4 Реализация компонентов

Основные проблемы реализации следующие: каким образом сохранять состояние компонента, когда он неактивен (пользователь работает с другим компонентом), каким способом идентифицировать компоненты и посылать сообщения другим компонентам в документе.

Для хранения состояния компонента во время его пассивности необходимо использовать механизм, называемый сессией, который позволяет сохранять значение структур данных между транзакциями (рис. 18). Альтернативно, состояние компонента можно хранить в отношении реляционной СУБД.

Поскольку в одном документе может быть расположено произвольное количество компонентов, то для возможности уникальной идентификации каждый компонент должен иметь уникальное имя. Имя компонента задается пользователем и позволяет идентифицировать отправителей и получателей сообщений.

Рисунок 18 — Фрагмент сессии

В силу специфики посылки запроса серверу по протоколу HTTP(S), сообщения каждого компонента должны содержать префикс (имя компонента), указывающий, что сообщение сгенерировано данным компонентом. Схема именования может быть произвольной, например:

<ComponentName>.<MessageParameterName> = <MessageParameterValue>

В качестве примера, рассмотрим простое сообщения компонента "Табличный отчет"с именем "Report1", которое меняет текущую отображаемую страницу.

Report1.Cmd_CurPage = 2

Легко заметить, что компоненты могут быть разделены на два больших класса: компоненты, которые полностью зависят от действий пользователя (назовем их чистыми) и которые зависят от внешних факторов, не связанных с деятельностью пользователя (назовем их смешанными).

Поскольку чистые компоненты при отсутствии операций не меняются, то они могут быть оптимизированы при помощи сохранения последнего представления в сессии и последующей загрузки.

Поскольку в одном документе может быть несколько компонентов, то требуется инфраструктура, которая позволит корректно обеспечивать работу как связанных, так и независимых компонентов. Инфраструктура обеспечивается диспетчером выполнения (рис. 19) и трансляторами сообщений.

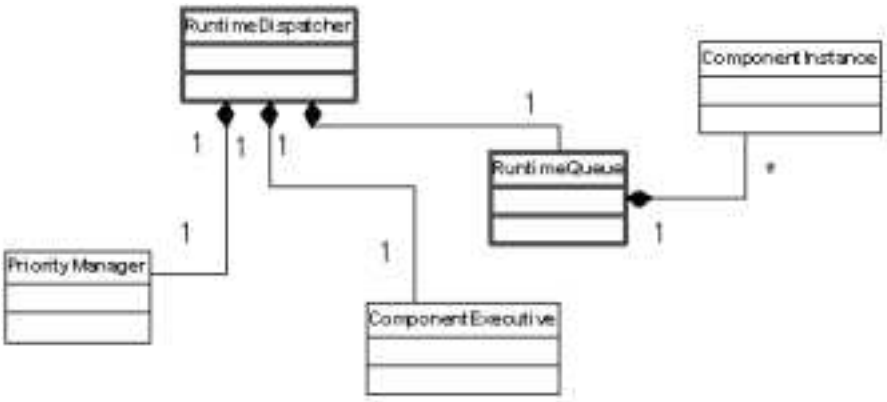

Рисунок 19 — Диспетчер выполнения

В предлагаемом подходе считается, что компонент выполняется изолированно. Изолированность означает, что компонент не имеет информации о других компонентах, размещенных в документе. В связи с этим для обмена сообщениями между компонентами используется специализированный компонент — "транслятор", задача которого передавать сообщения между компонентами.

Например, такое взаимодействие может понадобиться при добавлении элемента из каталога товаров в корзину покупателя. В этом случае компонент "каталог"должен послать сообщение "добавление товара"компоненту "корзина". Транслятор для данного сообщения выглядит следующим образом:

Каталог. Выбрать -> Корзина. Добавить.

Каталог. Выбрать -> Каталог. Понизить приоритет компонента.

Каталог. Выбрать -> Корзина. Повысить приоритет компонента.

Поскольку количество элементов в каталоге должно уменьшиться после добавления элемента в корзину, то необходим еще транслятор:

Каталог. Выбрать -> Каталог. Уменьшить количество.

7.5 Другие компонентно-ориентированные технологии

Существует несколько коммерческих решений, позволяющих реализовать компонентно-ориентированные приложения. На сегодняшний день наиболее популярные из них — ActiveX от Microsoft [2] , Java applets от Sun Microsystems [2], Flash от Macromedia [5]. Эти технологии реализованы при помощи специальных расширений для браузера и не закреплены в стандартах W3C. Браузер при обнаружении какого-либо расширения ищет обработчик, который может выполнять данное расширение. Например, для Java applets обработчиком является Java-машина (Java Runtime Environment). При успешном нахождении необходимого расширения браузер выполняет компонент при помощи найденного обработчика.

У вышеуказанных технологий имеется ряд достоинств для коммерческих приложений. Один из неоспоримых плюсов — закрытость кода, что позволяет реализовывать приложения, код которых не должен попасть в чужие руки. Но с другой стороны для их использования требуется установить специализированные расширения браузера, пользователь не может доверять закрытому коду, виртуальные машины обладают низкой производительностью. Другое достоинство — возможность функционирования независимо отостального документа в браузере. При этом возможны клиент-серверные взаимодействия. Пользователю предоставляется GUI.

В силу того, что расширения браузеров платформозависимы, использование компонентов ограничивается поддерживаемыми платформами. Поскольку компонент выполняется преимущественно на стороне клиента, то возникает необходимость получения кода компонента клиентом, что увеличивает нагрузку на канал.

Вышеперечисленные технологии не получили широкого распространения для создания компонентов в силу ограничений, приведенных выше. Технология разработки компонентов, представленная в данной работе имеет ряд неоспоримых преимуществ перед ранее представленными технологиями. Поскольку кодкомпонента выполняется на сервере, то отпадает возможность кражи данных и алгоритмов компонента. Компонент реализован как стандартный DHTML код, что позволяет выполнять компонентно-ориентированные приложения на всех платформах, поддерживающих DHTML. Поскольку на стороне пользователя к омпонент представляется как обычная форма, то нет угрозы безопасности для окружения пользователя.

Основной недостаток данной технологии — ориентированность на

WebGUI, что лишает ее интерактивно сти, присущей компонентным технологиям, реализованным через сторонние расширения браузеров.

8 Заключение

На текущий момент можно говорить о том, что цели и задачи дипломной работы успешно выполнены. Метафора приложения на сегодняшний день выглядит следующим образом: “HTTP-сервер с расширенной функциональностью”. Некоторые подсистемы не нашли прямого отражения в метафоре в силу того, что они зависят от частей следующих этапов реализации приложения.

Основное достижение данного этапа — инфраструктура целевого приложения. Инфраструктура — фундамент, на котором будут основаны последующие фазы разработки. Подсистема централизованного тестирования сыграла огромную роль на данном этапе, значительно упростив процесс интеграции. Очевидно, что с увеличением количества подсистем, роль данной подсистемы будет расти.

На основе концепций V.D.O.M. был разработан новый подход к созданию Web-приложений. Модель объектов, которая получила развитие названа “Компонентной моделью”, сформулированная в разделе 7 — способ построения приложений, который позволяет значительно упростить процесс разработки приложений для бизнеса.

Компонентно-ориентированные приложения имеют большую перспективу в коммерческих разработках малой и средней величины. Поскольку сфера деятельности многих организаций информационной отрасли нашей страны зачастую связана с выполнением малобюджетных проектов, в которых, в силу жесткой конкуренции, нет времени на создание инфраструктуры приложения, возникает особая необходимость в сокращении издержек проектирования и реализации.

Существующий опыт использования компонентов позволяет утверждать, что соотношение времени реализации функциональности с использованием компонентов и без их использования составляет порядка 1 к 10 (10 минут против 1 час — 2 часа, в зависимости от квалификации разработчика). В настоящее время реализовано порядка 5 приложений с использованием компонента табличного отчета.

Кроме компонента табличного отчета на сегодняшний день реализованы или готовятся к реализации следующие компоненты: авторизация, выход из приложения, экспорт данных, загрузка файлов с ограничениями, регистрация пользователя, каталог товаров, таблица с функциями поиска, галерея изображений, корзина покупателя, манипулятор изображений. Основной трудностью при создании компонента является сложность корректной и безопасной реализации. Однократно реализованный компонент окупается за одно — два приложения с его использованием.

Построение библиотеки качественных компонентов позволит организации значительно сократить стоимость разработки приложений и увеличить прибыль.

Список литературы

[1] Джим Коналлен. Разработка Web-приложений с использованием UML. Вильямс, 2001.

[2] Крис Джамса, Конрад Кинг, Энди Андерсон. Креативный Web-дизайн. HTML, XHTML, CSS, JavaScript, PHP, ASP, ActiveX. Текст, графика, звук и анимация. - ДиаСофтЮП, 2005.

[3] Роман Сузи. Python. СПб.: БХВ-Петербург, 2002.

[4] Grady Booch, James Rambaugh, Ivar Jakobson. The Unified Modelling Language User Guide. Adisson Wesley, 2000.

[5] Janet Burleson. Conducting the Webmaster Job Interview: IT Manager Guide with Javascript, Java applets, Front Page, Flash, Perl, PHP+, and DreamWeaver Interview Questions (IT Job Interview series). - Hungry Minds. 2004.

[6] Scott Guelich, Shishir Gundavaram, Gunter Birznieks. CGI Programming in Perl, O'REILLY, 2000.

[7] Steven Holzner, Inside XSLT, New Riders Publishing, 2002.

[8] Mohammed J. Kabir, Apache Server 2: Bible. Hungry Minds, 2001.

[9] Jenifer Niderst, Web design in a nutshell. O'REILLY, 1999.

[10] http://www.w3.org/

[11] Rakesh Pai. Web Applications - The Wave Of The Future. http://piecesofrakesh.blogspot.com/2005/01/web-applications-wave-of-future.html.

[12] RFC 2616 HTTP1.1 Reference, http://www.megalib.com/books/84/CHMFirstPage.htm