УДК 004.428.4

Баранов Д.А., Влацкая И.В.
Оренбургский Государственный Университет
E-mail: baranov@semograph.com

ИСПОЛЬЗОВАНИЕ СЕТЕВОЙ БАЗЫ ДАННЫХ ДЛЯ ХРАНЕНИЯ СЛОЖНОСТРУКТУРИРОВАННЫХ ДАННЫХ НА СТОРОНЕ КЛИЕНТА В ИНФОРМАЦИОННОЙ СИСТЕМЕ С WEB-ИНТЕРФЕЙСОМ

Развитие технологий Web 2.0 привело к перемещению сложных приложений в Web. Web-приложения обладают рядом преимуществ, однако, текущие ограничения платформы Web 2.0 (в частности, отсутствие поддержки распределённых систем объектов) не позволяют эффективно оперировать сложноструктурированными данными, что затрудняет реализацию информационных систем в виде веб-приложений.

К данным со сложной структурой (сложноструктурированным) относятся массивы, списки, деревья, графы, сети и их комбинации. Проблема состоит в сложности поддержки актуальности веб-страницы и синхронизации с сервером, необходимости передачи больших объёмов данных – это может негативно сказаться на скорости реакции пользовательского интерфейса приложения. Следовательно, важной задачей является минимизация объёма передаваемых данных, необходимых для выполнения той или иной операции и ликвидация повторной загрузки данных.

В качестве решения данной проблемы предлагается ввести отдельный (вспомогательный) уровень данных на стороне клиента. В ходе реализации такого уровня абстракции, решающего описанные выше проблемы, была разработана концепция унифицированного ориентированного графа (UOG, Unified Oriented Graph), представляющая собой частный случай сетевой базы данных.

Ключевые слова: распределённые системы, унифицированный объектный граф, сложноструктурированные данные, базы данных.

После наступления эпохи Web 2.0 и появления новых средств и стандартов разработки web-приложений наблюдается тенденция переноса привычных «настольных» приложений в web. Это обусловлено преимуществами, которые получают пользователи web-приложений нового поколения, например: запуск приложения без установки на клиент (компьютер, мобильное устройство, виртуальную машину) посредством браузера; доступ к персональным данным с любого устройства, поддерживающего используемые стандарты Web 2.0 и доступом в Интернет; интеграция с разнообразными web-сервисами, такими как социальные сети или платёжные системы. Дальше всех в этом направлении продвинулась корпорация Google, анонсировав в июле 2009 операционную систему ChromeOS, весь пользовательский интерфейс которой реализуется в браузере.

Фактически, любое web-приложение представляет собой распределённую информационную систему клиент-серверной архитектуры [1]. При этом архитектура уровней системы может варьироваться. С развитием Web 2.0 все большее число функций переносится на клиент. Это связано с возросшей вычислительной мощностью клиенстких устройств и производительностью браузеров. Если раньше web-сервер генерировал статическую HTML-страницу а задачей клиента была лишь визуализация этой страницы, то теперь сервер, как правило, лишь передаёт браузеру набор данных, где производится конструирование DOM (Document Object Model, Объектная Модель Документа) и визуализация страницы. Помимо снижения нагрузки на сервер, это так же позволяет динамически изменять web-страницу и реализовать интерактивный пользовательский интерфейс. Ключевыми факторами в этом процессе стало внедрение языка программирования JavaScript в браузер, появление технологии AJAX (Asynchronous JavaScript and XML) и концепции RIA (Rich Internet Application — веб-приложения с «насыщенным» интерфейсом). На рисунке 1 изображена диаграмма процесса изменения архитектуры клиент-сервер web-приложений.

Описанные выше преимущества актуальны и для информационных систем. Однако, текущие ограничения платформы Web 2.0 (в частности, отсутствие поддержки распределённых систем объектов) не позволяют эффективно оперировать сложноструктурированными данными, что затрудняет реализацию информационных систем в виде веб-приложений.

К данным со сложной структурой (сложноструктурированным) относятся массивы, списки, деревья, графы, сети и их комбинации [2], [3]. Основная проблема состоит в сложности

поддержки актуальности DOM веб-страницы и синхронизации с сервером. Кроме того, в современных веб-приложениях (особенно, в информационных системах) объёмы данных, с которыми работает пользователь, могут быть очень большими. Передача таких объёмов данных через Интернет может негативно сказаться на скорости реакции пользовательского интерфейса приложения. Следовательно, важной задачей является минимизация объёма передаваемых данных, необходимых для выполнения той или иной операции и ликвидация повторной загрузки данных.

В качестве решения данной проблемы предлагается ввести отдельный (вспомогательный) уровень данных на стороне клиента. Следует отметить, что стандарт HTML5 добавил в браузеры поддержку LocalStorage — реляционной БД, позволяющей веб-приложениям перманентно хранить в браузере произвольные данные. Однако, современные методы разработки информационных систем подразумевают использование средств более высокого уровня, чем прямое общение с БД через SQL, таких как ORM (Object-Relational Mapping, Объектно-Реляционное Отображение) [4]. Тем не менее, LocalStorage предоставляет возможность хранения данных на стороне клиента, что позволяет избежать необходимость повторной загрузки не изменявшихся данных при каждом открытии приложения. Наиболее логичным решением является реализация дополнительного уровня абстракции между LocalStorage и веб-приложением, предоставляющим удобный интерфейс программирования и использующий возможности LocalStorage.

В ходе реализации такого уровня абстракции, решающего описанные выше проблемы, была разработана концепция унифицированного ориентированного графа (UOG, Unified Oriented Graph). UOG представляет собой частный случай сетевой базы данных. Сетевые базы данных менее эффективны в плане скорости доступа и занимаемого объёма по отношению к реляционным БД, однако они позволяют естественным образом представлять сложносвязные структуры данных. Кроме того, схема сетевой базы данных, как правило, не накладывает жёстких ограничений на типы хранимых данных, что может быть полезным при хранении сложных структур данных.

Основной идеей унифицированного ориентированного графа является представление всех полученных с сервера данных в единой структуре (графе, сетевой БД) на клиенте, для обеспечения единого (унифицированного) интерфейса доступа к данным и ассоциативным связям между ними (ориентированность). Для этого ассоциативные связи представляются в виде узлов графа, наряду с прочими данными.

Концепция UOG близка к модели OEM (Object Exchange Model), разработанной для представления слабоструктурированных данных [5, 6]. Однако, в отличии от модели OEM, UOG имеет более жёсткую схему (которая является частью самого графа), позволяющая производить валидацию данных и контролировать их структуру до синхронизации с сервером. Схема

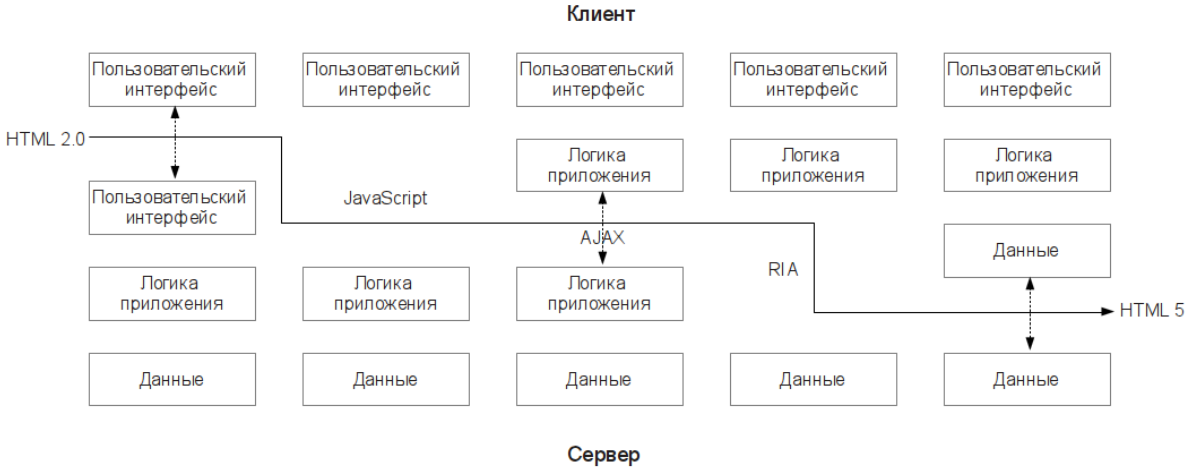

Рисунок 1— Развитие архитектуры клиент-сервер

UOG задаётся априорно и не допускает появления в модели неструктурированных данных.

При использовании концепции унифицированного ориентированного графа можно выделить следующие основные задачи: обмен данными с сервером, выборка узлов, визуализация данных, слежение за обновлениями данных.

Поскольку web-приложения не поддерживают распределённые системы объектов (CORBA, DCOM, GLOBE, SOAP), для решения задачи обмена данными (протокола синхронизации уровней представления данных сервера и клиента) возможно использование одного из способов обмена сообщениями в распределённых системах, таких как XML-RPC, JSON, REST. XML-RPC (Remote Procedure Call, удалённый вызов процедур на основе XML) является одним из первых протоколов, обеспечивающих удалённый вызов функций [1]. Как правило, он не используется в web-приложениях, поскольку с Web 2.0 появились более удобные способы обмена данными. JSON (JavaScript Object Notation) — текстовый формат данных, предоставляющий прозрачную сериализацию основных структур JavaScript и легко читаемый людьми. REST (Representational State Transfer, Передача состояния представления) — набор простых принципов доступа и манипуляции данными через унифицированный интерфейс, реализующий набор операций CRUD (Create, Read, Update, Delete) поверх протокола HTTP [7]. В качестве формата данных REST обычно использует JSON. Подход REST представляется наиболее естественным для синхронизации данных между разными уровнями представления данных.

Основными понятиями REST являются ресурс и коллекция ресурсов. Ресурсом является любая сущность, которой оперирует ИС. Любой ресурс входит в некоторую коллекцию ресурсов, в пределах которой он должен иметь уникальный идентификатор. Например:

/rest/books — коллекция книг;

/rest/books/15 — книга с идентификатором 15;

/rest/books/15/pages — коллекция страниц книги с идентификатором 15;

Следует подчеркнуть, что REST не накладывает никаких ограничений на способ представления адресов ресурсов и коллекций. Так, пример 3 может быть представлен следующим образом:

/rest/pages/15?book_id=15 — коллекция страниц с параметром запроса <идентификатор_книги=15>.

В рассматриваемой задаче необходимо учесть, что нужно предоставить доступ к связям данных на стороне сервера как к ресурсам. Для доступа к связям данных с помощью REST можно использовать три подхода:

Присвоить каждой связи уникальный идентификатор относительно других связей этого же типа. То есть, необходимо завести отдельную коллекцию связей:

/rest/book_has_author/183 — у книги есть автор, связь с идентификатором 183.

Такой подход удобен, поскольку не требует никаких изменений в логике UOG для работы с узлами, хранящими связи. Однако, хранение идентификатора связи влечёт накладные расходы на сервере БД. В реляционных БД связи типа многие-ко-многим, как правило, хранятся в виде отдельной таблцы, соедржащей кортежи (<идентификатор первого связываемого объекта>; <идентификатор второго связываемого объекта>; [<дополнительные поля>]) [4], следовательно, включение идентификатора связи в котреж может увеличить размер записи в 1,5 раза в случае отсутствия дополнительных полей. Учитывая потенциально большое количество связей (относительно обычных сущностей), это серьёзный недостаток данного подхода.

Очевидно, коллекция связей может быть выражена как подколлекция ресурсов, связываемых ею. Однако, такой способ представления оставляет неоднозначность выбора уникального идентификатора связи:

/rest/books/15/authors/3 — автор с идентификатором 3 книги с идентификатором 15.

/rest/authors/3/books/15 — автор с идентификатором 3 книги с идентификатором 15.

Использование параметров строки запроса. Для этого подхода, как и для первого, требуется коллекция связей. Однако, внутри неё ресурсам (связям) не будут присваиваться уникальные идентификаторы. Весто этого обращение к ресурсам будет осуществляться через параметры строки запроса:

/rest/book_has_author?book_id=15&author_id=3 — автор с идентификатором 3 книги с идентификатором 15.

Данный подход лишён недостатков предыдущих подходов, но следует заметить, что приведённый пример фактически является вызовом метода list, т. е. получения коллекции. Фактически, это означает, что результатом выполнения такого запроса будет массив объектов. Однако, уникальность связей должна обеспечиваться на уровне БД.

Очевидно, сложные структуры данных подразумевают сложную логику представления. Для реализации интерфейса таких информационных систем зачастую используют паттерн MVC (Model-View-Controller, Модель-Представление-Контроллер) [8], [9]. Основополагающей идеей данного паттерна является отделение данных от представления. Это означает, что одни и те же данные могут быть одновременно представлены разными способами, при этом обновление данных в модели автоматически вызывает обновление всех представлений этих данных посредством контроллера.

Рассмотрим особенности использования UOG в качестве модели в составе паттерна MVC. При реализации этого паттерна часто бывает необходимо представлять целые коллекции объектов как единое целое. Удобство UOG заключается в возможности динамического формирования произвольных коллекций из узлов графа (срезов) посредством запросов с возможностью наблюдения за изменениями в этих коллекциях. Эта возможность основывается на механизме выборок узлов из графа, описанном ниже. Значительным достоинством данного подхода является возможность повторного использования представлений для визуализации (в том числе одновременной) узлов одного типа, выбранных на основе связей с узлами различных типов, путём простой замены запроса. На рисунке 2 представлена схема клиент-серверного приложения с использованием UOG.

Помимо визуализации, использование UOG в качестве модели упрощает редактирование данных и синхронизацию с сервером, поскольку предоставляет единый интерфейс доступа к данным, их созданию и изменению.

Как уже отмечалось, во многих информационных системах объёмы данных, с которыми

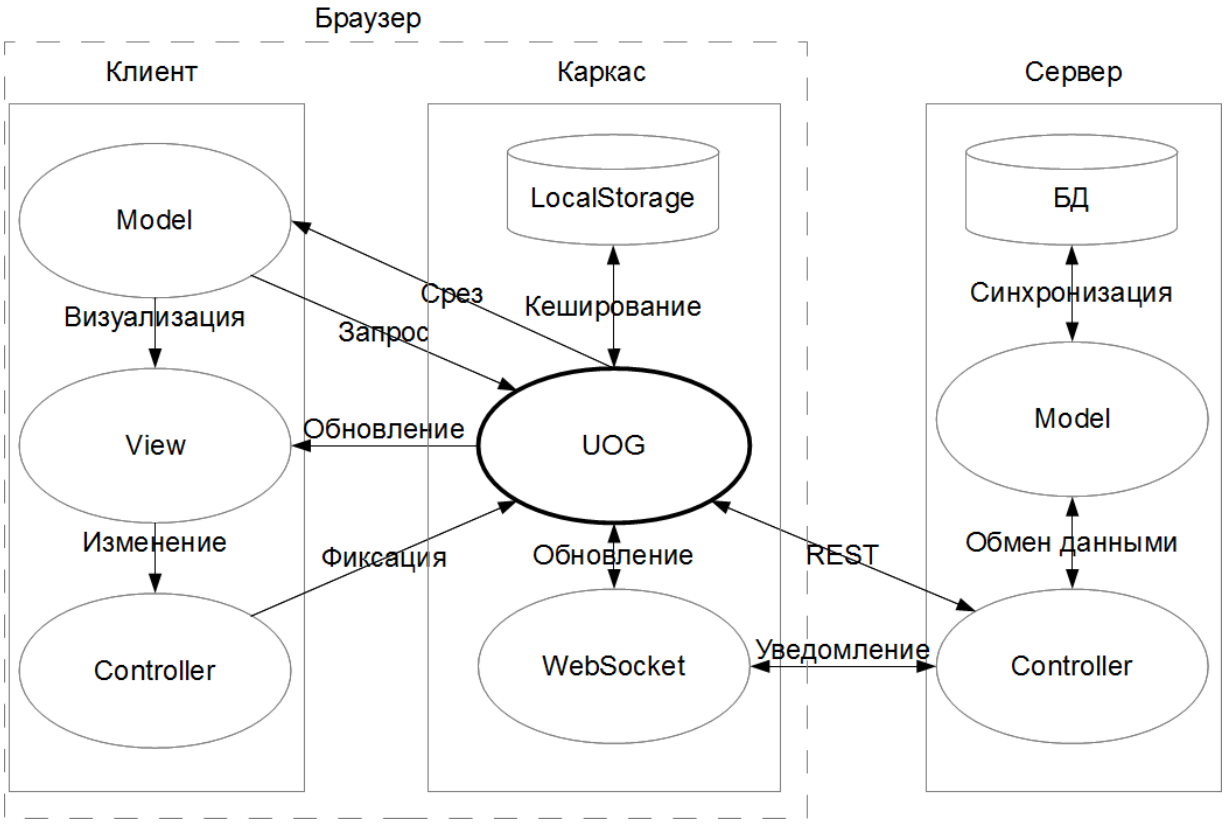

Рисунок 2 — UOG в клиент-серверном приложении

работает пользователь, слишком велики, чтобы загружать их единовременно. Поэтому необходим механизм отложенной загрузки данных, т. е. узлов графа в концепции UOG. Очевидно, в большинсве случаев объём необходимых пользователю данных значительно меньше объёма всех доступных ему данных, следовательно, механизм отложенной загрузки должен загружать только необходимые данные, с минимальной избыточностью, благодаря чему снизится время загрузки (и, как следствие, время реакции интерфейса) и нагрузка на сервер. В рамках концепции UOG были разработаны два подхода к реализации механизма отложенной загрузки данных:

Узлы-пустышки (Dummy Nodes)

Ленивые узлы (Lazy Nodes)

Для выборки узлов из графа используется алгоритм, принимающий на вход объект JavaScript (ассоциативный массив) с параметрами запроса и последовательно анализирующий доступные узлы. Это же механизм используется для наблюдения за новыми и удаляемыми узлами. Для оптимизации поиска узлы могут быть упорядочены по типу и другим вспомогательным полям.

Описанный подход был опробован на второй версии системы графосемантического моделирования «Семограф» [10], [11].

UOG разрабатывалась для хранения данных в ИС «Семограф» на стороне клиента. Первоначально для этих целей использовались классические коллекции (реализация из библиотеки BackboneJS), однако, особенности структур данных «Семографа» значительно усложняли этот подход (ядро структуры схематично представлено на рисунке 3). Такая структура стала следствием денормализации реляционной модели, проведённой с целью устранения дублирования больших объёмов текстовых данных.

Сложность заключается в наличии сложных взаимосвязей между типами данных (например один компонент может одновременно принадлежать проекту, нескольким контекстам этого проекта и нескольким полям этого проекта). Для манипулирования подобными данными были испытаны два подхода:

хранение нескольких экземпляров одного объекта (в данном случае, компонента), используя для поддержания целостности некоторый дескриптор (например, первичный ключ из БД);

использование проекций коллекций (срезов), подразумевает наличие общей коллекции для хранения однотипных объектов и управляющего объекта (агрегирующей коллекции), осуществляющего выборку из общей коллекции необходимых объектов.

Первый подход неудобен из-за возможной потери целостности данных, второй — из-за трудоемкости контроля и синхронизации управляющего объекта, коллекции и необходимости поддержки нескольких объектов управления.

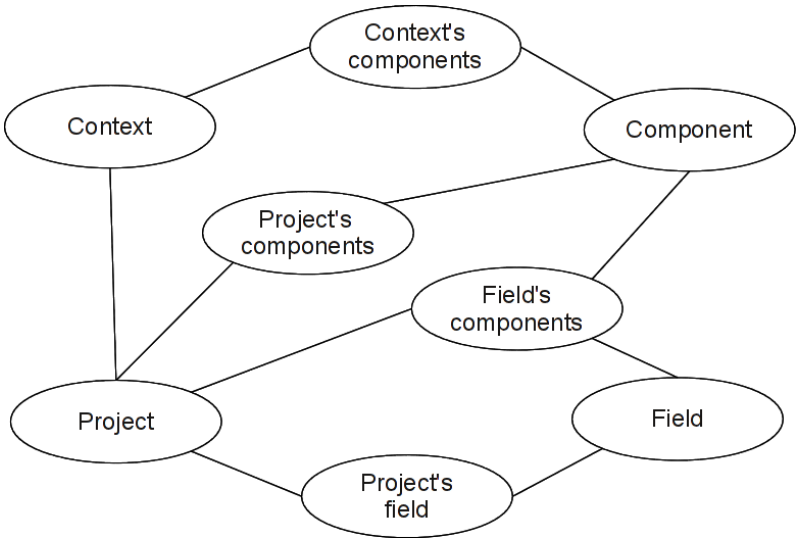

Рисунок 3 — Схема ядра структуры данных ИС «Семограф»

При всестороннем рассмотрении проблемы, в частности, представлении структуры данных в виде, изображённом на рисунке 3, стало очевидно, что самым логичным способом хранения этих данных является представление в виде графа. Эта идея стала основой концепции UOG.

Помимо ядра структуры данных (приведённой на рисунке 3), UOG включает большое количество периферийных и вспомогательных данных, не связанных непосредственно с

реляционной моделью данных ИС. К ним относятся: HTML-шаблоны, элементы интерфейса (окна, их положение и размер), сведения о рабочих столах. Подобное единое представление данных позволило организовать унифицированный интерфейс доступа к любым данным клиента. В ИС «Семограф» этот интерфейс реализован посредством специального языка запросов, основанного на реализации DataJS. Особенности данных системы потребовали существенной доработки и оптимизации интерпретатора запросов.

На данный момент система «Семограф» еще находится в стадии разработки, однако сам процесс разработки подтвердил ожидаемые преимущества UOG: унифицированный интерфейс доступа к данным, перманентное хранение данных на клиенте, автоматическая актуализация представлений данных и т. д.

24.09.2015

Список литературы:

1. Э. Таненбаум, М. ван Стеен, Распределенные системы. Принципы и парадигмы, 2003.
2. Р. Седжвик, Фундаментальные алгоритмы на С. Анализ/Структуры данных/Сортировка/Поиск/Алгоритмы на графах, 2003.
3. М.В. Гладков С.В. Шибанов, Сложные структуры в реляционных базах данных, 2004.
4. K. Roebuck, Object-relational mapping (Orm): High-impact Strategies – What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors, 2011.
5. М. Гринев, Системы управления полуструктурированными данными, 1999.
6. Д. Палей, Моделирование квазиструктурированных данных, 2002
7. J. Webber, S. Parastatidis, I. Robinson, REST in Practice: Hypermedia and Systems Architecture, 2010.
8. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес, Приемы объектно-ориентированного проектирования. Паттерны проектирования, 2007.
9. Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс, Паттерны проектирования, 2011.
10. Семограф. — URL: http://new.semograf.com (дата обращения: 4.08.2013).
11. Система графосемантического моделирования / Д. А. Баранов, К. И. Белоусов, И. В. Влацкая, Н. Л. Зелянская. — М. : Свидетельство о государственной регистрации в Федеральной службе по интеллектуальной собственности, патентам и товарным знакам. Зарегистрировано в Реестре программ для ЭВМ №20111617192 от 15.09.2011.

Сведения об авторах:

Баранов Дмитрий Александрович, преподаватель кафедры компьютерной безопасности и математического обеспечения информационных систем Оренбургского государственного университета
E-mail: baranov@semograph.com

Влацкая Ирина Валерьевна, заведующий кафедрой компьютерной безопасности и математического обеспечения информационных систем Оренбургского государственного университета, кандидат технических наук, доцент
E-mail: irina.vlatskaya@yandex.ru

460018, Оренбург, пр. Победы 13, ауд. 20520, тел.: (3532) 372534