Разработка событийной архитектуры Web Modeling Project

Танков В.Д., математическое обеспечение и администрирование информационных систем, СПбГУ, 344 группа, vdtankov@gmail.com

Брыксин Т.А., доцент кафедры системного программирования, СПбГУ

Аннотация

При реализации инструментов коллаборации в современных Web-редакторах часто возникает необходимость реализации событийной архитектуры, охватывающей всё приложение. Это довольно нетривиальная задача, часто требующая перепроектирования многих модулей проекта. Данная работа на примере реализации событийной архитектуры в Web Modeling Project описывает основные подходы к проектированию такого рода архитектуры.

Введение

В последние годы наблюдается тенденция по переводу настольных (desktop) приложений в Web-среду. Это происходит как с текстовыми редакторами (Microsoft Office 365[1]) и интегрированными средами разработки (Eclipse Che[2]), так и со сложными системами визуального моделирования (LucidChart[3]).

На кафедре системного программирования СПбГУ в течение многих лет ведутся разработки в области DSM-платформ (Domain-Specific Modeling, платформы предметно-ориентированного моделирования). Однако все разработанные ранее инструменты были настольными решениями. Следуя современным тенденциям, было принято решение о разработке Web DSM-платформы — Web Modeling Project.

Была реализована базовая функциональность создания и изменения графовых диаграмм, их хранения и авторизованного доступа к ним одного пользователя. Безусловно, это крайне скупой набор функциональности, не отвечающий всем современным требованиям к Web-приложениям и, в частности, к Web-редакторам.

---

[1] Основной сайт Microsoft Office 365 — `https://products.office.com/`

[2] Основной сайт Ecplipse Che — `http://www.eclipse.org/che/`

[3] Полнофункциональная Web альтернатива Visual Paradigm, основной сайт LucidChart — `https://www.lucidchart.com`

Если обратить внимание на современные Web-редакторы (такие как Google Docs[4], Microsoft Office 365), то важной частью предоставляемых ими функций являются инструменты коллаборации — инструменты для совместной работы над документами. Как правило, это совместное владение документами и их совместное редактирование.

Для реализации такого рода инструментов в Web Modeling Project требовалось реализовать общую событийную архитектуру для всех сервисов, составляющих Web Modeling Project, позволяющую передавать данные о событии изменения диаграммы, произошедшем на одном клиенте, всем остальным заинтересованным клиентам.

Выбор канала передачи событий

Используемые на данный момент в проекте технологии не позволяли осуществлять коммуникацию от сервера к клиенту (по запросу сервера), что требовалось для реализации событийной архитектуры, поэтому необходимо было выбрать технологию для такой коммуникации.

В качестве протокола передачи данных от сервера к пользователю был выбран STOMP over WebSocket[5]. STOMP — простой текстовый протокол сообщений [1]. Он реализует модель издатель-подписчик и позволяет использовать в качестве транспорта любое полнодуплексное надёжное соединение.

В реализации STOMP over WebSocket транспортом для STOMP служит WebSocket [2] — протокол передачи данных, использующий, в свою очередь, в качестве транспорта TCP. WebSocket поддерживается во всех современных браузерах, а свободно распространяемая библиотека SockJs[6] позволяет обеспечить его поддержку и в устаревших браузерах.

WebSocket не потребляет много ресурсов, а соединение устанавливается единожды и на всё время работы клиента. Большим плюсом данного протокола является полнодуплексное соединение, которое позволяет реализовать передачу инкрементальных обновлений.

Также важно отметить, что WebSocket изначально разрабатывался как протокол для обмена сообщениями в реальном времени. Соответственно, задержка передачи данных сведена к минимуму, что чрезвычайно важно для реализации инструментов коллаборации.

---

[4] Основной сайт Google Docs — `https://docs.google.com`

[5] Репозиторий STOMP over WebSocket (stomp.js) на Github — `https://github.com/jmesnil/stomp-websocket`

[6] Репозиторий SockJs на Github — `https://github.com/sockjs`

Создание схемы передачи событий

В случае Web Modeling Project клиенты являются как инициаторами, так и потребителями событий. При этом всякий клиент относится к какой-то группе, заинтересованной в подмножестве всех событий системы (так как каждый клиент заинтересован в событиях только открытой в данный момент диаграммы). Фактически, это канонический пример шаблона проектирования издатель-подписчик[3], часто используемого при реализации событийной архитектуры.

Издателем является клиент, совершивший некоторое изменение диаграммы. Он создаёт сообщение с указанием темы (идентификатора обновлённой диаграммы) и отправляет его брокеру сообщений — Push сервису. Тот сортирует сообщения по темам и отправляет их подписчикам — другим клиентам.

На Рис. 1 изображена диаграмма последовательностей разработанной схемы передачи событий.

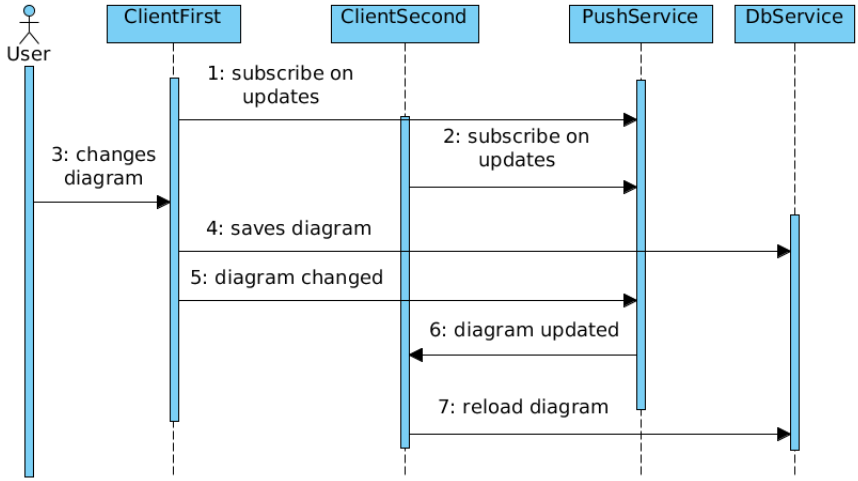

Рис. 1: Схема передачи событий

Каждый клиент подписывается на очередь сообщений открытой в данный момент диаграммы (1, 2). После этого, если пользователь изменяет диаграмму (3), она автоматически сохраняется на сервисе хранения диаграмм (4). Далее клиент уведомляет Push сервис о том, что данная диаграмма была изменена (5). Push сервис добавляет в очередь сообщений для данной диаграммы сообщение о том, что она была изменена, и сообщение рассылается всем

подписанным клиентам (6). Клиенты обновляют диаграмму (7).

Заметим, что мы рассмотрели лишь заголовки событий (сообщающие о типе события) и проигнорировали их тела (сообщающие конкретные параметры события)[4]. В теле события может передаваться дельта изменения, о чём будет сказано далее.

Рассмотрим некоторые аспекты реализации этой схемы.

Реализация Push сервиса

STOMP over WebSocket интерфейс Push сервиса позволяет клиентам подписаться на очередь сообщений интересующей их диаграммы. Все приходящие в данную очередь сообщения рассылаются всем подписчикам в соответствии со STOMP протоколом. При этом нужно отметить, что для каждого клиента постоянно поддерживается WebSocket соединение. Это позволяет снизить нагрузку на клиент, но предъявляет повышенные требования к серверу.

Реализация событийной архитектуры на клиенте

TypeScript не имеет встроенной поддержки событий, а предоставляемые библиотеки пока плохо совместимы с используемым в проекте ECMAScript 6. Поэтому была добавлена простая реализация концепции событий на основе Event классов.

Каждый Event класс — это статический класс со списком функций, которые необходимо вызвать при соответствующем событии. Все заинтересованные в некотором событии объекты могут подписать на него некоторую функцию (как правило, замыкание) и таким образом получить возможность исполнять некоторый код по сигналу о событии. Фактически, это довольно простая реализации концепции событий для TypeScript, языка, на котором реализован код клиента. Создавать её понадобилось, так как TypeScript не имеет встроенной поддержки событий, а предоставляемые библиотеки пока плохо совместимы с используемым в проекте ECMAScript 6.

При такой реализации концепции событий достаточно разместить во всех требуемых местах кода клиента вызовы signalEvent методов у соответствующих классов. Подписанные на данное событие функции будут вызваны автоматически, в частности, на любое изменение диаграммы можно будет отправлять необходимое системное событие.

Инкрементальные обновления диаграммы

Естественно, сохранять и обновлять диаграммы можно разными способами. При нормальной работе пользователь совершает не более десятка событий в секунду и, учитывая то, что WMP на данный момент не поддерживает очень большие диаграммы, вполне приемлем вариант с полной загрузкой и полным же обновлением диаграммы на каждый сигнал. Однако, есть возможность дополнительно снизить нагрузку — применять инкрементальное сохранение и загрузку, передавая дельты изменений в теле события.

Приведём обновлённую диаграмму последовательностей (см. Рис. 2).

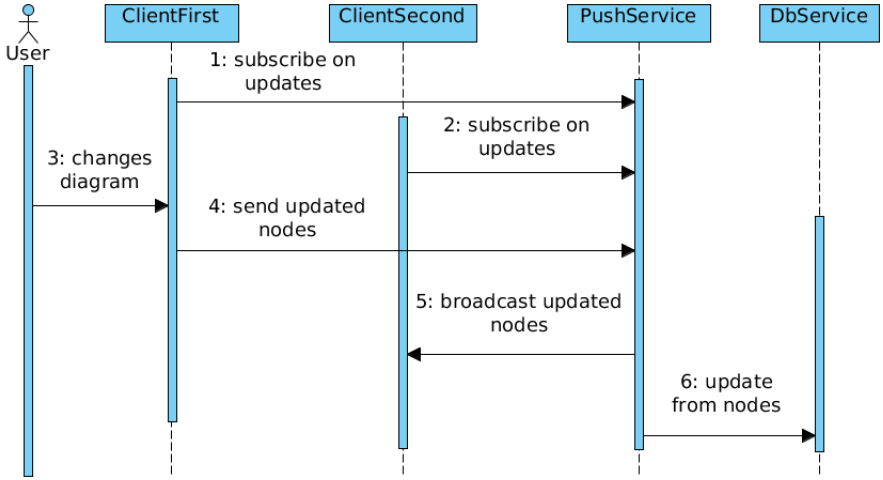

Рис. 2: Схема передачи событий с инкрементальным обновлением

Шаги (1) и (2) полностью соответствуют предыдущей диаграмме. Однако после шага (3) клиент отправляет не просто сигнал об обновлении диаграммы, а всю дельту изменений Push сервису (4). При этом клиенту уже не нужно отправлять отдельно сообщение об обновлении диаграммы. Push сервис, получив дельту, передаёт её всем заинтересованным клиентам (5), которые применяют её к своим диаграммам. Теперь и получателям не нужно перезагружать всю диаграмму. Также Push сервис передаёт дельту сервису хранения диаграмм (6), таким образом обновляя диаграмму в базе данных.

Реализация такой схемы передачи событий не представляет существенных трудностей и была выполнена на основе уже описанной реализации с минимальными изменениям.

Заключение

В данной статье были рассмотрены основные моменты проектирования и реализации событийной архитектуры на примере инструментов коллаборации, реализованных в проекте Web Modeling Project.

Литература

[1] STOMP Protocol Specification, Version 1.2 – 2012 – `https://stomp.github.io/stomp-specification-1.2.html`

[2] Internet Engineering Task Force (IETF). RFC 6455 — The WebSocket Protocol – 2011 – `https://tools.ietf.org/html/rfc6455`

[3] Birman, K. and Joseph, T. Exploiting virtual synchrony in distributed systems. — Proceedings of the eleventh ACM Symposium on Operating systems principles (SOSP '87), 1987. стр. 123-138.

[4] Brenda M. Michelson, Event-Driven Architecture Overview — Patricia Seybold Group, 2006.