Санкт-Петербургский государственный университет
Кафедра системного программирования

Микросервисная архитектура QReal-Web

Курсовая работа студента 344 группы
Безгузикова Артемия Валерьевича

Научный руководитель: ст. преп. Брыксин Т. А.

Санкт-Петербург
2016

Оглавление

Введение……..……..…….……3

Постановка задачи…….…..…….……..……4

1. Обзор существующего решения…….…….5

1.1 Обзор архитектуры серверной части QReal-Web……5

1.2 Обзор архитектуры клиентской части QReal-Web……..……6

1.3 Общая структура приложения…….……..…7

2. Проектирование и анализ новой архитектуры…….……8

2.1 Микросервисная архитектура……..…….…….……8

2.2 Анализ инструментария……10

3. Реализация…….…..…..…12

3.1 Наращивание существующего кода…….……..……..…12

3.2 Добавление сервисов к новому проекту..…….…….…13

3.3 Реализованные компоненты……..……..……..…14

Заключение…….……17

Результаты…….……..…..……17

Направления дальнейшей работы…….……..…..…17

Список литературы…….……..……..…..18

Введение

Ни для кого не секрет, что в последнее время повышенное внимание привлекают роботы. Прогресс не стоит на месте, и с каждым годом робототехника набирает все большую популярность. Зачастую людям хочется не просто иметь робота, исполняющего заданные функции, но и уметь его программировать, уметь задавать его поведение. К сожалению, этот процесс нетривиален и требует серьезных навыков даже от квалифицированных инженеров. Чтобы как-то упростить процесс разработки, была создана визуальная среда программирования роботов TRIK Studio[1]. Данное программное обеспечение предоставляет возможность писать программы на исполнение в виде диаграмм – последовательности картинок, соединенных линиями. TRIK Studio имеет довольно широкий функционал, включая запуск программы на самом роботе, моделирование поведения робота в 2D модели, пошаговую интерпретацию и многое другое. Сейчас система продолжает развиваться, дописываются новые модули, расширяется функциональность. Однако существует и стабильно работающая версия, которую можно скачать на сайте.

В наши дни доступ в интернет есть практически везде. Учитывая огромное разнообразие устройств, позволяющих воспользоваться ресурсами всемирной паутины, логичным следующим шагом был перенос проекта в интернет пространство. Так родился QReal-Web – браузерная версия TRIK Studio. Проект довольно молодой, и первые разработки начались два года назад. На данный момент реализован редактор диаграмм, связь с роботом и возможность отправлять ему диаграммы на исполнение, а так же функционал 2D модели. QReal-Web продолжает разрабатываться и сейчас, причем над проектом работает довольно много студентов кафедры системного программирования СПбГУ. Над TRIK Studio трудятся уже много лет, его разработка продолжается до сих пор. Также активно ведется работа над

[1] http://blog.trikset.com/p/trik-studio.html

браузерной версией, которая несомненно нуждается в доработках и наращивании функциональности.

Основная проблема состоит в том, что компоненты QReal-Web очень сильно завязаны между собой. Реализация, существующая на данном этапе развития, имеет очень много непрозрачных внутренних зависимостей, и программист, работающий над одной конкретной компонентой, вынужден разбираться в структуре проекта целиком. Также разработчик вынужден знакомиться со всеми инструментами, библиотеками и конструкциями, используемыми в QReal-Web, каких немало. В связи с этим возникает потребность облегчить этот процесс и предоставить прозрачный механизм расширения и дополнения функционала.

Постановка задачи

Целью данной курсовой работы является разработка новой архитектуры QReal-Web, позволяющей минимизировать число зависимостей внутри приложения и, тем самым, сделать проще и прозрачнее процесс разработки.

Для достижения поставленной цели были сформулированы следующие задачи:

- анализ текущей архитектуры QReal-Web;
- проектирование и анализ новой архитектуры;
- анализ инструментария для реализации выбранной архитектуры;
- реализация серверной части;
- реализация клиентской части.

1. Обзор существующего решения

В этом разделе рассматривается имеющаяся на данный момент архитектура. Производится анализ серверной и клиентской частей приложения с целью декомпозиции кода и выделении используемых структур, а также основных библиотек и инструментариев. После, производится анализ архитектуры целостного приложения - взаимодействие клиента и сервера.

1.1 Обзор архитектуры серверной части Qreal-Web

В качестве основного языка программирования используется Java, с использованием технологии JavaBeans [16]. В качестве каркаса веб-приложения используется Spring MVC Framework [9], реализующий шаблон проектирования Model-View-Controller [10]. Этот шаблон проектирования помогает отделить презентацию от бизнес-логики.

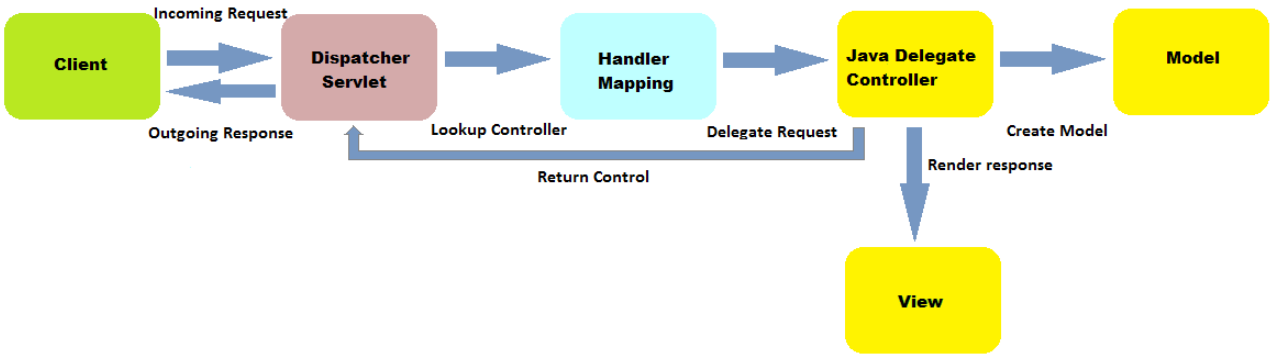

рис. 1: Spring MVC

Все веб-запросы обрабатываются контроллером, который и выбирает соответствующий данному запросу обработчик. Обработчик, в свою очередь, отвечает за взаимодействие объектов (model) и решает, на какое представление (view) нужно будет перейти. Выбранное представление использует данные модели для создания презентации (jsp страница), которая отображается

пользователю. Одним из главных преимуществ Spring MVC Framework с точки зрения обработки запросов является предоставление готовой инфраструктуры.

За сборку проекта отвечает система Maven [1]. Этот инструмент предоставляет возможность описать проект, различные внутренние зависимости и используемые в нем сторонние библиотеки посредством xml файла. Все объявленные инструментарии подгружаются во время сборки. Для конвертации Java-модели в формат JSON была взята библиотека Jackson [10]. Для объектно-реляционного отображения выбрана библиотека Hibernate [11]. Она является довольно популярным и удобным средством отображения Java-классов в структуры базы данных, причем поддерживает задание данного отображения как посредством xml, так и используя аннотации. Также Hibernate предоставляет средства для автоматического построения запросов. Для управления базой данных используется СУБД MySQL. В качестве веб-сервера был взят Apache Tomcat.

Также на серверной стороне реализован парсер, позволяющий получить системные настройки для роботов, написанный на языке программирования Groovy [15].

1.2 Обзор архитектуры клиентской части Qreal-Web.

На стороне клиента реализованы страницы регистрации и аутентификации, редактор диаграмм, возможность просматривать, добавлять и редактировать роботов и функционал 2D модели. Также на клиентской стороне был написан интерпретатор и несколько обслуживающих менеджеров. В качестве основного языка программирования используется TypeScript [17]. Данный язык позиционируется как средство разработки веб-приложений, расширяющее возможности JavaScript, и компилируется в него. В качестве основного каркаса разработки был взят AngularJS [18], который расширяет браузерные приложения на основе MVC шаблона. Данный каркас расширяет HTML путем ввода дополнительных директив и отделяет логику приложения от его представления. Также в клиентской части используется библиотека jQuery [19]. Для реализации диаграмм поведения роботов использовалась библиотека

JointJs [9]. Для реализации двумерной модели поведения была выбрана библиотеку Raphaël, которая упрощает работу с векторной графикой. Для сохранения/загрузки состояния диаграммы посылается соответствующий Ajax запрос на сервер с сериализованными в формат JSON данными. Для разбиения функциональности, отвечающей за управление редактором диаграмм и двумерной моделью, были созданы два отдельных контроллера AngularJS, наследуемые от корневого.

1.3 Общая структура приложения

Схематически, текущая реализация имеет следующую архитектуру:

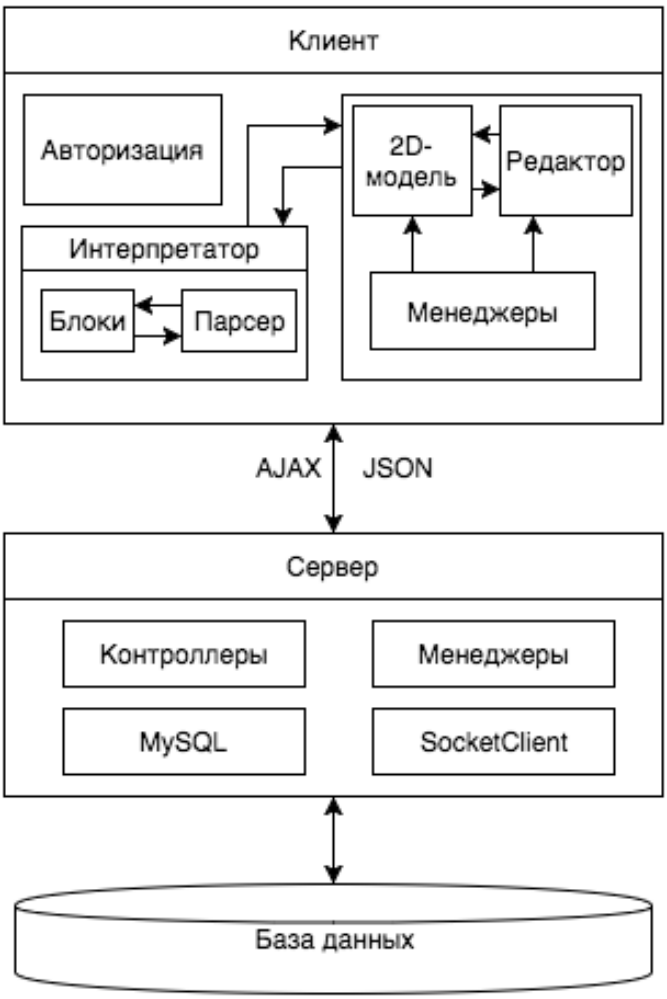

рис. 2: архитектура QReal-Web

Как видно из схемы, приложение имеет монолитную структуру, где функционал реализуется посредством построения внутренних связей и

определения зависимостей между объектами, как на серверной части, так и на клиентской. Следует отметить, что серверная часть QReal-Web реализует паттерн MVC, причем некоторые представления, как клиентские части, на внутреннем уровне также реализуют паттерн MVC. Добавляя к этому огромное количество различных библиотек и функциональности, мы получаем довольно сложную внутреннюю архитектуру, что, конечно, затрудняет процесс совместной разработки.

2. Проектирование и анализ новой архитектуры

Идея использования микросервисов приобретает все большую популярность в мире, она подразумевает деление приложения на небольшие функциональные модули. Данная архитектурная модель подразумевает масштабируемость и гибкость системы, а также способствует выбору лучших вариантов технологий для каждой конкретной компоненты. Рассмотрим эту идею подробнее.

2.1 Микросервисная архитектура

Микросервисная архитектура [4, 5] – это архитектура, где приложение представляется в виде независимых компонент, которые общаются между собой определенным образом. Каждый сервис должен фокусироваться на конкретной задаче, и только на ней. Они не должны иметь сложную и нагруженную внутреннюю структуру. Предлагается наличие простого механизма регистрации новых сервисов и замены уже имеющихся.

На рисунке 3 хорошо иллюстрируется идейное отличие микросервисной архитектуры от монолитной – разделение приложения на независимые модули.

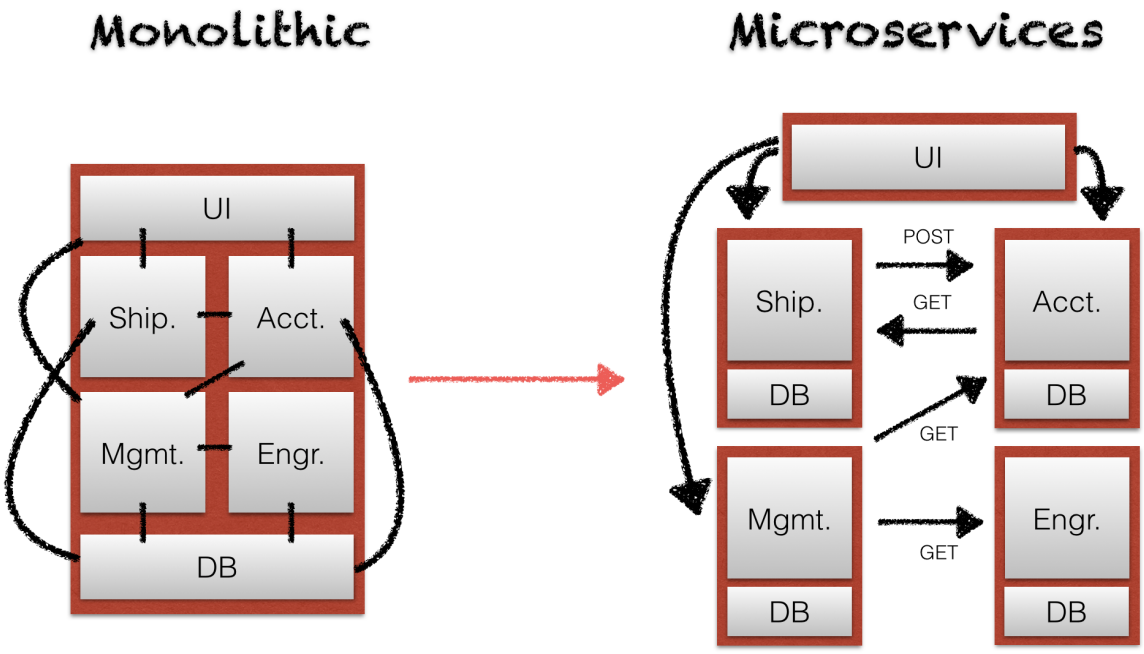

рис. 3: переход от монолитной к микросервисной архитектуре

Таким образом, в отдельные сервисы можно выделить репозиторий диаграмм, операции с данными о пользователях, манипуляции с роботами, редактор диаграмм, работу с 2D моделью и другое. Данное архитектурное решение позволяет рассматривать typesript файл как клиента, который общается с нужными ему сервисами. Кроме того, предполагается разработка относительно простого способа создания нового сервиса и интеграция его в проект.

Преимущества использования микросервисов.

- Писать и поддерживать небольшие сервисы всегда проще, чем большие. Чем меньше кода, тем легче уместить его в голове.
- Любой модуль может быть относительно просто заменен или переписан отдельно от всего остального кода.

- Каждый микросервис должен выполнять поставленную задачу и ничего больше. Таким образом, происходит четкое разделение функциональности.
- Независимость. Общение между сервисами строго структурировано и происходит посредством реализации прописанных интерфейсов.

Основываясь на этих фактах, видно, что использование микросервисов решает поставленные задачи. Данный архитектурный тип позволит избавиться от внутренних зависимостей, поскольку взаимодействие происходит только посредством удаленных вызовов. Благодаря строгой модульности, появится возможность разделить функциональность и свести задачу программиста к простому написанию сервисов, что избавит от необходимости разбираться и держать в голове все нюансы структуры большого монолитного приложения.

2.2 Анализ инструментария

В качестве каркаса для разработки был выбран Apache Thrift[2,3]. Это кросс-языковой инструментарий для создания сервисов, разработанный и активно используемый компанией Facebook. Thrift предоставляет свой язык описания интерфейсов служб и по данному описанию генерирует код для разработки этих служб, которые эффективно и легко работают между такими языками, как C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, JavaScript, TypeScript, Node.js, Smalltalk, OCaml, Delphi и др.

Преимущества использования Thrift:

- Кросс-языковая сериализация с более низкими накладными расходами.
- Простая и чистая библиотека. Не использует XML-конфигурацию.
- Языковые привязки ощущаются естественными. Например, Java использует ArrayList<String>. C++ использует std::vector<std::string>.
- Формат связи уровня приложений и формат связи уровня сериализации строго разделены. Они могут быть изменены независимо друг от друга.

Thrift предоставляет довольно простой алгоритм создания сервисов:

1. описываем требуемую функциональность и все необходимые структуры на специальном thrift языке[2].
2. производим компиляцию на нужный нам язык программирования. Например, для java это « thrift -gen java <file-name>.thrift ».
3. прописываем обработчик к сгенерированному классу.

Теперь, чтобы сделать наш сервис доступным для запросов, пропишем соответствующий сервлет[3], который и будет запущен на Web-Сервере.

Таким образом мы описали серверную часть. Компилируя thift-код, например в javascript, сгенерируется файл <file-name>.js, где так же будет прописан весь необходимый функционал для взаимодействия со стороны клиента. Остается только обратиться к нашему сервису[4], что требует не больше пары строчек кода.

---

[2] https://diwakergupta.github.io/thrift-missing-guide/

[3] http://www.tutorialspoint.com/servlets/

[4] https://thrift.apache.org/tutorial/js

3. Реализация

3.1 Наращивание существующего кода

Переходить к новой архитектуре можно двумя способами: преобразовывать текущую версию QReal-Web, постепенно добавляя модули там, где их можно добавить, и переписывать приложения с нуля.

Было реализовано несколько тестовых сервисов и налажено их взаимодействие с QReal-Web. Spring framework предполагалось сохранить в качестве основного каркаса разработки, так как при его отсутствии, фактически пришлось бы переписывать все приложение. Учитывая этот факт, запросы с клиентской части по прежнему направлялись в контроллер, который выступал в роле некоторого шлюза, перенаправляющего уже полученные данные конкретным сервисам.

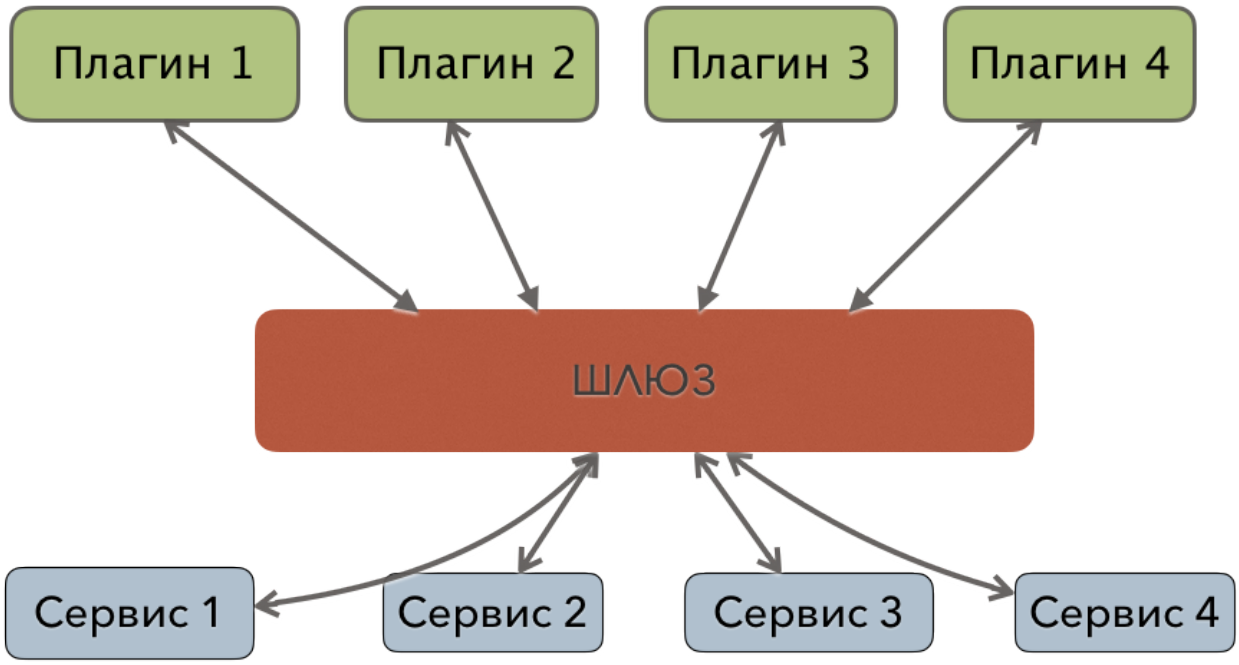

рис. 4: Использование шлюза

Такой подход позволял надстраивать новые сервисы, но не раскрывал всех возможностей Thrift. Структура приложения как была сложной, так и оставалась. Программисту и в этом случае все равно пришлось бы потратить немало усилий, чтобы разобраться во всех архитектурных особенностях.

3.2 Добавление сервисов к новому проекту

В отличие от предыдущего решения предполагается написание приложения с нуля с постепенным добавлением все новых и новых сервисов, пока необходимая функциональность не будет достигнута. Клиентская часть в свою очередь также делится на слабо связанные части – плагины. Под плагинами будем понимать набор typescript или javascript файлов. Каждый плагин несет в себе свою определенную функцию и реализует свою какую-то задачу.

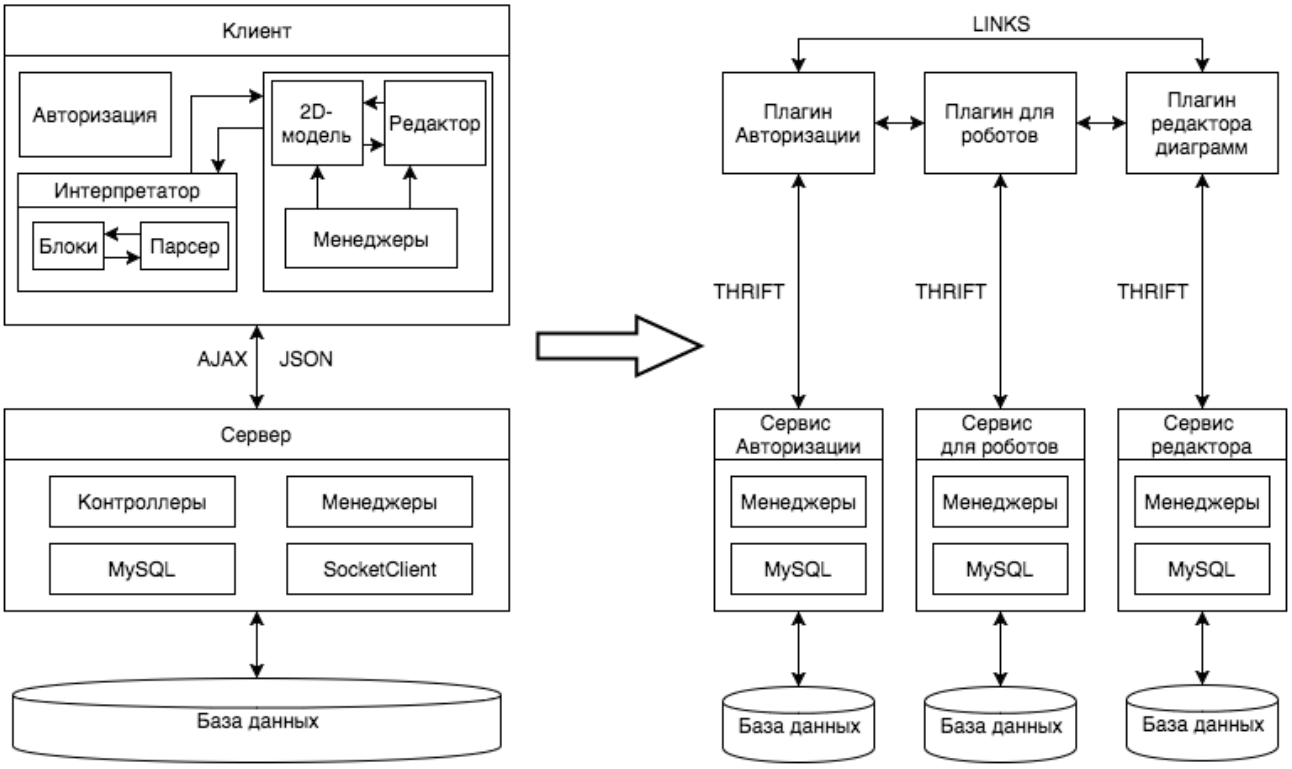

рис. 5: Переход к новой архитектуре

На рисунке 5 изображен переход от предыдущей (монолитной) архитектуры к новой (микросервисной). Была произведена декомпозиция и клиентской и серверной частей приложения с целью выделения отдельных функциональных блоков. Далее был произведен реинжиниринг кода этих блоков, и на стороне клиента были получены плагины, а на стороне сервера - микросервисы.

Теперь, чтобы добавить новый плагин, достаточно просто создать папку и положить туда javascript или typescript файлы. Сервисы же прописываются по следующему алгоритму:

1. создается интерфейс и все необходимые структуры в файле .thrift;
2. прописываются обработчики процедур сервиса;
3. объявляется сервлет с нужным нам url.

3.3 Реализованные компоненты

На данном этапе были реализованы плагины для авторизации, панели для роботов и редактора диаграмм. Также были прописаны сервисы, обслуживающие их. К этим сервисам объявляются клиенты, которые и осуществляют вызовы процедур.

Плагин авторизации является AngularJs приложением, где взаимодействие с соответствующим сервисом происходит внутри контроллера. Теперь не требуется никакого шлюза, и связь осуществляется непосредственно между сервисом, который работает с таблицей пользователей, и данным плагином. При описании в thrift файле структуры User появляется возможность работать с этой структурой как со стороны сервиса (принимать ее как параметр процедуры), так и со стороны клиента (оборачивать в нее данные и отправлять сервису).

Сервис для просмотра роботов позволяет добавлять и удалять роботов с панели инструментов. Соответствующий ему плагин имеет структуру AngularJs приложения и взаимодействие происходит так же из контроллера.

Чтобы сохранить робота, раньше приходилось вручную конструировать запрос и отправлять его с помощью AJAX, теперь достаточно создать клиент и вызвать удаленную функцию.

```
var name = $('#robotName').val();
var code = $('#ssid').val();
var transport = new Thrift.TXHRTransport("http://localhost:8080/RegisterServlet");
var protocol  = new Thrift.TJSONProtocol(transport);
var client = new RegisterRobotServiceClient(protocol);
var result = client.registerRobot(name, code);
```

Для реализации сервиса редактора диаграмм в thrift файле потребовалось задать следующую систему структур:

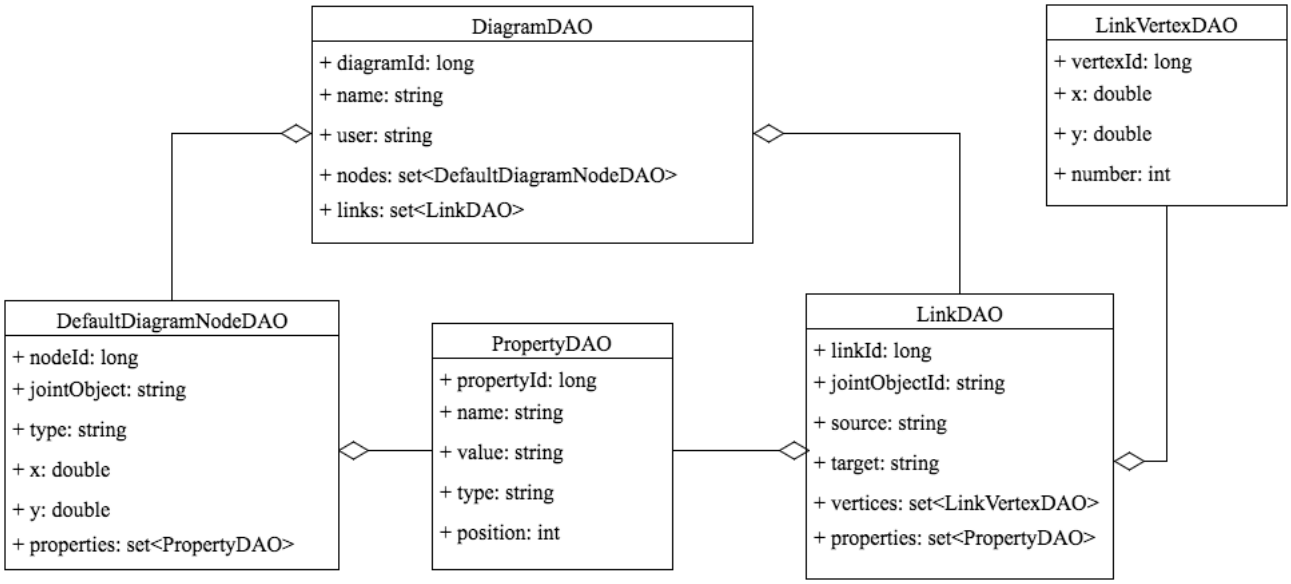

рис. 6: Структуры для DiagramService

Эти структуры удобно задавать именно в .thrift файле, ввиду возможности их использования как на клиентской, так и на серверной частях. То есть генерация кода осуществляется как на java, так и на typescript. Плагин редактора диаграмм позволяет создавать и редактировать диаграммы, тем самым полностью реализуя требуемую функциональность.

Сервис же описывается следующим образом:

```
service DiagramService {
	bool save(1:DiagramDAO dia);
	DiagramDAO open(1:string name);
}
```

и осуществляет взаимодействие с базой данных. Как уже было сказано, если описать необходимые структуры в .thrift файле, то исчезнет необходимость задавать эти классы вручную.

Дополнительно, в проекте были реализованы несколько менеджеров, обслуживающих основную функциональность и SystemConfigService для получения системных настроек для робота.

Заключение

Результаты

В рамках курсовой работы были получены следующие результаты:

- произведена декомпозиция существующей версии QReal-Web;
- спроектирована новая архитектура;
- изучена возможность применения Apache Thrift для создания микросервисов;
- реализованы плагины для аутентификации, редактирования диаграмм и панели инструментов;
- реализованы соответствующие сервисы.

Направления дальнейшей работы

Первое и самое важное - интеграция с текущей версией QRreal-web. Данная курсовая работа велась с актуальной версией на июль 2015 года, и за это время проект успел довольно сильно поменяться. Конечно, предстоит еще много работы по переносу реализованной идеи на свежую версию. Следующим шагом планируется интеграция с другими продуктами данной области (TRIK Studio, курсы на Stepic, и др.). Если учесть, что Apache Thrift кросс-языковой, можно одинаково легко наладить взаимодействие как с программами, написанными на java и javascript, так и с программами на C#, C++ и на многих других языках.

Список литературы

[1] Документация Apache Maven

url: https://maven.apache.org/pom.html, дата обращения: 01.01.2016

[2] Документация Apache Thrift

url: https://thrift.apache.org/docs, , дата обращения: 01.01.2016

[3] Mark Slee, Aditya Agarwal, Marc Kwiatkowski.

Thrift: Scalable Cross-Language Services Implementation.

Facebook, 156 University Ave, Palo Alto, CA

url: https://thrift.apache.org/static/files/thrift-20070401.pdf,

дата обращения: 01.01.2016

[4] Jakob Jenkov. SOA - Service Oriented Architecture:

url: http://tutorials.jenkov.com/soa/index.html, дата обращения: 01.01.2016

[5] James Lewis, Martin Fowler. Microservices:

url: http://martinfowler.com/articles/microservices.html,

дата обращения: 01.01.2016

[6] Дэвид Гери, Кей Хорстман. JavaServer Faces — 3-е издание, Вильямс, 2011

[7] Документация Java EE 7

url: http://docs.oracle.com/javaee/7/tutorial, дата обращения: 01.01.2016

[8] Документация JointJs

url: http://www.jointjs.com/api, дата обращения: 01.01.2016

[9] Wheeler Willie, White Joshua. Spring in Practice.

— Manning Publications Co, 2013.

[10] Документация Jackson

url: http://wiki.fasterxml.com/JacksonDocumentation, дата обращения: 01.01.2016

[11] Christian Bauer, Gavin King. Java Persistence with Hibernate

— Manning Publications Co, 2006

[12] Э. Гамма Р. Хелм Р. Джонсон Дж. Влиссидес. Приемы объектно ориентированного проектирования. — Санкт-Петербург: Питер, 2001

[13] Захаров В. А. Разработка онлайн среды графического программирования роботов: бакалаврская работа — Санкт-Петербург, 2014.

[14] Агеев Д.В. Создание онлайн инфраструктуры для конфигурирования и управления роботами: бакалаврская работа — Санкт-Петербург, 2014.

[15] Документация Groovy: http://docs.smartthings.com/en/latest/getting-started/groovy-basics.html, дата обращения: 01.01.2016

[16] Robert Englander. Developing Java Beans. — O'Reilly Media, 1997

[17] Документация TypeScript
url: https://www.typescriptlang.org/docs/tutorial.html, дата обращения: 01.01.2016

[18] Документация AngularJs
url: https://docs.angularjs.org/api, дата обращения: 01.01.2016

[19] Документация jQuery
url: https://api.jquery.com/, дата обращения: 01.01.2016