

## Способы организации межсервисного взаимодействия

*Вихляев Дмитрий Романович*

*Приамурский государственный университет имени Шолом-Алейхема*

*Студент*

### Аннотация

В данной работе описываются способы организации межсервисного взаимодействия. В содержании представлены сравнительные характеристики синхронного и асинхронного взаимодействия между клиентом и сервером. Приведены примеры организации «запрос-ответ» и событийной моделей. Результаты работы позволяют определить лучшее архитектурное решения при построении сервисов для организации передачи данных.

**Ключевые слова:** микросервисная архитектура, межсервисное взаимодействие, событийная модель.

## Methods for organizing interservice interaction

*Vikhlyayev Dmitry Romanovich*

*Sholom-Aleichem Priamursky State University*

*Student*

### Abstract

This paper describes methods for organizing interservice interactions. It compares the characteristics of synchronous and asynchronous interactions between the client and server. Examples of request-response and event-driven models are provided. The results of this paper allow us to identify the best architectural solution for building data transfer services.

**Keywords:** microservice architecture, interservice interaction, event model.

## 1 Введение

### 1.1 Актуальность

Чтобы успешно построить систему на принципах микросервисной архитектуры важно правильно выбрать способы интеграции. В число способов взаимодействия включены виды синхронного и асинхронного взаимодействия, оркестрация, хореография, текстовые и бинарные сообщения, а также метод протокола. Выбор протоколов и способов взаимодействия в первую очередь зависит от того какие задачи решает система, какие требования нужно реализовать и какую функциональность сделать. Помимо этого, требуется учитывать, как выбранный способ интеграции обеспечивают обратную совместимость, технологическую зависимость, заботу о потребителях и возможность предоставить чёткий интерфейс. Добавление и изменение функциональности должны происходить таким образом, чтобы

изменения не влияли на потребителей, а обратную несовместимость требуется избегать или оттягивать как можно дальше.

### **1.2 Обзор исследований**

Статья А.М.Шитько посвящена изучению технологии построения микросервисной архитектуры, которая позволяет сохранять модульность и независимость развертывания компонент Enterprise приложения [1]. В работе Д.С.Кугушева дано описание принципа работы микросервисной архитектуры, приведены ее отличительные особенности, позволяющие существенно упростить процесс разработки больших продуктов, а также упомянуты основные технологические особенности микросервисного подхода, которые необходимо учитывать при выборе этой технологии разработки. [2]. В исследовании М.Э.Джалалова особое внимание уделяется шаблонам API Gateway, Backend For Frontend и Circuit Breaker, их реализации и оптимизации в контексте микросервисных архитектур. [3]. В качестве результата исследования А.Абдирашитулы, стала предложенная концепция LLM-ориентированной микросервисной архитектуры, обеспечивающая изоляцию ресурсов и независимое масштабирование компонентов. [4]. В статье М.Г.Городничев и Р.В.Полонский сделан обзор типов современных веб-приложений и архитектуры, используемой для их разработки, дано сравнение микросервисной и монолитной архитектуры [5]. М.Н.Карпович исследовал особенности проектирования микросервисно-событийных архитектур для высоконагруженных распределенных систем обработки информации [6]. С.В.Маличенко рассмотрел проблемы перехода от монолитной к микросервисной архитектуре [7]. Г.Р.Берайлик описал способы спроектирования информационных систем с помощью микросервисной архитектуры [8].

### **1.3 Цель исследования**

Цель исследования – описать способы и представить примеры организации межсервисного взаимодействия

## **2 Материалы и методы**

В материалах приводятся способы организации межсервисного взаимодействия для событийной модели через хореографию и для модели «запрос-ответ» через оркестрацию.

## **3 Результаты и обсуждения**

При развитии системы могут появляться новые сервисы, создаваться новые команды. И выбор способа и технологии взаимодействия не ограниченный определённым стеком, позволит реализовать в полной мере преимущества микросервисной архитектуры. Потому что тогда можно будет создать сервис на том стеке технологии, который больше подходит для решения конкретной задачи. Тем не менее, межсервисное взаимодействие

является общим в связи, с чем требуется упрощение использования внешних интерфейсов сервисов.

Разработчику должно быть просто, понятно и быстро начать использовать внешний интерфейс сервиса. Для этого используется документация с инструментами ручного вызова, например, «Swagger». И из названий методов и параметров API понятно, что будет происходить при вызове конкретного метода. А также готовые клиентские библиотеки под разные технологические стеки. Нельзя позволить потребителям сервиса зависеть от внутренней реализации. В этом случае повышается общая связанность системы. И при изменении сервиса придётся менять и всех его потребителей.

Существуют различные способы межсервисного взаимодействия. При синхронной коммуникации клиент, отправивший запрос, остаётся заблокированным до получения ответа. Это гораздо проще воспринимать программистам (рис.1).

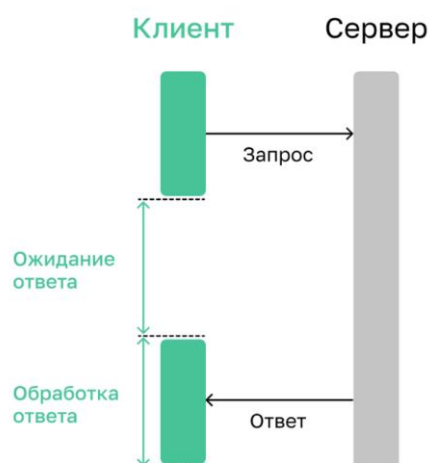


Рис. 1. Синхронное взаимодействие

В случае асинхронной коммуникации клиент, отправивший запрос, может продолжать работать. При этом ответ сервис пришлёт в каком-то будущем промежутке времени. И его уже можно будет проанализировать, обработать и, соответственно, продолжить выполнение алгоритма. Асинхронное взаимодействие выполняют в том случае, если нужно экономить ресурс. Что важно при длительной операции, так как впустую не тратятся ресурсы и соединение не держится открытым (рис.2).

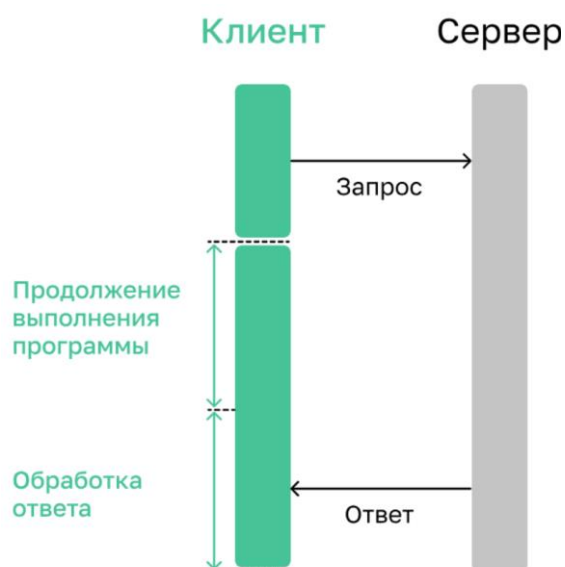


Рис. 2. Асинхронное взаимодействие

Существует два принципиально разных подхода к взаимодействию между сервисами.

Первый подход «запрос-ответ», предполагает наличие клиента и сервиса. Клиент посылает запрос, а сервис обрабатывает этот запрос и присылает обратный ответ. В связи с простотой «Запрос-ответ» может быть реализован как с помощью асинхронного, так и с помощью синхронного взаимодействия. При этом негативным моментом является то, что запрос-ответ ведёт к централизации логики и тем самым увеличивает общую связанность системы.

Вторым подходом является «событийная модель». В данном случае сервис, в процессе выполнения функционала высылает событие о том, что произошло. Другие сервисы подписываются на эти события и реагируют на них. При этом сервис, который отправляет эти события, никаким образом не знает о том, как именно реагируют клиенты на события. Это позволяет с минимальными усилиями расширять систему. Уменьшение связности и децентрализация логики упрощают расширение системы. Но при этом общая сложность системы существенно растёт.

В процессе развития системы появляется бизнес-функциональность, которая требует взаимодействия нескольких сервисов. Для этого используют один из двух подходов: оркестрация или хореография. В оркестрации есть один сервис, который управляет взаимодействием остальных для реализации определённой функции или алгоритма. Главный сервис определяет, что и когда будут делать остальные тем самым управляя бизнес-процессом. Такой подход, концентрирует бизнес-логику в каком-то одном месте и, ведёт к повышению связанности системы (рис.3).

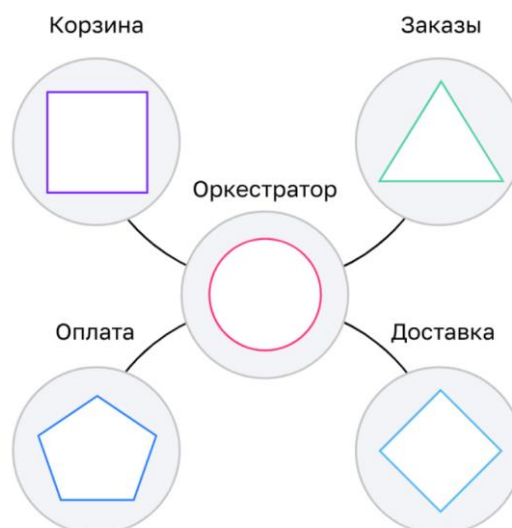


Рис. 3. Взаимодействие между сервисами через оркестрацию

Альтернативой такому подходу является хореография. Чаще всего хореография основана на событийной модели, когда сервисы реагируют на события, происходящие в других сервисах, и реализуют свой кусочек бизнес-функционала. Такой подход распределяет логику по всей системе, исключая единого места, где можно было посмотреть последовательность действий и убедиться в том, что алгоритм правильно настроен. Поэтому в случае использования хореографии система существенно усложняется и требует внедрения дополнительных контролей. А именно систем, которые бы следили за корректностью выполнения бизнес-процесса, о том, что все части его выполнены, что все сервисы выполнили свою часть работ. Поэтому, хоть хореография и существенно уменьшает общую связанность системы, но гораздо выгоднее использовать оркестрацию (рис. 4).

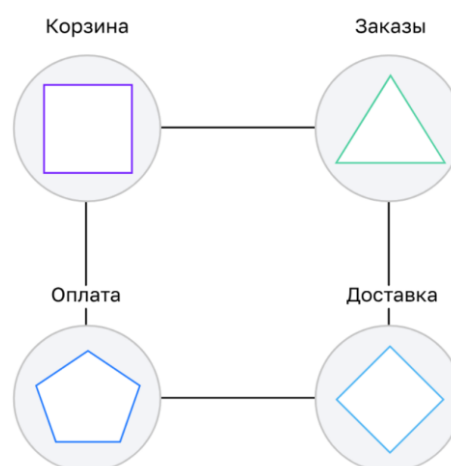


Рис. 4. Взаимодействие между сервисами через хореографию

Событийная модель полностью асинхронна. Главное чтобы сервис мог отправлять сообщения, а клиенты получать. В этом случае используются брокеры сообщений. Брокер занимается только доставкой сообщений до

потребителей. Обработкой и маршрутизацией занимаются конечные сервисы.

#### **4 Вывод**

Таким образом, необходимо избегать любых технологий, которые требуют раскрытия внутренней реализации во внешних контрактах. Наилучшим решением при добавлении новых функций является избежание обратно несовместимых изменений. Что бы упростить контроль и мониторинг за всей совокупностью системы следует дать предпочтение оркестрации, нежели хореографии. Всё это не правила, а скорее признаки, по которым можно выбирать интеграционную технологию для межсервисного взаимодействия.

#### **Библиографический список**

1. Шитько А.М. Проектирование микросервисной архитектуры программного обеспечения //Труды БГТУ. Серия 3: Физико-математические науки и информатика. 2017. №. 9 (200). С. 122-125.
2. Кугушева Д.С. Проектирование сложного программного обеспечения с использованием микросервисной архитектуры //Инновации и инвестиции. 2020. №. 5. С. 188-190.
3. Джалалов М.Э. Применение шаблонов проектирования для управления API в микросервисной архитектуре //Экономика и качество систем связи. 2024. №. 1 (31). С. 128-136.
4. Абдирашитулы А. Микросервисная архитектура в NLP платформах: проектирование и масштабируемость //Вестник науки. 2026. Т. 1. №. 5 (98). С. 609-619.
5. Городничев М.Г., Полонский Р.В. Оценка возможности использования микросервисной архитектуры при разработке пользовательских интерфейсов клиент-серверного программного обеспечения //Экономика и качество систем связи. 2020. №. 3 (17). С. 33-43.
6. Карпович М.Н. Особенности проектирования микросервисно-событийных архитектур для высоконагруженных распределенных систем обработки информации //Труды БГТУ. Серия 3: Физико-математические науки и информатика. 2023. №. 1 (266). С. 89-95.
7. Маличенко С.В. Проблемы перехода от монолитной к микросервисной архитектуре //Евразийский научный журнал. 2022. №. 5. С. 8-19.
8. Берайлик Г.Р. Проектирование информационных систем с помощью микросервисной архитектуры //Инновационные подходы к решению технико-экономических проблем. 2020. С. 187-191.