УДК 004.853

Анализ специфики и области применения архитектурных шаблонов в развернутых экосистемах

К. А. Терещенко, Р. В. Мальчева
ГОУВПО «Донецкий национальный технический университет»
E-mail: malcheva.raisa@yandex.ru

Аннотация
В статье рассмотрены основные шаблоны системной архитектуры. Сформулирована система оценки их применения в развёрнутых экосистемах. Проведена оценка применимости в рамках сформированной системы. Выбран наиболее применимый шаблон. Описаны дополнительные шаблоны для повышения соответствия определенным обязательным критериям шаблона. Сформулированы критерии оценки архитектур: безопасность, масштабируемость, оптимальность, интегрируемость, сложность и адаптивность, на основании которых сделан вывод, что наиболее применимой является микросервисная архитектура.

Введение

Как отмечают многие авторы, индустриальная фаза роста мировой экономики закончилась в 2018-2020 гг., и ее дальнейшее развитие будет осуществляться под все большим воздействием всеобщей цифровизации [1, 2]. И, если первоначально этот процесс затрагивал отдельные производственные, банковские, управленческие или иные процессы, то современное внедрение цифровых систем происходит повсеместно. Следует отметить, что эффективность их применения не всегда отвечает ожиданиям инвесторов.

Типичным представителем современных цифровых систем являются экосистемы, которые предоставляют крайне разнообразный набор услуг, включающих в себя широкий спектр связанных систем, погружающих клиента в цикл взаимодействующих продуктов. Примером может служить реализация продукта, предоставляющего функционал по поиску и покупке недвижимости, который будет интегрирован с банковскими продуктами для упрощенной процедуры выдачи ипотеки на выбранные клиентом объекты. Таких продуктов, может быть, масса и все они направлены на мотивацию клиента в использовании различных услуг. Поэтому актуальной проблемой является проектирование эффективной архитектуры экосистем с учетом таких ключевых требований как масштабируемость, открытость, неоднородность, безопасность, доступность, и производительность [3-6].

Целью данной статьи является анализ особенностей и области применения шаблонов системной архитектуры с точки зрения эффективности их применения в развернутых экосистемах.

Общая постановка проблемы

Формирование масштабной сети взаимно интегрированных продуктов и систем порождает ряд существенных трудностей, сопряженных в первую очередь с планировкой и реализацией системной архитектуры, которая должны обладать рядом обязательных свойств [7]:

Безопасность. Взаимодействующие системы обменивается критичной для клиента персональной информацией: персональными данными, платежными данными и т.д. Очевидно, что сохранность данных должна быть гарантирована, так как помимо репутационных, компания несет юридические потери.

Масштабируемость. В условиях постоянного роста интеграций между различными системами вследствие увеличения числа информационных потоков, а также потенциальным расширением их объемов, например, в связи с ростом числа клиентов, система должна иметь возможность кратно увеличивать свою производительность, функциональность и объем обрабатываемых данных без значительного снижения эффективности или производительности.

Надежность. Крупные связанные системы зачастую являются высоконагруженными, так как обмениваются большим объемом данных. Это вызывает необходимость формирования надежных отказоустойчивых систем. Также надежность является фактором обеспечения сохраннности данных, утеря которых несет существенные репутационные и юридические риски.

Оптимальность. Система должна быть сконфигурирована таким образом, чтобы обеспечить возможность осуществления функционала с максимальной производитель-

ностью и эффективностью в рамках заданных ограничений, связанных со значительными объемами передаваемой и обрабатываемой информации.

Интегрируемость. В связи со значительной развернутостью и необходимостью взаимодействовать с большим числом других узлов экосистемы, система должна иметь доступный интеграционный интерфейс или доступ к эффективным интеграционным технологиям обмена данными.

Сложность. Сложные системные шаблоны требуют существенных финансовых издержек для реализации и поддержки, что может стать препятствием для реализации. Кроме того поддержка и внедрение сложных системных шаблонов влечет за собой повышение рисков сбоев и ошибок.

Адаптивность. Система должна быть эластичной и иметь возможность к адаптации изменяющихся технических и юридических условий функционирования.

Подбор архитектурного фундамента для достижения данных свойств является комплексной задачей, которая требует выполнения как обзора существующих решений, так и реализации специфичных, направленных на устранение локальных противоречий с изменяемыми требованиями, разработок.

Анализ ключевых архитектурных шаблонов

Рассмотрим основные шаблоны архитектуры, которые сейчас используются в различных системах и приложениях [8, 9].

Слоистая архитектура. Согласно модели слоистой архитектуры, компоненты системы структурированы в горизонтальные слои, каждый из которых отвечает за свою определенную функцию (рисунок 1). Хотя количество и типы слоев не фиксированы, обычно слоистая архитектура включает четыре основных слоя:

- слой представления;
- слой бизнес-логики;
- слой доступа к данным;
- слой абстракции баз данных.

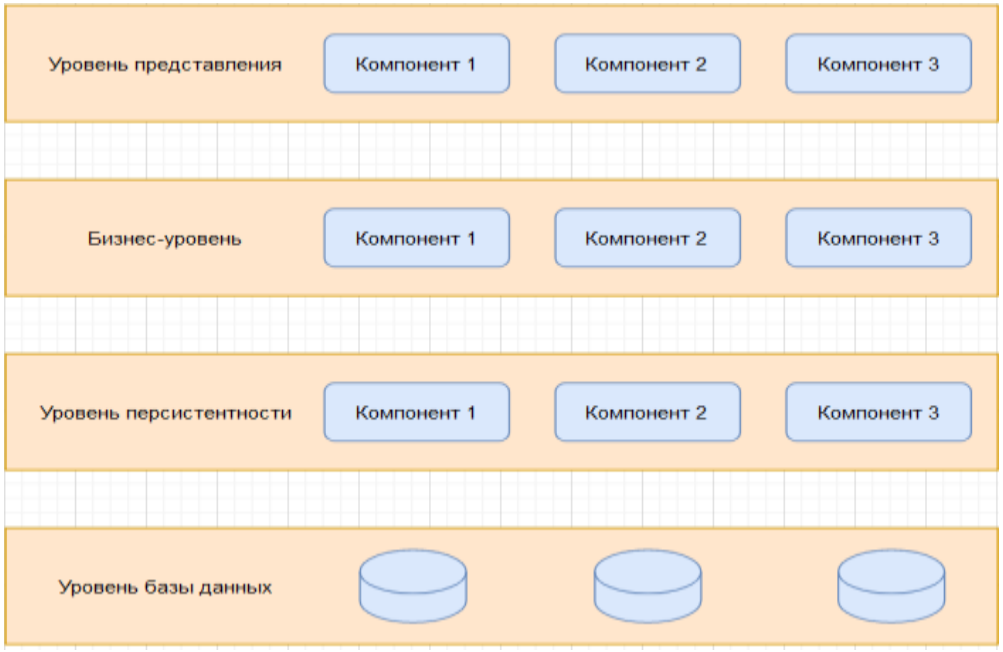

Рисунок 1 – Слоистая архитектура

Иногда слой бизнес-логики и слой доступа к данным объединяются в один, особенно если логика доступа к данным встроена в компоненты бизнес-логики. Таким образом, количество слоев может варьироваться в зависимости от размера и сложности приложения - от трех слоев для небольших проектов до пяти и более для крупных и сложных бизнес-приложений.

Каналы и фильтры. В данной архитектуре (рисунок 2) существует 4 ключевые компоненты:

- генератор (источник) - начальная точка процесса;
- преобразователь (сопоставление) - выполняет преобразование всех или некоторых данных;
- испытатель (редуцирование) - проверяет один или несколько критериев;
- потребитель (приемник) - конечная точка.

Фильтры взаимодействуют друг с другом через коммуникационные каналы.

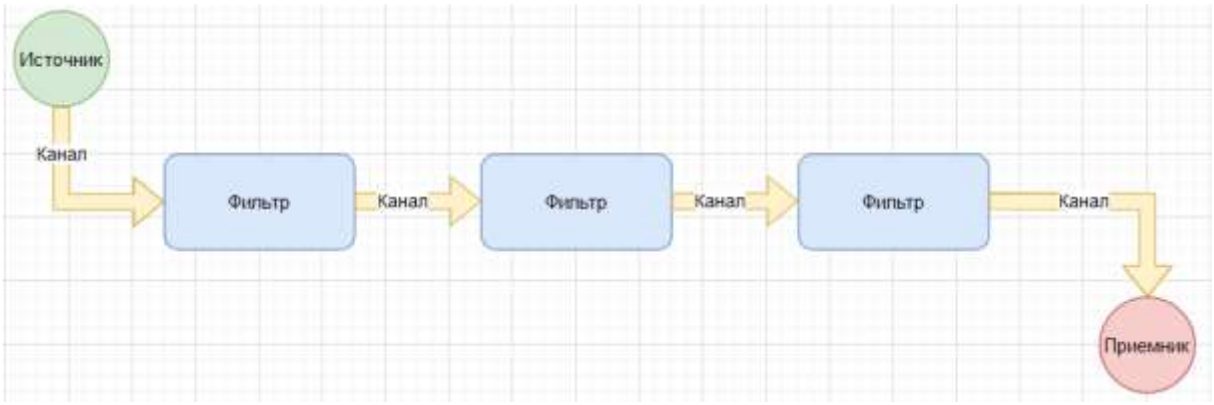

Рисунок 2 – Каналы и фильтры

Основная концепция заключается в том, что каждый канал типа "точка-точка" для повышения производительности и является однонаправленным, принимая данные от одного источника и направляя их к приемнику.

Микроядерная архитектура. Архитектура микроядра состоит из двух видов компонентов (рисунок 3): основной системы и модулей-плагинов. Функциональная часть приложения разделена между независимыми модулями-плагинами и основной системой, что позволяет обеспечить гибкость, расширяемость и изоляцию функций приложения и пользовательской логики обработки данных.

Рисунок 3 – Микроядерная архитектура

Модульный монолит. Это концепция, при которой в рамках одной системы кодовая база разделяется на модули, часто реализующие различные области функциональности. Между объектами функционального модуля формируется «сильная» связанность (объекты модуля созависимы друг от друга и изменения одного зачастую влечет цепочку изменений в других). Между самими же модулями слабый тип связанности, что позволяет трансформировать конкретный модуль без необходимости трансформировать остальные (Рисунок 4.)

Микросервисная архитектура. Архитектура микросервисов (рисунок 5) состоит из нескольких отдельных сервисов, каждый из которых работает в собственном процессе и выполняет единственную функцию. Каждый сервис взаимодействует с другими сервисами через API. Это позволяет разработчикам создавать и обновлять приложения независимо друг от друга, улучшая масштабируемость и гибкость разработки.

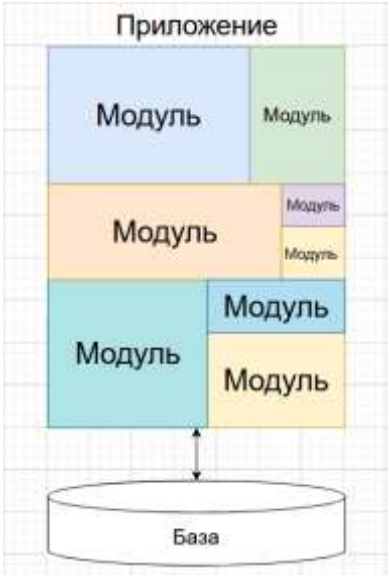

Рисунок 4 - Модульный монолит

Сравнение шаблонов дано в таблице.

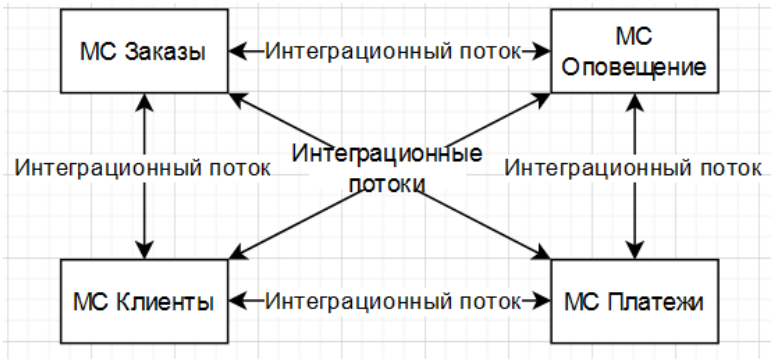

Рисунок 5 – Микросервисная архитектура

Таблица - Сравнение шаблонов системной архитектуры

Название шаблона	Значимые Метрики	Соответствие метрики	Обоснование
Слоистая архитектура	1. Безопасность	Средняя	Архитектура позволяет изолировать чувствительные данные и функциональности в отдельных слоях, что уменьшает риск их компрометации, тем не менее в рамках реализации слоенной архитектуры могут возникнуть дополнительные точки для атаки злоумышленников, так как каждый слой представляет собой потенциальную уязвимую точку.
	2. Масштабируемость	Низкая	Из-за тесной связанности и монолитности данного Шаблона системы, созданные с его использованием обычно затруднительно масштабировать. Слоистую архитектуру можно масштабировать, разделяя слои на отдельные физические экземпляры или копируя всё приложение на несколько узлов. Однако, в целом, излишняя детализация такой архитектуры приводит к высоким затратам на масштабирование.
	3. Надежность	Средняя	С увеличением числа слоев возрастает сложность конфигурации и взаимодействия между ними, что может привести к ошибкам и уязвимостям, тем не менее упрощенная структура шаблона на малых по объему приложениях минимизирует количество излишних интеграционных потоков и потенциальных точек отказа.
	4. Оптимальность	Низкая	Несмотря на то, что некоторые слоистые архитектуры могут хорошо выполнять свою работу, этот Шаблон не эффективен для высокопроизводительных приложений. Это связано с необходимостью проходить через несколько слоёв архитектуры для выполнения бизнес-запроса.
	5. Интегрируемость	Низкая	Шаблон позволяет выделить отдельный слой под интеграционное взаимодействие, однако донесение запроса до последующих слоев может занять значительный объем времени, так же реализации оркестрации распределение бизнес-запросов не предусматривается в данном шаблоне, что может вызвать трудности в реализации.
	6. Простота	Высокая	Этот шаблон отличается легкостью в разработке, в основном потому, что он хорошо известен и не слишком сложен в реализации. Большинство компаний разрабатывают приложения, опираясь на разделение компетенций по уровням (например, представление, бизнес-логика, база данных), поэтому данный шаблон естественным образом становится предпочтительным выбором для создания многих типов бизнес-приложений.
	7. Адаптивность	Средняя	Хотя возможно изолировать изменения с использованием концепции изолированных слоев, внесение изменений в структуре архитектуры затруднено из-за того, что реализации монолитны и компоненты в определенной степени сопряжены между собой.

Продолжение таблицы

Каналы и фильтры	1. Безопасность	Средняя	Система фильтров позволяет устанавливать проверки безопасности, фильтрующие данные соответствующим образом, тем не менее, каналы, как интеграционные потоки, представляют собой дополнительные узлы риска. Также система нуждается в комплексном конфигурировании взаимодействия: настройка фильтров, установка необходимого типа взаимодействия с поставщиком данных и т.д., что повышает риск допущения уязвимостей.
	2. Масштабируемость	Низкая	Шаблон используется для ограниченного ряда решений. В основном для формирования систем, организующих обмен данными между другими системами. Сама структура решения не подразумевает значительного диапазона вариаций масштабирования.
	3. Надежность	Средняя	Шаблон обладает надежностью в контексте правильной его конфигурации и использования подходящей инфраструктуры. Система каналов позволяет устанавливать доступный мониторинг на способность канала отслеживать уязвимости, потерю данных и проводить необходимый аудит. Тем не менее, сложная система фильтрации и установленных на ней проверок может приводить к отсеиванию критически важной информации, а также последующей ее потери, Поэтому надежность шаблона сильно зависит от реализации.
	4. Оптимальность	Высокая	В целом, шаблон достигает высокой оптимальности и производительности благодаря асинхронным возможностям (передача сообщения источником без необходимости получения ответа от потребителя), простоте реализации и доступности решений в рамках современной инфраструктуры.
	5. Интегрируемость	Высокая	Шаблон используется как система регулирования интеграции между системами, Поэтому подразумевает использование удобного для интеграции интерфейса.
	6. Сложность	Средняя	Разработка может быть трудной из-за асинхронной структуры шаблона, а также из-за необходимости формулирования контрактов и предусмотрительного обработчика ошибок в коде для сбойных обработчиков событий и брокеров. Шаблоны сильно зависят от конфигурирования, потому требуют тщательного подхода к реализации. Тем не менее простейший концепт шаблона можно создать без значительных ресурсных затрат.
	7. Адаптивность	Низкая	Шаблон используется для ограниченного ряда решений. В основном для формирования систем, организующих обмен данными между другими системами. Решение не может быть адаптировано для реализации значительного ряда концепций.
Микроядерная архитектура	1. Безопасность	Высокая	Обеспечивает высокий уровень безопасности, поскольку все драйверы устройств работают в отдельном от ядра системы адресном пространстве. Это снижает риск повреждения и взлома системы.
	2. Масштабируемость	Низкая	Большинство реализаций микроядерной архитектуры предназначены для использования в продуктовых приложениях и обычно имеют более компактный размер. Они представляются как единый модуль развертывания, что ограничивает их способность к масштабированию. В зависимости от того, как реализованы подключаемые модули, время от времени масштабируемость может быть обеспечена на уровне этих модулей.

Продолжение таблицы

	3. Надежность	Средняя	Основной функционал реализован в ядре системы, а подключаемые плагины представляют собой отдельные плагины, содержащие опциальные возможности, поэтому нарушение работы плагинов не несет в себе критической угрозы для системы. Тем не менее, нарушения в работе непосредственно ядра, могут привести к значительным проблемам и полной потере части критически важной информации.
	4. Оптимальность	Низкая	Микроядерная архитектура требует больше времени для выполнения операций, так как каждый запрос к ядру должен проходить через слой абстракции. Это зачастую приводит к снижению производительности системы.
	5. Интегрируемость	Низкая	Каждая интеграция дополнительной опциональности происходит только в определенном, ограниченном формате «ядро -> плагин», что значительно затрудняет межсистемную интеграцию.
	6. Сложность	Высокая	Для успешной реализации архитектуры необходимо тщательное планирование и контроль за контрактами, что делает этот процесс довольно сложным. Учет версий контрактов, обновление реестров модулей, детализация модулей и широкий выбор методов их подключения являются основными факторами, которые добавляют сложности при внедрении данного подхода
	7. Адаптивность	Высокая	Изменения могут быть изолированы и оперативно внедрены за счет слабо связанных модулей. В общем, система в большинстве микроядерных архитектур быстро стабилизируется и требует лишь минимальных изменений со временем.
Модульный монолит	1. Безопасность	Средняя	В рамках реализации модульного монолита есть возможность изолировать критически важные данные в различных модулях, однако реализация модульного монолита зачастую не подразумевает значительной сепарации данных, в результате, зачастую, компрометация данных БД приводит к потери значимого объема критически важных данных.
	2. Масштабируемость	Средняя	Дополнительная функциональность реализуется через внедрение дополнительных модулей, что в данной архитектуре является шаблонной реализацией. Тем не менее, значительное увеличение числа модулей усложняет процесс стратегирования связей по ходу масштабирования системы, а также усложняет процесс развертывания системы на сервисе.
	3. Надежность	Средняя	Основной функционал реализован в рамках монолитной структуры, что, с одной стороны, снижает число потенциальных точек отказа (интеграционных потоков), а, с другой стороны, из-за незначительной степени сепарации данных нарушение процессов функциониро-вания БД может привести к потери значимого объема критически важных данных.
	4. Оптимальность	Средняя	Взаимодействие между модулями не обременено комплексными потоками данными, что позволяет реализовывать функциональность с высокой производительностью. Однако, при значительном расширении опциональности приложения, количество связей между модулями будет расти, что может привести к падению производительности
	5. Интегрируемость	Средняя	Модульность позволяет создавать отдельные точки для интеграции, однако последующее распределение данных может быть затруднено из-за слабой связанности определенных компонент.
	6. Сложность	Низкая	Данная архитектура является одним из наиболее популярных и естественных способов реализации монолитной архитектуры потому существует значительное число как технических, так и стратегических методологий реализации.

Продолжение таблицы

	7. Адаптивность	Высокая	Новая функциональность внедряется путем создания дополнительных модулей и сопряжения их с необходимыми компонентами, что в контексте обсуждаемой концепции является относительно простым способом.
Микросервисная архитектура	1. Безопасность	Средняя	Микросервисная архитектура подразумевает сепарацию информации по отдельным БД для сервисов, что снижает риск компрометации, тем не менее, сервисы взаимосвязаны между собой интеграционными потоками, каждый из которых может стать точкой кражи информации.
	2. Масштабируемость	Высокая	Приложение разбито на отдельные компоненты, которые могут быть масштабированы по отдельности. Это позволяет точно настроить масштабирование всего приложения в зависимости от его потребностей.
	3. Надежность	Средняя	Микросервисная архитектура подразумевает сепарацию информации по отдельным БД для сервисов, что снижает риск потери данных. Тем не менее, сервисы взаимосвязаны между собой интеграционными потоками, каждый из которых может стать точкой кражи отказа.
	4. Оптимальность	Низкая	В связи с необходимостью формировать дополнительные точки нагрузки в виде интеграционных потоков, связывающих микросервисы, подход не является наиболее оптимальным решением с точки зрения производительности.
	5. Интегрируемость	Высокая	Помимо возможности выделить отдельный сервис для интеграции со сторонними системами, который будет оркестровать данные от других систем, каждый микросервис может иметь свой собственный интеграционный интерфейс, что позволяет выстраивать необходимые потоки данных направленые в требуемые узлы.
	6. Сложность	Низкая	Благодаря выделению функциональных возможностей в отдельные сервис-компоненты, разработка становится проще, поскольку объем работы уменьшается, и каждый компонент изолирован от других. Это уменьшает риск внесения изменений в один компонент, которые могут повлиять на другие, и, следовательно, уменьшает необходимость согласования между разработчиками и командами разработки.
	7. Адаптивность	Высокая	Благодаря концепции отдельных развернутых модулей, изменения вносятся в изолированные сервис-компоненты, что ускоряет процесс развертывания. Приложения, разработанные с учетом этого подхода, имеют низкую взаимосвязь друг с другом, что упрощает внесение изменений.

Анализ таблицы показывает, что наиболее эффективным шаблоном реализации архитектуры является микросервисная архитектура по совокупности признаков, важнейшими из которых являются масштабируемость, интегрируемость и адаптивность.

Следует отметить, что данный шаблон также обладает рядом недостатков и проблем, часть из которых могут привести к критическим инцидентам в масштабных экосистемах, например, относительно низкая оптимальность, средняя надежность и безопасность. В результате экосистемы устанавливают дополнительные условия реализации в той или иной конфигурации, призванные решить возникающие проблемы. Также компании накладывают дополнительные условия реализации, применяя, а иногда и создавая, топические системные шаблоны.

Одним из таких шаблонов является **Circuit Breaker («Автоматический выключатель»)** приведенный на рисунке 6 [10]. При взаимодействии микросервисов возможны ситуации, когда один из сервисов перестает функционировать. Для борьбы с временными сбоями, вызванными, например, медленным сетевым соединением, используются повторные вызовы. Однако при серьезных сбоях, приводящих к полному отказу сервиса, повторные вызовы будут просто расходовать ресурсы.

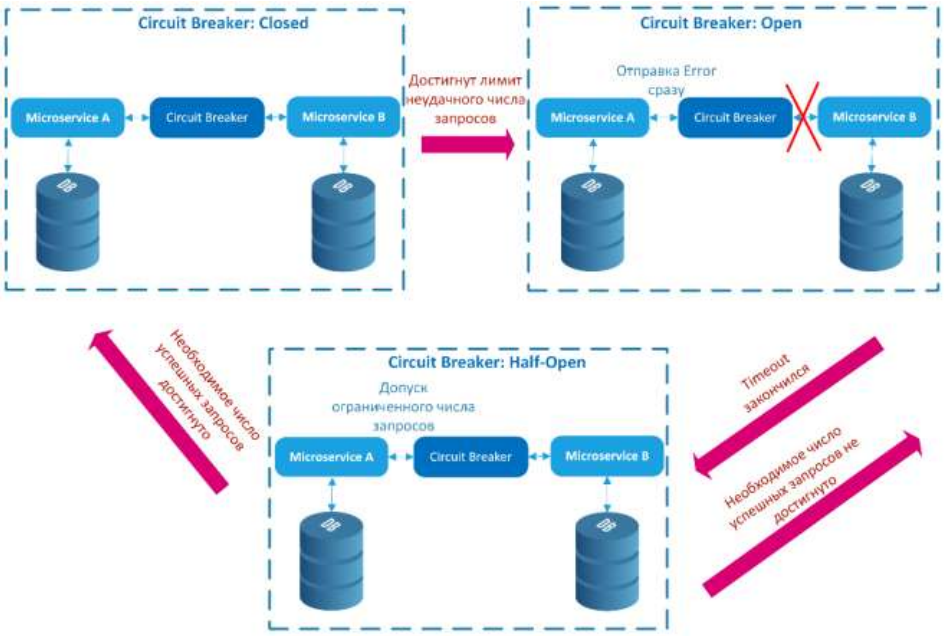

Рисунок 6 – Автоматический выключатель

Чтобы уменьшить вероятность каскадных сбоев в таких случаях, рекомендуется использовать данный шаблон. При его применении микросервис обращается к другому микросервису через прокси-сервер, который контролирует количество сбоев и принимает решение о том, следует ли продолжать отправку запросов или вернуть ошибку. Прокси-сервер может находиться в трех состояниях:

Закрытое. Происходит обмен данными и подсчет сбоев. В случае превышения порога сбоев за определенный период времени, прокси-сервер переключается в режим "Открытое".

Открытое. Все запросы сразу возвращаются с ошибкой. После истечения заданного временного интервала прокси-сервер переходит в режим "Полуоткрытое".

Полуоткрытое. Прокси-сервер ограничивает количество запросов и подсчитывает успешные попытки. Если достигнуто необходимое количество, прокси-сервер переключается в режим "Закрытое", иначе возвращается в режим "Открытое".

Использование шаблона "Circuit Breaker" помогает повысить отказоустойчивость системы и предотвратить распространение сбоев, но требует тщательной настройки и мониторинга.

MDM – повышение безопасности. Как было отмечено, шаблон микросервисной архитектуры обладает средними показателями безопасности. В значительной степени это связано с тем, что зачастую крупные системы для оптимизации работоспособности и повышения отказоустойчивости, применяют шаблон, в рамках которого для каждого микросервиса используется своя БД. Это сильно повышает риск компрометации данных, так как каждая дополнительная база является критическим узлом. Для разрешения данной проблемы периодически применяется шаблон, основанный на подходе MDM (master data management).

Этот шаблон характеризуется сохранением критически значимой информации исключительно в мастер-системах без хранения их в качестве характеристик других сущностей.

Рассмотрим упрощенный абстрактный пример (рисунок 7). Предположим, что есть микросервис, отвечающий за работу с клиентом. Информация о банковских картах клиента является одной из его характеристик, она необходима для формирования отчетности, сохранения чеков, повышения удобства пользователя (чтобы ему не приходилось при каждой покупке заново вводить данные карты) и т.д. Однако, также эти данные являются критически важными и хранение их во всех базах, которые с этой информацией взаимодействуют, приведет к значительному повышению сопутствующих рисков.

В результате сохраняется не конкретная информация о банковской карте клиента, а ссылка на эту информацию в мастер-системе, условный идентификатор данных в базе другого микросервиса. Теперь, когда сервису, отвечающему за работу с клиентами, требуется совершать действия с данными карт, он обращается по сохраненному Id в мастер-систему, и, после проверки на наличие необходимых прав, система передает данные, которые применяются для конкретной операции.

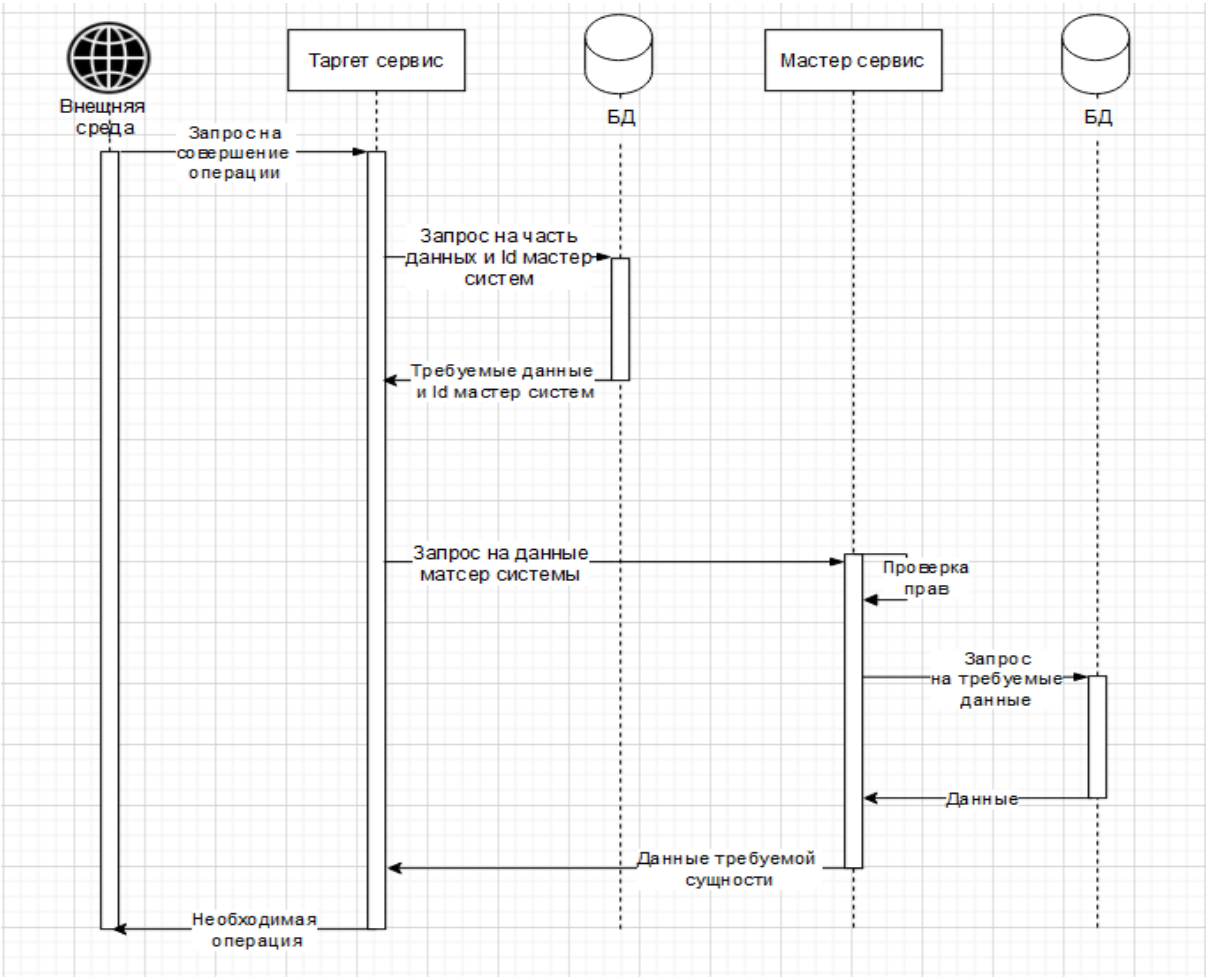

Рисунок 7 – Sequence-диаграмма процесса взаимодействия с мастер-системой

Шаблон «API-шлюз» (API Gateway) – повышение оптимальности *(рисунок 8) [11]*

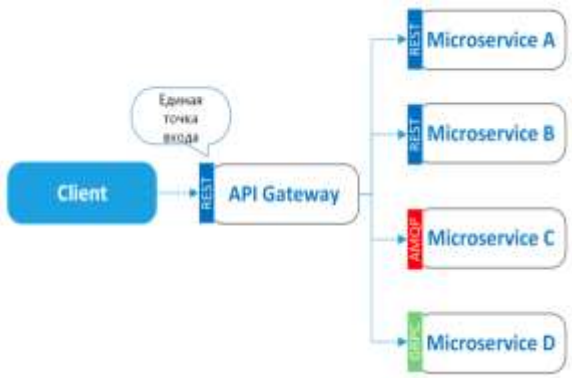

Рисунок 8 - API Gateway

В зависимости от цели использования имеет следующие модификации:

- Gateway Routing. Шлюз работает как прокси-сервер, который направляет запросы клиента к соответствующему сервису.
- Gateway Aggregation. Шлюз используется для разделения клиентского запроса на несколько микросервисов и объединения ответов, возвращаемых клиенту.

Gateway Offloading. Шлюз выполняет общие для сервисов задачи, такие как аутентификация, авторизация, настройка SSL, ведение журналов и другие.

Как было отмечено шаблон микросервисной архитектуры обладает низкими показателями оптимальности. Более того применение различных дополнительных шаблонов, как например MDM может еще сильнее снизить оптимальность работы микросервисов.

Наиболее доступный метод взаимодействия между микросервисами — прямое обращение между ними. В крупных экосистемах с большим числом микросервисов число обращений может быть значительным, более того нередкими становятся ситуации формирования целых цепочек последовательных вызовов от сервиса к сервису значительно повышающих время обработки задач из внешней среды и увеличивающих нагрузку на систему (рисунок 9).

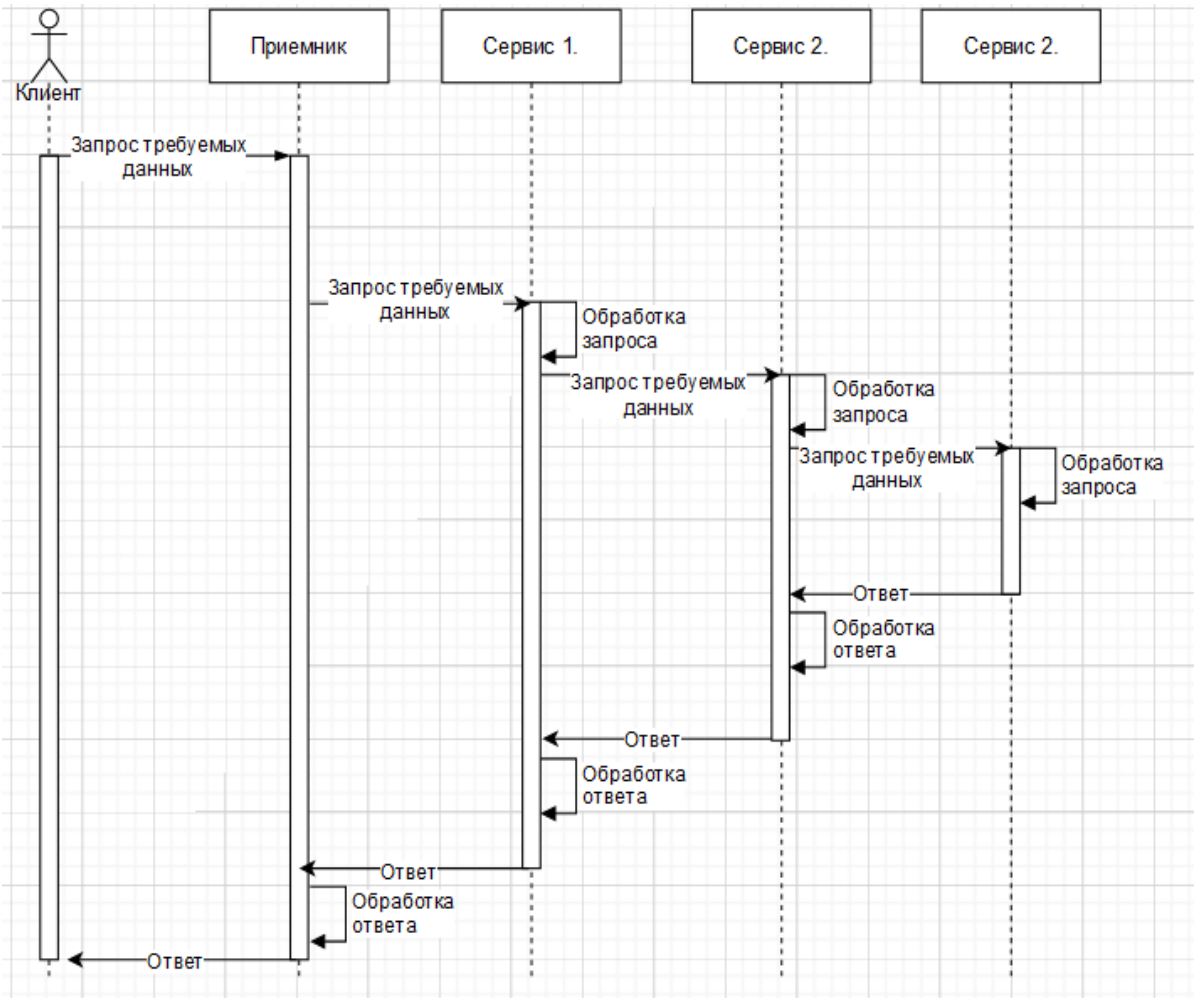

Рисунок 9 – Sequence-диаграмма цепочек последовательных вызовов

Выводы

В ходе рассмотрения популярных шаблонов системной архитектуры с точки зрения их применения их в развёрнутых экосистемах:

- сформулированы критерии оценки безопасности, масштабируемости, оптимальности, интегрируемости, сложности и адаптивности систем;
- сделан вывод, что наиболее применимой является микросервисная архитектура.

Также выявлены проблемы, возникающие при применении микросервисной архитектуры с точки зрения удовлетворения надежности, безопасности и оптимальности. Предложены шаблоны, призванные решить данные проблемы. Рассмотрены их особенности.

Литература

1. Мальчева, Р. В. Компьютерные технологии – основа цифровой экономики // Бизнес-инжиниринг сложных систем: модели, технологии, инновации. Сборник материалов III международной научно-практической конференции. – Донецк: ДОННТУ, 2018. - С. 102-105.

2. Паньшин, Б. Цифровая экономика: особенности и тенденции развития [Электронный ресурс]. – Режим доступа: https://cyberleninka.ru/article/n/tsifrovaya-ekonomika-osobennosti-i-tendentsii-razvitiya

3. Мальчева, Р. В. Проектирование архитек-туры системы электронных денег / Р. В. Мальчева, К. А. Терещенко // Информатика и кибернетика. – Донецк: ДонНТУ, 2023. - № 1(31). – С. 23-29.

4. Терещенко, К. А. Проектирование распределенной архитектуры компьютерной сети банка / К. А. Терещенко, Р. В. Мальчева // Информационное пространство Донбасса: проблемы и перспективы : материалы V Респ. С междунар. Участием науч.-практ. Конф., 27 окт. 2022 г. – Донецк : ГОУ ВПО «ДонНУЭТ», 2022. – С. 131 -134.

5. Терещенко, К. А. Анализ особенностей функционирования и развития объектов и процес-сов компьютерной сети банка / К. А. Терещенко, Р. В. Мальчева // Информатика, управляющие системы, математическое и компьютерное моделирование (ИУСМКМ-2022): Материалы XIII Международной научно-

технической конференции в рамках VIII Международного Научного форума Донецкой Народной Республики. Донецк, 2022. – Донецк: ДонНТУ, 2022. – С.392-295.

6. Service-oriented architecture [Electronic resource] – URL: htpps://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/service-oriented-architecture

7. Миндалёв, И. В. Электронный учебно-методический комплекс для направления 09.03.03 (230700.62) «Прикладная информатика» : Раздел 1.3 Требования [Электронный ресурс]. – Режим доступа: http://enisey.name/umk/upr/ch01s03.html

8. Raj, P. Architectural Patterns: Uncover essential patterns in the most indispensable realm of enterprise architecture [Электронный ресурс] / P. Raj, A. Raman, H. Subramanian. – Packt Publishing - ebooks Account, 2017. – 468 p. Режим доступа: - htpps://ru.z-library.se/book/3493736/28a234/architectural-patterns-uncover-essential-patterns-in-the-most-indispensable-realm-of-enterprise-arc.html?dsource=recommend

9. Richards, M. Software Architecture Patterns: Understanding Common Architectural Styles and When to Use Them, 2nd Edition [Электронный ресурс]. - O'Reilly Media, Inc., 2022. – 92 p. — ISBN-13: 978-1-098-13427-3. - Режим доступа: https://vtome.ru/knigi/programming/595543-software-architecture-patterns-2nd-edition.html

10. Circuit Breaker pattern [Электронный ресурс]. – Режим доступа: - https://learn.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker

11. Использование шлюзов API в микрослужбах [Электронный ресурс]. – Режим доступа: - https://learn.microsoft.com/ru-ru/azure/architecture/microservices/design/gateway

Терещенко К. А., Мальчева Р. В. Анализ специфики и области применения архитектурных шаблонов в развернутых экосистемах. *В статье рассмотрены основные шаблоны системной архитектуры. Сформулирована система оценки их применения в развёрнутых экосистемах. Проведена оценка применимости в рамках сформированной системы. Выбран наиболее применимый шаблон. Описаны дополнительные Шаблоны для повышения соответствия определенным обязательным критериям шаблона. Сформулированы критерии оценки архитектур: безопасность, масштабируемость, оптимальность, интегрируемость, сложность и адаптивность, на основании которых сделан вывод, что наиболее применимой является микросервисная архитектура.*

Ключевые слова: *экоситема, шаблон, архитектура, критерии, система оценок*

Tereshchenko K. A., Malcheva R. V. Analysis of the specifics and scope of architectural patterns in deployed ecosystems. *The article discusses the main patterns of the system architecture. A system for evaluating their application in deployed ecosystems has been formulated. The assessment of applicability within the framework of the formed system is carried out. The most applicable template has been selected. Additional patterns are described to enhance compliance with certain mandatory template criteria. The criteria for evaluating architectures are formulated: security, scalability, optimality, integrability, complexity and adaptability, on the basis of which it is concluded that the microservice architecture is the most applicable.*

Key words: *ecosystem, pattern, architecture, criteria, evaluation system*

Статья поступила в редакцию 10.12.2023
Рекомендована к публикации профессором Зори С. А.