

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

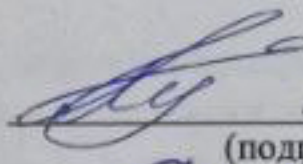
Направление подготовки: 09.04.01 «Информатика и вычислительная техника»
Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

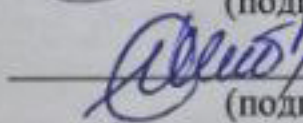
на тему

АЛГОРИТМЫ И ПРОГРАММНЫЕ СРЕДСТВА
УПРАВЛЕНИЯ МЕТАДААННЫМИ В ДИСКРЕТНО-
СОБЫТИЙНЫХ ДИНАМИЧЕСКИХ
ИНФОРМАЦИОННО УПРАВЛЯЮЩИХ СИСТЕМАХ

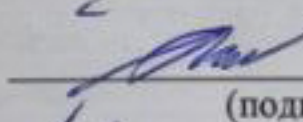
Студент

 30.05.24 Гусаров Александр Сергеевич
(подпись, дата) (ФИО полностью)

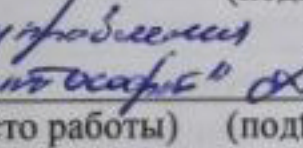
Руководитель

 30.05.24 С.В. Шибанов
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

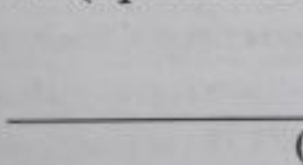
 30.05.24 А.С. Бычков
(подпись, дата) (фамилия, инициалы)

Рецензент

исполнительное управление
ИТТИ «КриптоСарис»  30.05.24 А.Ю. Афонин
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 27.05.2024 № 11)

Заведующий кафедрой

 М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой

отлично (протокол заседания ГЭК от 6.06.24 № 9)

Секретарь ГЭК

 Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 28 » марта 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

1. Студент Гусаров Александр Сергеевич гр. 22ВВИм2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Алгоритмы и программные средства управления метаданными в
дискретно-событийных динамических информационно управляющих системах

Тема утверждена приказом ПГУ № 290/о от " 28 " марта 2024 г.

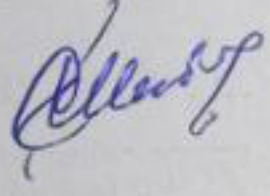
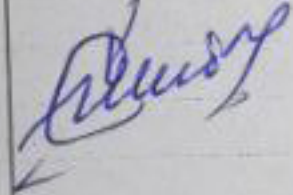
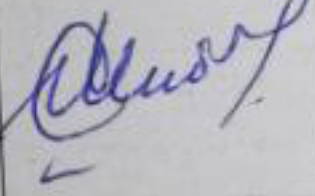
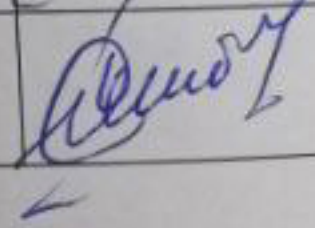
3. Руководитель работы к.т.н., доцент кафедры «МО и ПЭВМ» Шибанов С.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

- 1) *Назначение: алгоритмы и программные средства управления метаданными*
- 2) *Среда выполнения: Linux. Языковые средства разработки: TypeScript. Инструментальные средства: VS Code, PostgreSQL, Redis, Prometheus, Grafana, DBeaver, Docker, Draw.io. Технологии разработки: ООП, сервисно-ориентированных подход.*
- 3) *Основные задачи работы:*
 - а) *разработка обобщенной метамодели ДС ИУС в рамках микросервисной архитектуры с применением потоковой обработки событий и механизма активных правил;*
 - б) *разработка единой архитектуры метаданных и модели распределенного репозитория метаданных в рамках микросервисной архитектуры ДС ИУС на основе обобщенной метамодели;*
 - в) *разработка алгоритмов и программных средства управления метаданными, включая конструирование, анализ, тиражирование и синхронизацию метаданных в компонентах микросервисной архитектуры ДС ИУС;*
 - г) *экспериментальная оценка и анализ предлагаемых решений;*

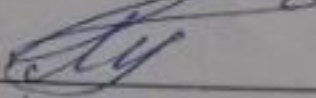
5. Перечень подлежащих разработке вопросов:
- 1) Анализ применения метаданных в реагирующих дискретно-событийных динамических информационно управляющих системах;
 - 2) Разработка Архитектуры, моделей и алгоритмов управления метаданными в реагирующей ИУС потоковой обработки событий и исполнения активных правил в рамках единой метамодели;
 - 3) Разработка программных средств управления метаданными в составе сервиса потоковой обработки событий и исполнения активных правила;
 - 4) Экспериментальная оценка разработанных моделей, алгоритмов и программных средств управления метаданными;
 - 5) Оформление пояснительной записки.

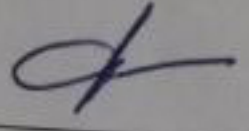
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ применения метаданных в реагирующих дискретно-событийных динамических информационно управляющих системах	15%	28.03.2024 – 06.04.2024	
2	Разработка Архитектуры, моделей и алгоритмов управления метаданными в реагирующей ИУС потоковой обработки событий и исполнения активных правил в рамках единой метамодели	10%	07.04.2024 – 13.04.2024	
3	Разработка программных средств управления метаданными в составе сервиса потоковой обработки событий и исполнения активных правила	50%	14.04.2024 – 16.05.2024	
4	Экспериментальная оценка разработанных моделей, алгоритмов и программных средств управления метаданными	10%	17.05.2024 – 23.05.2024	
5	Оформление пояснительной записки	15%	24.05.2024 – 31.05.2024	

Дата выдачи задания «28» марта 2024 г.

Руководитель магистерской диссертации  28.03.24 Шибанов С.В.
(подпись, дата)

Задание к исполнению принял магистрант  28.03.24 Гусаров А.С.
(подпись, дата)

Работу к защите допустить
Декан факультета  04.06.2024 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Пояснительная записка содержит 163 с., 46 рис., 6 табл., 33 источн., 4 прил.

**АЛГОРИТМЫ И ПРОГРАММНЫЕ СРЕДСТВА УПРАВЛЕНИЕ
МЕТАДААННЫМИ, ДИСКРЕТНО-СОБЫТИЙНЫЕ ИНФОРМАЦИОННО
УПРАВЛЯЮЩИЕ СИСТЕМЫ, ЕДИНАЯ МЕТАМОДЕЛЬ.**

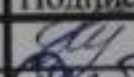
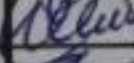
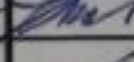
Объект исследования – процессы управления метаданными в дискретно-событийных информационно управляющих системах в рамках микросервисной архитектуры с применением потоковой обработки событий и механизма активных правил.

Цель работы – исследование и разработка моделей, алгоритмов и программных средств управления метаданными в дискретно-событийных информационно управляющих системах, построенных на базе микросервисной программной платформы потоковой обработки и исполнения активных правил.

Технологии разработки – разработка серверных программных средств проводилась на языке программирования TypeScript с использованием фреймворка Nest и ORM Sequalize, программное обеспечение разворачивалось при помощи средств Docker, для создания репозиторов использовались PostgreSQL и REDIS, обмен данными с микросервисами производился при помощи средств REDIS PUB/SUB и gRPC. Для сбора и анализа метрических показателей использовались Prometheus и Grafana.

Среда исполнения – Linux. IDE – VS Code.

В результате выполнения магистерской работы были разработаны модули, алгоритмы и программные средства управления метаданными, проведён эксперимент для подтверждения эффективности разработанных средств.

					ПГУ 1.090401.29.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Алгоритмы и программные средства управления метаданными в дискретно-событийных динамических информационно управляющих системах	Лит.	Лист	Листов
Разраб.		Гусаров А.С.		30.05.24				
Провер.		Шибанов С.В.		30.05.24			2	163
Н. Контр.		Бычков А.С.		30.05.24		ФВТ, каф. ВТ гр. 22ВВИм2		
								
Утверд.		Митрохин М.А.		30.05.24	Пояснительная записка			

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	6
1 Аспекты применения метаданных в реагирующих дискретно-событийных динамических информационно управляющих системах	13
1.1 Реагирующие дискретно-событийные динамические ИУС.....	13
1.1.1 Информационно-управляющие системы и их классификация	13
1.1.2 Понятие и роль обработки событий в информационно-управляющих системах.....	16
1.1.3 Свойства реагирующих дискретно-событийных динамических ИУС.....	18
1.2 Применение технологий потоковой обработки и активных правил для обнаружения событий и синтеза управляющих воздействий в реагирующих ИУС	19
1.2.1 Потоковая обработка событий.....	19
1.2.2 Концепции активных правил	23
1.2.3 Классификация активных правил.....	25
1.3 Применение метамоделирования для проектирования и реализации систем потоковой обработки событий с применением активных правил	25
1.3.1 Понятие и роль метаданных.....	26
1.3.2 Основы метамоделирования	27
1.3.3 Обзор подходов к разработке информационных систем с применением метамоделирования.....	29
1.4 Анализ требований к составу, архитектуре и процессам управления метаданными в реагирующих ИУС потоковой обработки событий и исполнения активных правил	33
1.5 Выводы по разделу 1	36
2 Архитектура, модели и алгоритмы управления метаданными в реагирующей ИУС потоковой обработки событий и исполнения активных правил в рамках единой метамодел.....	38

2.1 Единая метамодель реагирующей ИУС.....	38
2.1.1 Метамодель внешних систем.....	39
2.1.2 Метамодель событий	41
2.1.3 Метамодель активных правил.....	43
2.1.4 Отображение единой метамодели в модели данных	46
2.2 Алгоритмы первичной валидации метаданных при добавлении их в центральный репозиторий реагирующей ИУС	50
2.3 Распределенная архитектура метаданных реагирующей ИУС поточковой обработки событий и исполнения активных правил	53
2.4 Обоснование выбора подходов синхронизации метаданных	55
2.4.1 Модели согласованности в распределённой архитектуре	55
2.4.2 Управление версиями с использованием временных меток для синхронизации метаданных	57
2.4.3 Обработка конфликтов при тиражировании и синхронизации метаданных	58
2.5 Алгоритм тиражирования и синхронизации метаданных.....	60
2.5.1 Алгоритм тиражирования метаданных.....	60
2.5.2 Алгоритм синхронизации метаданных	65
2.6 Выводы по разделу 2.....	68
3 Программные средства управления метаданными в составе сервиса поточковой обработки событий и исполнения активных правила.....	70
3.1 Назначение, архитектура и функциональные возможности программных средств управления метаданными сервиса поточковой обработки событий и исполнения активных правил	70
3.2 Программные средства конструирования метаданных.....	75
3.3 Программные средства тиражирования и синхронизации распределенных метаданных	85
3.3.1 Экспериментальное обоснование выбора технологий для реализации алгоритма.....	85

3.3.2 Архитектура программных средств в процессе тиражирования и синхронизации	87
3.3.3 Применение Sidecar паттерна	92
3.4 Выводы по разделу 3	94
4 Экспериментальная оценка разработанных моделей, алгоритмов и программных средств управления метаданными	96
4.1 Планирование и постановка эксперимента	96
4.1.1 Сценарии и условия проведения эксперимента	96
4.1.2 Параметры экспериментальной оценки программных средств	99
4.1.3 Предметная область ИУС для экспериментальной оценки	100
4.2 Разработка, развертывание и настройка информационного и программного обеспечения эксперимента.....	101
4.3 Проведение эксперимента	109
4.4 Анализ и систематизация экспериментальных данных	114
4.5 Выводы по разделу 4.....	122
ЗАКЛЮЧЕНИЕ	123
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	124
ПРИЛОЖЕНИЕ А. Глоссарий	128
ПРИЛОЖЕНИЕ Б. Программный код для проведения эксперимента.....	130
ПРИЛОЖЕНИЕ В. Результаты апробации работы	140
ПРИЛОЖЕНИЕ Г. Презентация.....	148

ВВЕДЕНИЕ

Актуальность исследования. В соответствии с международным стандартом ISA-95 под информационно-управляющей системой (ИУС) понимается совокупность компонентов, собирающих, подготавливающих и предоставляющих данные, необходимые для безопасного и точного управления технологическим процессом, производством или предприятием в целом [1].

Быстрое развитие компьютерных и телекоммуникационных технологий, а также технологий автоматизированного управления вызвало появление и широкое распространение дискретно-событийных информационно-управляющих систем (ДС ИУС), которые характеризуются дискретностью пространства состояний и изменение состояний во времени производится асинхронными дискретными событиями [2-3].

В соответствии с классификацией Д. Харела информационно-управляющие системы можно разделить на трансформационные системы, которые преобразуют входные данные в выходные, и реагирующие системы, которые реагируют на изменения внешней среды [4]. В реагирующими системами входному событию, требующему отклика системы, ставится в соответствие заданное действие. При этом действия должны выполняться строго в том порядке, в котором происходят входные события. Если трансформационные системы запускаются по мере необходимости, то реагирующие системы, в большинстве своем, являются постоянно действующими.

Одним из подходов к построению событийно-ориентированных ИУС, непрерывно функционирующих в режиме 24/7 является применение технологии потоковой обработки событий (Event Stream Processing, ESP) [5]. Данная технология позволяет не только оперативно получать события от различных источников (датчиков, баз данных, приложений и др.), но и организовать их эффективную обработку и оперативно реагировать на возникновение событий. Технология ESP гарантирует получение, обогащение

событий метаданными и доставку событий адресатам, позволяет обрабатывать огромные объемы событий и реагировать на них в реальном масштабе времени, обнаруживать зависимости между событиями на основе заданных шаблонов, применяя технологию сложной обработки событий (Complex Event Processing, CEP).

Реагирование на поступившие или обнаруженные в потоках события удобным образом реализуется при помощи механизма активных правил (Active Rule, AR). Базовой концепцией представления активных правил является нотация «Событие-Условие-Действие» (Event-Condition-Action, ECA) [6], которая постоянно развивается и расширяется для эффективной разработки бизнес-логики управления.

Потоковая обработка событий и выработка реагирующих воздействий в ответ на возникновение событий реализуют асинхронный процесс, поэтому обоснованным является применение микросервисной модульной архитектуры для реализации программного обеспечения в ДС ИУС. Программная микросервисная платформа для построения ДС ИУС обеспечит быстрое развертывание, интеграцию и настройку ДС ИУС под конкретную предметную область и технологии сбора данных о событиях.

Low-code и no-code разработка в сочетании с микросервисной платформой значительно уменьшают время разработки и ввода в эксплуатацию ДС ИУС. Это позволяет инженерам, специалистам индустрии создавать и разворачивать как рыночно-ориентированные продукты, так и внутренние сервисы ИУС, не обладая большим опытом в программировании. Применение единой программной платформы, low-code и no-code средств разработки, ориентированных на специалистов предметной области, позволит сократить состав команды разработчиков, уменьшить расходы на оборудование, разработку, эксплуатацию и сопровождение программного обеспечения.

Достижение всех указанных преимуществ возможно на основе применения концепции мета-моделирования, широкого внедрения

метаданных на всех этапах разработки и на всех уровнях ИУС, автоматизации процессов управления метаданными [7]. При этом распределенная микросервисная архитектура ИУС порождает потенциальные проблемы с поддержкой актуальных значений метаданных в компонентах программных приложений. Применение классических сценариев обмена данными в распределенных приложениях для постоянно функционирующих высоконагруженных ДС ИУС ограничено, т.к. может потребовать останова и перезапуска работающих модулей после обновления метаданных. Таким образом, тема магистерской выпускной квалификационной работы представляется актуальной.

Объектом исследования являются процессы управления метаданными в ДС ИУС в рамках микросервисной архитектуры с применением потоковой обработки событий и механизма активных правил.

Предметом исследования являются модели, алгоритмы и программные средства конструирования, анализа и тиражирования метаданных в дискретно-событийных динамических информационно управляющих системах в рамках микросервисной архитектуры с применением потоковой обработки событий и механизма активных правил.

Целью диссертационной работы является исследование и разработка моделей, алгоритмов и программных средств управления метаданными в ДС ИУС, построенных на базе микросервисной программной платформы потоковой обработки и исполнения активных правил.

Для достижения поставленной цели определены следующие задачи:

- разработка обобщенной метамодели ДС ИУС, функционирующей на базе микросервисной архитектуры с применением технологии потоковой обработки событий и механизма активных правил,
- разработка единой архитектуры распределенного репозитория метаданных для ДС ИУС, функционирующей на базе микросервисной архитектуры с применением технологии потоковой обработки событий и механизма активных правил,

- разработка алгоритмов управления метаданными ДС ИУС на этапах жизненного цикла, включая конструирование, анализ, тиражирование и синхронизацию метаданных,

- разработка программных средств управления метаданными с применением low-code и no-code подходов,

- экспериментальная оценка и анализ предлагаемых решений.

Научная новизна, оригинальность идей в работе:

- предложена обобщенная метамодель ДС ИУС на основе теоретико-множественного подхода, описывающая расширенный набор классов событий и активных правил,

- разработана единая архитектура распределенного репозитория метаданных, ориентированная на микросервисную архитектуру ДС ИУС, применение потоковой обработки событий и активных правил,

- разработаны алгоритмы тиражирования и синхронизации метаданных в локальных репозиториях ДС ИУС, соответствующие требованиям скорости, надежности и восстановления состояния метаданных в случаях сбоев.

Методы проведенных исследований. Методы исследований основаны на положениях теории множеств и математической логики, теоретических положениях системного анализа, реляционной теории, методах проектирования реляционных и NoSQL баз данных, объектно-реляционного анализа и проектирования приложений.

Основные положения, выносимые на защиту:

- обобщенная метамодель ДС ИУС на основе теоретико-множественного подхода, описывающая расширенный набор классов событий и активных правил,

- единая архитектура распределенного репозитория метаданных для ДС ИУС, функционирующей на базе микросервисной архитектуры с применением технологии потоковой обработки событий и механизма активных правил,

- алгоритмы верификации метаданных на этапе конструирования,

– алгоритмы тиражирования и синхронизации метаданных в локальных репозиториях ДС ИУС, обеспечивающие высокую скорость, надежность и постоянную доступность приложений,

– программные средства управления метаданными в ДС ИУС.

Апробация работы. По тематике работы имеется 9 публикаций, 7 выступлений на научно-технических конференциях, участие в 3 конкурсах.

Основные положения и результаты работы докладывались на следующих конференциях:

– Открытая XXXII университетская научно-практическая конференции «Актуальные проблемы науки и образования» (Пенза, 2022 г.)

– XIV Международная научно-техническая конференция с элементами научной школы и конкурсом научно-исследовательских работ «Шляндинские чтения – 2022» (Пенза, 2022 г.),

– X Ежегодная всероссийская межвузовская научно-практическая конференция «Информационные технологии в науке и образовании. Проблемы и перспективы.» (Пенза, 2023 г.),

– Всероссийская научная конференция с международным участием «Информационные технологии в моделировании и управлении: подходы, методы, решения» (Тольятти, 2023 г.),

– IX Международная научно-практическая конференция (школа-семинар) молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук» (Тольятти, 2023 г.),

– Всероссийская научно-практическая конференция молодых ученых «Наука, инновации и технологии: от идей к внедрению» (Комсомольск-на-Амуре, 2023 г.),

– XX Международная научно-техническая конференция «Новые информационные технологии и системы» (Пенза, 2023 г.),

– XXXV Всероссийская с международным участием научно-техническая конференция, посвященной 80-летию Пензенского

государственного университета «Проблемы автоматизации и управления в технических системах» (Пенза, 2023 г.).

Участие в конкурсах научно-исследовательских работ:

– конкурс научно-исследовательских докладов в рамках XIV международной научно-технической конференции с элементами научной школы и конкурсом научно-исследовательских работ «Шляндинские чтения – 2022» (Диплом победителя I степени),

– конкурс научно-исследовательских докладов в рамках X ежегодной всероссийской межвузовской научно-практической конференция «Информационные технологии в науке и образовании. Проблемы и перспективы.» (Диплом победителя I степени в секции «Программное обеспечение автоматизированных систем и вычислительной техники»),

– конкурс научно-исследовательских докладов в рамках всероссийской научной конференции с международным участием «Информационные технологии в моделировании и управлении: подходы, методы, решения» (Сертификат участника),

– конкурс научно-исследовательских докладов в рамках IX международной научно-практической конференции молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук» (Сертификат участника),

– конкурс научно-исследовательских докладов в рамках всероссийской научно-практической конференции молодых ученых «Наука, инновации и технологии: от идей к внедрению» (Сертификат участника),

– конкурс научно-исследовательских докладов в рамках XX международной научно-технической конференции «Новые информационные технологии и системы» (Диплом победителя I степени в секции «Современные технологии передачи, хранения и обработки данных»),

– конкурс научно-исследовательских докладов в рамках XXXV всероссийской с международным участием научно-технической конференции, посвященной 80-летию Пензенского государственного университета

«Проблемы автоматизации и управления в технических системах»
(Сертификат участника),

– региональный конкурс научно-исследовательских работ обучающихся «Научный прорыв» (Сертификат участника),

– XXV Международный конкурс научно-исследовательских работ «Путь в науку – 2022» (Диплом победителя II степени),

– международный конкурс «Студент года – 2023», проводимого по инициативе проекта Intercllover (Лауреат III степени).

Структура и объем диссертации. Диссертация состоит из введения, 4 разделов, заключения, списка использованных источников, включающего 33 источника. Материал диссертации изложен на 163 листах, включает 46 рисунков, 6 таблиц и 4 приложения. Глоссарий представлен в приложении А.

1 Аспекты применения метаданных в реагирующих дискретно-событийных динамических информационно управляющих системах

Данный раздел работы посвящен рассмотрению специфики использования метаданных в реагирующих дискретно-событийных динамических информационно управляющих системах. Рассмотрены различные аспекты подобного рода систем с упором на микросервисную организацию и на потоковую обработку событий с последующим использованием активных правил. Приведены подходы метамоделирования для проектирования подобного рода систем. Предложены требования к составу, архитектуре и процессам управления метаданными.

1.1 Реагирующие дискретно-событийные динамические ИУС

Информационно-управляющие системы (ИУС) представляют собой организованную интеграцию аппаратных и программных средств, данных и процессов, ориентированных на управление информационными технологиями для обеспечения эффективного и действенного принятия решений. Реагирующие ИУС должны автоматически выполнять управляющие воздействия над контролируемыми внутренними и внешними элементами.

1.1.1 Информационно-управляющие системы и их классификация

ИУС реализуют свои процессы управления на основе обработки информации. Подобные системы могут использоваться для координации, контроля, анализа и визуализации данных. Благодаря анализу, компании могут определить свои сильные и слабые стороны. Выявление этих параметров позволяет в дальнейшем улучшать имеющиеся бизнес-процессы и операции. Выдвигают ряд требований, которыми должны обладать такие системы:

- механизмы управления и обработки информации должны работать в реальном масштабе времени,
- нулевое время простоя,
- большая часть управляющих воздействий должна вырабатываться без участия пользователя,

– умение обрабатывать нестандартные и аварийные ситуации.

Выделяется ряд требований к данным. Поступающая информация должна содержать полный набор фактов, которые необходимы для принятия решения. Также она должна фильтроваться, так как большой поток данных может вызывать серьезные проблемы с нагрузкой на систему.

Большая часть информации носит динамический характер, как, например, торговые каналы фондового рынка. Её полезность часто недолговечна [8]. Рассматриваемые системы часто могут страдать от дефекта задержек сообщении об отклонениях и принятии корректирующих мер, поскольку информация может устареть на момент её обработки. Необходимо ориентироваться на данную особенность подобного рода систем и предусматривать необходимые алгоритмы на этапе проектирования.

Информационно-управляющая система должна не только обрабатывать данные, проверять и выявлять отклонения, но и уметь вызывать решения для исправления таких отклонений. Например, если датчик на складе обнаружит, что расход электричества в доме достигнет критической отметки, то система должна проанализировать и отключить неиспользуемые человеком приборы. Хорошая система управления, должна делать акцент на исключениях. Нет необходимости уведомлять пользователей о деятельности, которая проходит без всяких проблем. Необходимо доводить до человека только важные или критические ситуации.

Система должна быть гибкой, иметь возможность интеграции средств контроля и возможность постоянно расширяться. Информационная среда носит очень динамичный характер. Технологические изменения происходят очень быстро. Жесткая система управления не подходит для изменяющейся среды.

Подобные системы можно классифицировать по типу выполнения задач, по типу управляемых процессов и по архитектурному стилю (рисунок 1).



Рисунок 1 – Классификация информационных систем

Программные системы могут быть классифицированы на трансформационные системы (transformational system) и реагирующие системы (reactive system). Трансформационные системы функционируют по мере необходимости, они преобразуют входные данные в выходные. Порядок действий в таких системах не имеет значение, важно лишь правильный результат. Реагирующие системы, напротив, функционируют при любых изменениях внешней среды. В таких системах необходимо, чтобы каждому входному событию соответствовало своё определённое действие. Порядок выполнения действий имеет существенное значение, так это напрямую влияет на результат.

По типу управляемых процессов системы классифицируются на статические и динамические. В статической системе переменные или свойства системы не изменяются вовсе или на протяжении длительного времени. Основная цель таких систем – это обеспечение надёжного и удобного управления данными, которые не требуют частого обновления. Самыми распространёнными статическими системами являются архивные системы, которые используются для хранения исторических данных, документов и различной научной информации. Динамические ИУС необходимы для управления процессами и данными, которые часто изменяются. Основная

задача таких систем – это оперативное реагирование на изменения в управляемом объекте и адаптирование к новым условиям в реальном времени. Такие системы способны интегрироваться с различными источниками данных и сервисами для получения актуальной информации. Такие системы, например, могут применяться в управлении производственными процессами, когда необходим мониторинг и управление на технологическом предприятии в реальном времени, или в сложных системах безопасности, где необходимо анализировать большие потоки данных и моментально принимать решения. Таким образом, отличие статических и динамических ИУС заключается в их подходе к управлению данными и процессами. В то время, как статические системы ориентированы на стабильность и надежность, а динамические системы акцентируют своё внимание на гибкости, адаптивности и способности к быстрому реагированию на изменения.

По архитектурному стилю системы могут быть монолитными и микросервисными. Монолитные системы содержат всю функциональную часть в едином приложении. Обычно с такими системами возникают сложности при их масштабировании. Микросервисные системы, наоборот, разделяют функциональную часть на множество служб (микросервисов). Они легко подвергаются масштабированию без необходимости остановки всей системы.

1.1.2 Понятие и роль обработки событий в информационно-управляющих системах

Под событием необходимо понимать действие, которое произошло или будет происходить. Например, сообщение в службе новостей, электронное письмо, сигнал датчика или текстовое сообщение. События бывают разных размеров. Некоторые из них представляют собой небольшие события, такие как получение информации от датчика температуры, а другие – очень большие события, такие как авария в производственном цехе предприятия.

События имеют собственный контекст, например, время, когда они произошли, сколько времени потребовалось для их возникновения,

взаимосвязь с другими событиями или какие объекты их инициировали. Атрибуты времени и отношения между событиями являются важными подсказками, используемыми при их обработке, чтобы определить, какие из них являются наиболее важными для бизнес-процессов в текущий момент времени. Обработка событий очень важна, так как она является возможностью принимать соответствующие действия.

События могут возникать в разных сферах. На предприятии события могут быть инициированы датчиками. В корпоративной сети события могут создаваться при запуске и завершении бизнес-процессов или сбое. Деятельность предприятия и его бизнеса может контролироваться и изменяться в результате событий.

Архитектура обработки событий основана на взаимодействиях между тремя компонентами: источником событий, обработчиком событий и потребителем событий (рисунок 2).

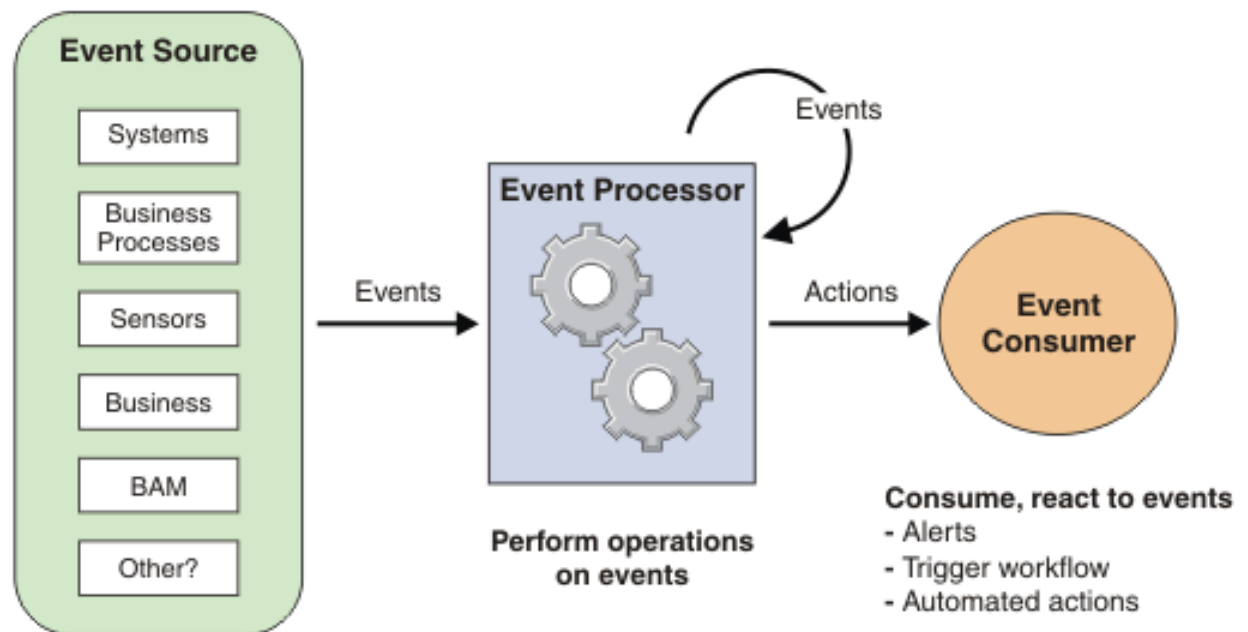


Рисунок 2 – Структурная схема обработки событий в ИИС

Источник событий (Event Source) выдает события в систему обработки событий (Event Processor). Примерами источников событий могут являться датчики или исполнительные механизмы на производстве. Система обработки событий может выполнять различные действия с событиями: фильтрацию,

добавление дополнительных данных в контекст события, форматирование, маршрутизацию отдельных бизнес-событий, анализ, создание простого или управляющего события [9]. Потребитель (Event Consumer) должен реагировать на поступающее событие. Реакция потребителя может быть как простой, например, обновление базы данных, так и сложной, например, аварийное выключение линии производства на предприятии.

1.1.3 Свойства реагирующих дискретно-событийных динамические ИУС

Дискретно-событийные динамические системы (DEDS) представляют собой класс информационно-управляющих систем, эволюция элементов которой полностью зависит от возникновения асинхронных событий в дискретный момент времени [10]. В DEDS состояние системы изменяется не непрерывно, а в определённые моменты времени, которые соответствуют возникновению различных событий. Дискретные события позволяют точно управлять процессами и отвечать на изменения с высокой степенью гибкости. Такие события могут включать в себя изменения состояний, внешние запросы или команды управления.

Такой класс систем имеет асинхронную событийно-ориентированную модель управления, которая позволяет системе параллельно обрабатывать множество независимых друг от друга событий и оперативно реагировать на них по мере возникновения. Благодаря асинхронной обработке событий, DEDS является высокопроизводительной системой, которая способна обрабатывать большие потоки данных, что критически важно для систем реального времени.

DEDS обладают высокой степенью адаптивности, так как поведение системы способно динамически изменяться в зависимости от изменений во внешней среде или во входных данных. Системы могут адаптировать свое поведение для оптимизации реакций на события.

DEDS обязательно спроектированы таким образом, чтобы быть способным быстро восстанавливаться после сбоев и минимизировать влияние

ошибок на работу системы. Это делает такой класс систем надежным и устойчивым к ошибкам, что необходимо при непрерывной обработке критически важных процессов.

Несмотря на дискретность и асинхронность, в таких системах могут использоваться алгоритмы и методы прогнозирования, позволяющие предвидеть будущие события и подготавливать систему к их эффективной обработке или предотвращению проблемных ситуаций.

1.2 Применение технологий потоковой обработки и активных правил для обнаружения событий и синтеза управляющих воздействий в реагирующих ИУС

Потоковая обработка событий – это современный способ работы с информацией в ДС ИУС. Применение активных правил совместно с потоковой обработкой событий значительно увеличивает производительность системы в масштабе реального времени.

1.2.1 Потоковая обработка событий

С помощью потоковой передачи любая информация в систему подается в режиме реального времени, что позволяет достигнуть моментальной реакции по управлению различными процессами.

Перед построением подобного рода систем необходимо точно понимать для чего понадобится потоковая обработка событий. Главным аналогом является пакетная обработка. Проведем сравнение данных подходов.

При использовании пакетной обработки данные в систему подаются с задержкой. В качестве задержки выступает период вычислений, который всегда присутствует в информационных системах. Чем больше происходит итераций, тем больше будет отставание. Такая задержка критична в некоторых бизнес-процессах. Потоковая обработка устраняет данный недостаток и предоставляет механизмы для передачи данных в реальном масштабе времени.

Еще одним ключевым отличием является нагрузка на вычислительную систему. При вычислениях в пакетном режиме возникают случаи, когда

период вычислений может быть слишком длительный (день, неделя). Помимо проблемы с долгими вычислениями здесь ярко выражена проблема, связанная с пиковой нагрузкой. От большого количества обрабатываемой информацией в один период система может работать на пределе своих возможностей, постоянно упираясь в различного рода лимиты. В конечном итоге это может привести к сбоям всей системы. В системах с потоковой обработкой событий данная проблема не выражена, так как нагрузка на аппаратные ресурсы происходит равномерно. В таких системах будущая нагрузка является более предсказуемой, что положительно сказывается на предварительном масштабировании технических средств.

Сложность разработки и поддержки систем с потоковой обработкой отличается от её аналогов. Тестирование, управление и получение информации намного тяжелее, если сравнивать с пакетной обработкой. Также такую систему сложнее восстанавливать в случаях сбоя. В разы сложнее уловить момент, когда именно произошел сбой и вернуться на последнюю критическую точку.

Принятие решения к выбору такого рода систем в итоге сводится к двум ключевым вопросам: действительно ли важна обработка информации в реальном времени и много ли потоковых источников данных. Если ответ будет положительным, то именно в таком случае система потоковой обработки данных будет хорошим выбором.

Архитектуру систем с применением потоковой обработки событий можно разделить на несколько логических блоков [11]:

- обнаружения и первичной обработки событий,
- очередей сообщений,
- обработки и анализа событий,
- глобального хранилища,
- взаимодействия с управляемыми объектами.

Архитектурная диаграмма блоков представлена на рисунке 3.

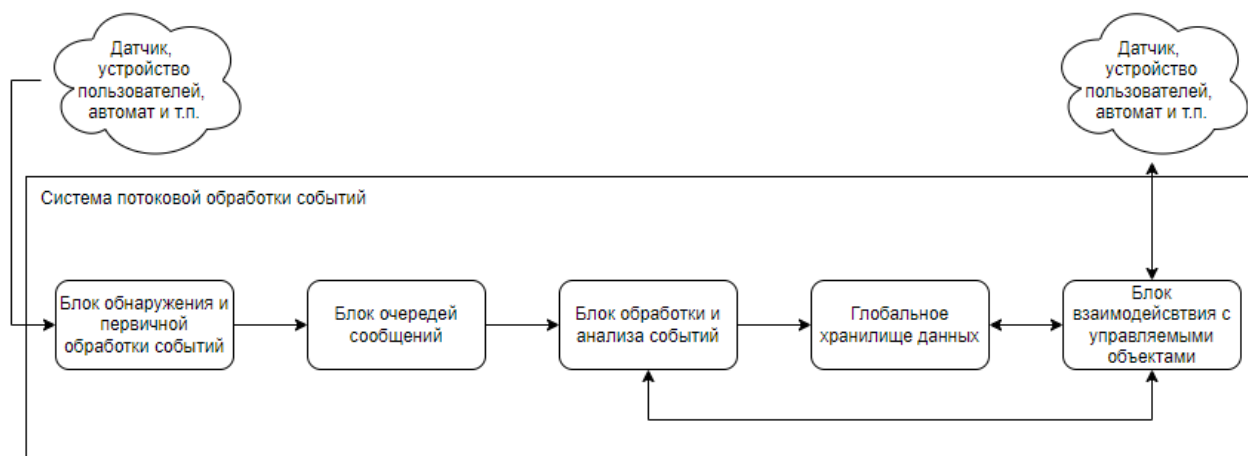


Рисунок 3 – Архитектурная диаграмма блоков системы с применением технологии потоковой обработки событий

Важно понимать, что определение блоков не имеет строгий характер, они могут модифицироваться и проектироваться под конкретные требования. Блоки не подразумевают определенного сценария развертывания, физически они могут находиться в различных местах.

Одной из трудностей в построении таких систем является поддержка приема событий, обеспечение их сохранности и передача в режиме реального времени. Архитектура должна иметь средства для решения подобных задач, особенно при постоянных и больших потоках информации, которая может поступать как от физических устройств, так и от различных облачных решений. Актуальным примером являются устройства интернета вещей (IoT), которые передают данные в реальном времени.

В такого рода системах доставкой данных занимаются брокеры сообщений. Они, организуя взаимосвязь между отдельными элементами архитектуры, всю работу по пересылке информации берут на себя. Благодаря этому снижается нагрузка на различные элементы и увеличивается производительность системы в целом.

В работе брокера используются две основные сущности: издатель (producer) и потребитель (consumer). Существует две популярные схемы работы брокеров сообщений в системах потоковой обработки данных:

- отправка сообщений напрямую от издателя к потребителю,
- паттерн публикации/подписки.

В первом случае сообщение пересылается однократно, используя при этом очередь (рисунок 4). Издателю необходимо знать своего потребителя.



Рисунок 4 – Прямой обмен сообщениями через брокера

Во втором случае, наоборот, знать адрес потребителя не обязательно. В контексте с сообщением передается название его темы. Потребители с свою очередь подписываются на конкретные темы и получают сообщение при его появлении (рисунок 5).



Рисунок 5 – Обмен сообщениями через паттерн подписчик/читатель

Ко всему прочему, они обеспечивают надежность доставки сообщений. В них реализованы механизмы повторной или многократной отправки сообщений, в случаях если данные не были доставлены.

Собрав необходимые сообщения, система должна уметь выполнять для них фильтрацию, статистическую обработку и другие операции для подготовки данных к анализу. Обработанные данные структурируются и

записываются в соответствующие хранилища с низкой задержкой. В блоке обработки и анализа данных происходит выявление закономерностей, сброс системы или генерация уведомлений при выполнении определенных условий в потоке. При этом события могут проходить через несколько обработчиков потока, которые передают их от одного элемента системы к другому. Например, для полной обработки и анализа может потребоваться несколько микросервисов.

Для постоянного хранения данных используются более масштабные хранилища, нежели для их анализа. Обычно это большие реляционные базы данных, которые имеют свойство легко масштабироваться.

1.2.2 Концепции активных правил

Активные правила всесторонне исследовались за последние несколько десятков лет. Были предложены несколько подходов по работе с активными правилами, которые успешно применялись в коммерческих организациях. Данные подходы привели к различным формализмам и языкам.

В области экспертных систем широко исследовались продукционные правила [12]. Зачастую они реализовывали семантику с прямой цепочкой «действие – условие», работая с набором данных. Каждое правило состоит из предварительного условия и постусловия, которым набор данных либо соответствует, либо нет. Например, если данные удовлетворяют условию, то правило применяется успешно. Алгоритмы эффективны в случаях, когда необходимо просто узнать, какие новые факты верны, так как такие правила могут имитировать вывод, посредством утверждения вывода как следствия доказанного условия, то есть «если условие, то факт окажется верным».

Вопреки того, что продукционные правила реагируют на изменения состояния условий и не имеют явной формализации событий, как в правилах ЕСА, в них были реализованы расширения правил до обработки сложных событий (СЕР), например в таких системах, как TIBCO BusinessEvents и Drools. Типы и классы объявляются совместно с правилами. Экземпляры данных о событиях динамически добавляются в базу фактов. Происходит

фильтрация и присоединение экземпляров объявленных событий к условиям продукционного правила посредством сопоставления типизированных шаблонов. Если они проходят список условий, то активируется часть действия правила. То есть условная часть таких правил используется для определения фильтров событий, а обработка самих событий выполняется посредством сопоставления с шаблоном.

Активные базы данных получили свое развитие в 1990-х годах. В их основу, в том числе, были заложены методы экспертных систем. Они использовались для автоматизации действий в ответ на соответствующее событие.

Для изменения состояний в базах данных была разработана концепция ЕСА. Она позволила пользователям объявлять активные правила, выполнение которых может автоматически модифицировать данные и инициировать другие различные действия. В основе правила находится логический блок, который при выполнении условий вызывает заранее запрограммированные действия.

Было создано несколько систем активных баз данных, такие как ADL, ACCOOD, NAOS, HiPac. Данные системы способны обнаруживать события и выполнять их процедурную обработку. Были созданы алгебры событий, например, SAMOS или ODE, где есть возможность конструировать комплексные события при помощи разных операторов, таких как and, or, sequence и так далее. ODE является объектной базой данных и реализует механизм обнаружения событий с использованием конечных автоматов. SAMOS использует цветные сети Петри при работе с активными правилами. Стилль правила ЕСА и операторы алгебры событий оказали влияние на развитие языков обработки событий на основе правил (EPL).

Концепция ЕСА хорошо подходит для обработки событий в реальном времени, т.к. «отправной точкой» для исполнения правила служит именно само событие и его метаданные.

1.2.3 Классификация активных правил

Система реагирует на появление экземпляра того или иного события через исполнение одного или нескольких активных правил, связанных с типом экземпляра события. В общепринятой классификации активное правило может иметь две разновидности:

- классическое правило,
- расширенное правило.

Классическое активное правило имеет стандартный вид ЕСА (Event-Condition-Action), где Е (event type) – тип события, при появлении экземпляра которого должно быть исполнено правило; С (condition) – условие, логическое выражение над контекстом типа события, при истинности которого для экземпляра события происходит «срабатывание» активного правила; А (action) – действие, набор операций над объектами управления, определяющее реакцию на возникновение событий. Действие будет выполняться в случае истинности условия.

Расширенное ЕСА-правило имеет в себе альтернативное действие, которое выполняется в случае ложности условия С (condition). Такое правило является объединением двух взаимно исключающих по условию активных правил. Данный тип правила похож на конструкцию «if-then-else» в структурном программировании.

Возникает необходимость в формировании дополнительных классов правил для метамоделирования ДС ИУС. Это также необходимо для минимизации множества активных правил и требованиями усложняющихся бизнес-процессов.

1.3 Применение метамоделирования для проектирования и реализации систем потоковой обработки событий с применением активных правил

Одним из наиболее эффективных способов описания информационных объектов являются метаданные [13]. Они могут быть использованы для

описания внутренних объектов системы, определения процессов обработки событий и выполнения активных правил.

1.3.1 Понятие и роль метаданных

Метаданные представляют собой дополнительную информацию о свойствах объекта, описывающую различные его аспекты. Они могут касаться самого объекта, его поведения, бизнес-операций, бизнес-процессов, служебных данных для контроля и управления доступом и т.д.

Метаданные имеют свою специфику, относительно тех объектов, которые они описывают. В информационно-управляющих системах метаданные имеют важное значение и необходимы для выполнения следующих задач:

- предоставление сведений о контексте поступающих событиях в систему для их дальнейшей обработки,
- отбор и систематизация наблюдаемых и управляемых объектов, событий, активных правил,
- описание способов реагирования посредством активных правил,
- выявление сбоев и неисправностей системы на основе статического и динамического анализа метаданных,
- поиск стандартных шаблонов событий,
- выполнение дополнительных технических задач, связанных с внутренней обработкой событий.

В информационно-управляющих системах источником событий может выступать как отдельный объект, так и целая система. Для обеспечения успешного обмена и интерпретации описаний необходимо стандартизировать формат метаданных, а также синтаксис и семантику их элементов [14]. Унифицированный стандарт облегчает обработку данных, поступающих из различных источников. В качестве основных требований к стандартам метаданных можно выделить следующие:

- структурированность и формализованность,
- универсальность в рамках заданных информационных объектов,

- достаточная выразительность для возможности автоматизированного управления,
- возможность определения ограничения целостности,
- возможность расширения метаданных без нарушения бизнес-логики системы.

Стандарты метаданных следует определять на этапе проектирования архитектуры системы. Метаданные можно классифицировать на три типа в зависимости от их назначения:

- описательные,
- структурные,
- служебные.

Описательные метаданные предназначены для идентификации и включают контекст информационного объекта. Структурные метаданные описывают общую структуру объекта, включая его объем, формат, связи с другими объектами или ресурсами и т.д. Служебные метаданные содержат внутреннюю служебную информацию, такую как дата создания или изменения объекта, права доступа и другие данные, необходимые для решения внутренних задач системы.

1.3.2 Основы метамоделирования

Моделью называется объект-заменитель, который способен дать представление об объекте-оригинале: его особенности, характеристики и так далее. Большая гибкость при разработке систем достигается благодаря использованию метамodelей. Они помогают описать модель в формальном виде, что в дальнейшем упростит этапы планирования и разработки информационных систем. Примером метамodelи может являться диаграмма, написанная на языке графического описания и моделирования UML. Так диаграмма классов будет метамodelью, а описываемые классы – моделью.

Абстрактные характеристики системы должны быть значимыми с точки зрения моделирования и поведения модели самой системы. В случае сложных систем следует ограничиваться одной моделью, так как описание всех свойств

в одной модели может усложнить разработку. Хорошим подходом к созданию любой сложной системы является использование комбинаций множества моделей, которые могут не зависеть друг от друга. Такой метод позволяет без затруднений описать все аспекты разрабатываемой системы и удобно поддерживать её на протяжении всего жизненного цикла.

По степени абстракции модели могут быть:

- концептуальные, представляющие высокоуровневое описание проблемы в заданной предметной области,
- определяющие, специфицирующие поведение системы и характеризующие её основные особенности,
- реализационные, изображающие внутреннюю структуру системы и её бизнес-логику.

В общем плане метамодель можно рассматривать как язык моделирования, используемый для формального описания системы. Существует также деление на лингвистические и онтологические метамодели. Различие заключается в том, что лингвистическая модель описывает язык, независимый от предметной области, тогда как онтологическая модель описывает предметно-зависимый язык.

В научных трудах зачастую встречается четырёхуровневая иерархия моделей, которая используется при проектировании информационных систем [15]. Иерархия представлена на рисунке 6.

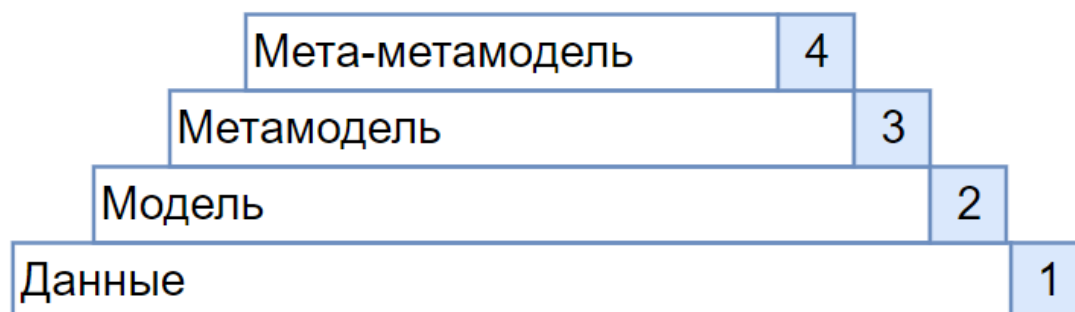


Рисунок 6 – Многоуровневая иерархия моделей информационной системы

На самом низком уровне находится модель, содержащая в себе данные для описания предметной области. На следующем уровне располагается онтологическая метамодель самой предметной области для уровня 1.

Метамодель на третьем уровне определяет лингвистическую метамодель для уровней 2 и 1. С лингвистической метамоделью зачастую работают разработчики, аналитики, проектируются системы при помощи CASE-средств и т.д. Самый верхний уровень встречается реже остальных. Он определяет формализмы, при помощи которых описывается метамодель третьего уровня.

1.3.3 Обзор подходов к разработке информационных систем с применением метамоделирования

Можно выделить несколько основных подходов при разработке систем с использованием метамоделирования, где количество создаваемых уровней моделей может варьироваться в зависимости от конкретной задачи.

При традиционном подходе к разработке информационных систем данные, описывающие состояние предметной области, хранятся внутри самой системы. Модель предметной области может быть описана на любых формальных или естественных языках. Модель разрабатывается аналитиками, после чего разработчики внедряют ее с помощью выбранных инструментальных средств. Недостатком данного подхода является то, что со временем и развитием информационной системы разработчики могут отходить от исходной метамодели. Она не обновляется при внесении изменений в различные структуры данных или алгоритмы их обработки, что приводит к различным трудностям.

При разработке с использованием CASE-средств модель предметной области определяется формально и описывается исключительно в терминах метамодели, которая может быть реализована CASE-системой. В этом подходе метамодель также разрабатывается аналитиками и затем реализуется разработчиками. На данный момент существует множество CASE-средств, которые позволяют редактировать ранее созданные модели и даже способствуют автоматической генерации кода. Такие модели могут создать объектно-ориентированные классы данных, организовать базу данных и сконфигурировать к ней доступ, например, на основании пользовательских ролей. Данный подход имеет значительное преимущество – экономия времени

на этапах разработки. Также возможно поддерживать соответствие модели и разработанной системы, но стоит учитывать, что любые изменения модели приводят к изменению кода.

Для разработки систем, управляемых метаданными, также используется подход с трехуровневой иерархией моделей, где созданная метамодель находится внутри информационной системы во время её функционирования. В этом случае программное обеспечение действует как интерпретатор, а модель выполняет роль управляющего системой и устанавливает правила её работы. Основным недостатком такого подхода является снижение производительности системы во время её функционирования. Также необходимо предусмотреть возможность подключения сторонних программных модулей для реализации специфических бизнес-задач конкретной системы. Преимуществом является то, что изменения в метамодели не требуют повторного кодирования – система адаптируется и начинает работать в соответствии с новой метамodelью.

Подход Domain Specific Modeling (DSM) с автоматической генерацией кода позволяет моделировать в терминах конкретной предметной области. В этом подходе используется четвертый уровень иерархии – мета-метамодель. Для этого применяется собственный язык описания, включающий термины из предметной области системы. На основе модели четвертого уровня создается CASE-средство, с помощью которого разрабатывается информационная система. Данный подход получил широкое применение в No-Code инструментах разработки (рисунок 7).

Подход DSM с интерпретацией метаданных является комбинацией двух из ранее рассмотренных подходов. В данном случае CASE-средства уже позволяют генерировать модели и интерпретировать их в ходе функционирования системы. Для данного подхода появляется необходимость разработки run-time компонента. Данный подход применяется в CASE-технологии METAdata System (METAS).

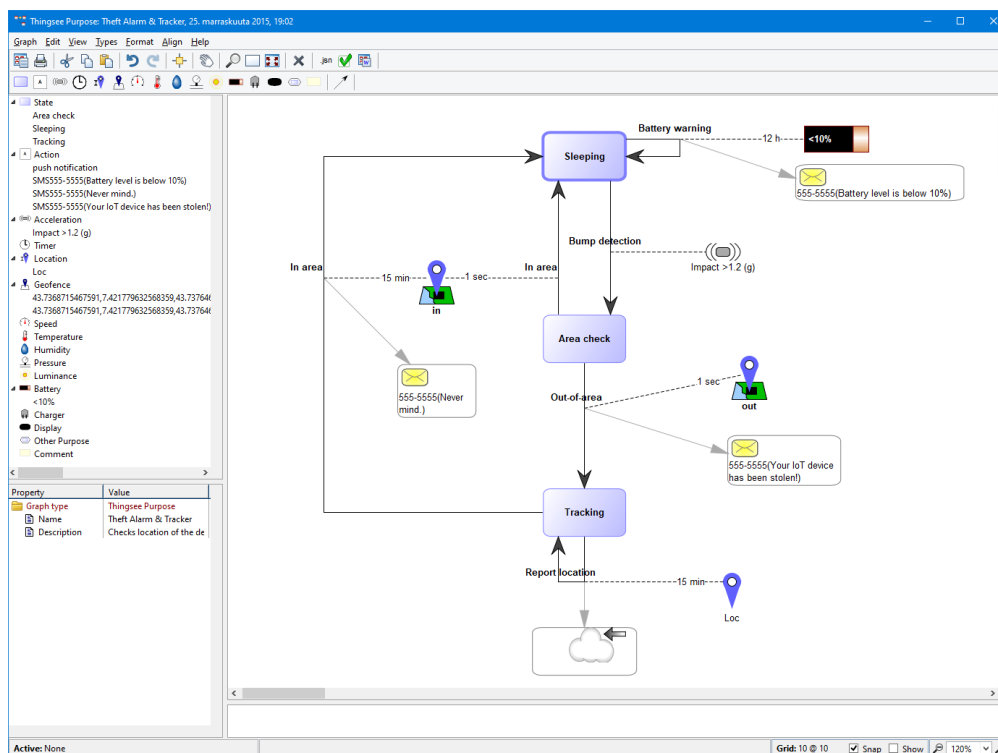


Рисунок 7 – Применение DSM при разработке системы IoT

METAS облегчает создание динамически-настраиваемых систем с использованием взаимосвязанных метаданных, которые описывают информационную систему с разных сторон и на разных уровнях абстракции. Разработанные «интеллектуальные» системы обладают способностью изменять свою бизнес-логику в условиях модификации метаданных. Технология основана на языке моделирования UML и предметно-ориентированных языках для разработки моделей системы и описания активных правил. Используются технологии XP (eXtreme Programming), RUP (Rational Unified Process), платформа .NET Framework.

Метаданные в такой системе разделяются на несколько слоёв, каждый из которых представляется собственной моделью: физической, логической или презентационной. Физическая модель описывает представление и структуру объектов в базе данных, например, данные в реляционных таблицах, которые связаны между собой. Данная модель служит основой для логической модели, в которой описывается поведение и допустимые операции над данными. Презентационная модель описывает взаимодействие

пользователя с системой при помощи интерфейса управления. Модели метаданных METAS изображены на рисунке 8.

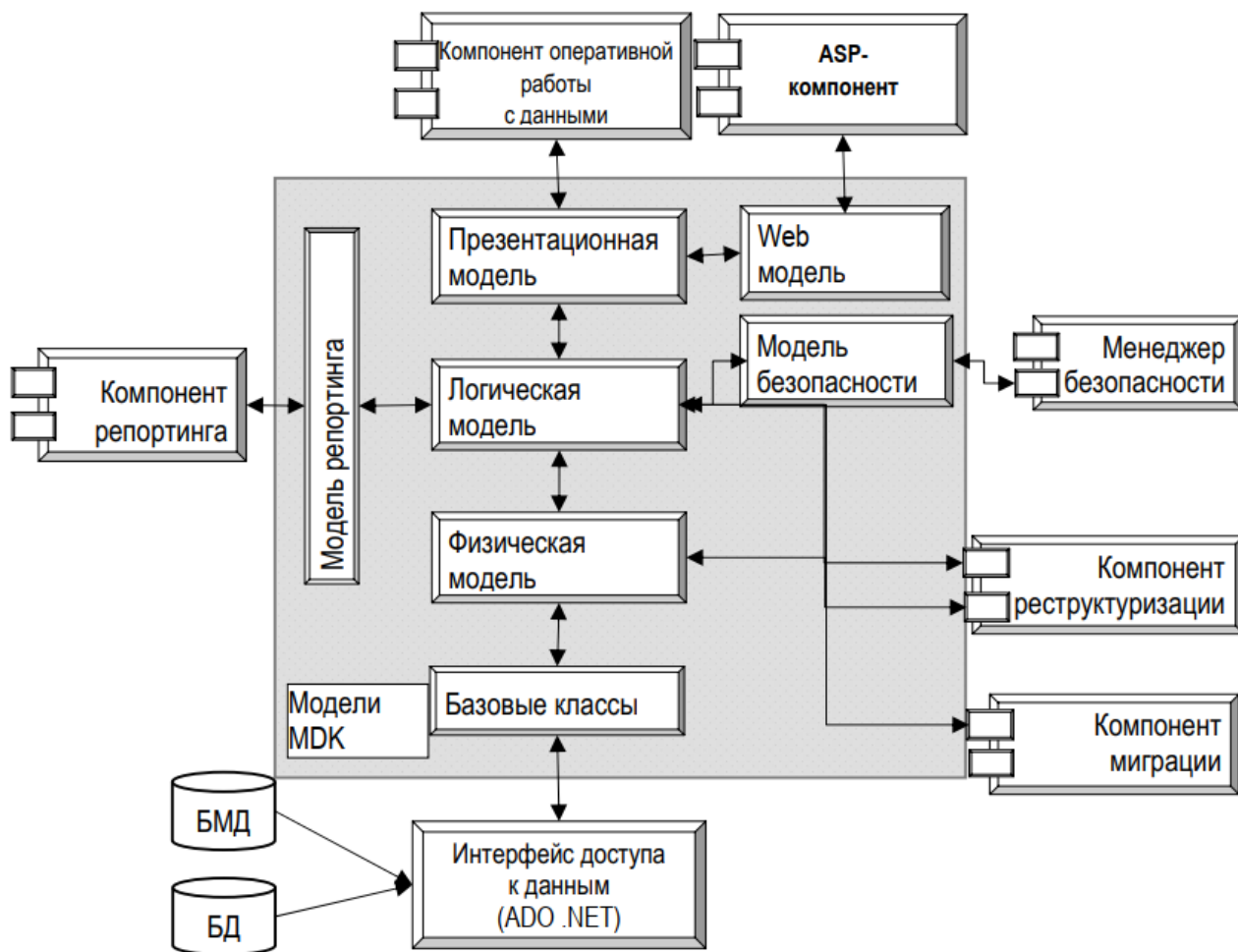


Рисунок 8 – Модели метаданных METAS

Указанное множество метаданных может быть расширено при помощи добавления новых моделей, которые описывают иные стороны информационной системы. Например, изображенная на рисунке 11 модель безопасности, позволяет настраивать уровень доступа отдельных пользователей, права на выполнение операций по работе с метаданными и т.д. Указанное CASE-средство позволяет создать систему с клиент-серверной архитектурой.

К главным преимуществам можно отнести:

- возможность динамической конфигурации системы в условиях изменяемости метаданных и потребностей бизнес-процессов,
- возможность интеграции с внешними наблюдаемыми или управляемыми системами,

– допустимость использования и создания систем пользователями без специальной подготовки.

Основным недостатком является необходимость иметь мощные средства сбора, анализа и обработки данных.

1.4 Анализ требований к составу, архитектуре и процессам управления метаданными в реагирующих ИУС потоковой обработки событий и исполнения активных правил

Описываемые ДС ИУС, основанные на потоковой обработке событий и исполнении активных правил, представляют собой сложные динамические системы, которые способны автоматически реагировать на изменения во внешней и внутренней среде. Как было сказано ранее, важной особенностью таких систем является управление метаданными, которое играет ключевую роль в обеспечении своевременной и корректной обработки событий.

Метаданные в подобных системах могут иметь сложную структуру и содержать:

- описания событий и данных, включая их структуру, типы, источники, а также схемы данных, используемых в системе,
- описания логики обработки событий и условия активации правил,
- конфигурации наблюдаемых и управляемых систем, включая параметры подключения к внешним источникам данных и сервисам,
- конфигурационные параметры сервиса,
- пользовательские метаданные и правила доступа к данным, а также политики безопасности, применяемые к обработке информации.

Правильное планирование и разработка архитектуры управления метаданными может положительно сказаться на быстродействии, отказоустойчивости и масштабируемости системы. Архитектура должна обеспечивать хранение, управление и распределение метаданных внутри системы. Ключевыми компонентами архитектуры являются:

- репозитории метаданных, содержащие все метаданные внутри системы и обеспечивающие их целостность, согласованность и доступность,
- компоненты и сервисы, отвечающие за создание, обновление, извлечение, тиражирование и удаление метаданных,
- интерфейсы доступа к метаданным, которые позволяют разработчикам и системам-потребителям получать доступ к метаданным, необходимым для выполнения операций обработки событий и правил.

Архитектура должна быть модульной, чтобы различные компоненты могли быть легко добавлены, заменены или обновлены без необходимости модификации всей системы. Также необходимы механизмы безопасности и контроля доступа к метаданным на всех уровнях архитектуры, чтобы обеспечить защиту чувствительной информации.

При использовании микросервисного подхода к построению архитектуры реагирующих ИУС, управление метаданными приобретает особые черты, отражающие децентрализованный характер системы. В такой архитектуре каждый микросервис может функционировать как независимый элемент, взаимодействующий с другими микросервисами через различные протоколы связи, например HTTP и WS. Данный подход обеспечивает высокую гибкость и масштабируемость системы, а также упрощает разработку, тестирование и сопровождение отдельных компонентов [16]. К особым требованиям к такой архитектуре можно отнести:

- наличие центрального репозитория метаданных, который служит основным источником метаданных и управляется отдельным элементом в системе,
- наличие локальных репозитория в микросервисах, которые содержат необходимую часть метаданных для конкретного микросервиса,
- механизм тиражирования и синхронизации метаданных между центральным и локальными репозиториями.

Локальные репозитории помогают увеличить отказоустойчивость и быстродействие системы, но требуют дополнительной разработке механизмов

и алгоритмов тиражирования и синхронизации, что зачастую является сложной задачей. Эти средства требуют тщательной проработки и тестирования, поскольку сбои в синхронизации метаданных могут привести к некорректной обработке событий. Синхронизация может быть реализована через периодические запросы к центральному репозиторию или через механизмы Pub/Sub, которые отслеживают изменения метаданных с использованием модели издатель-подписчик.

Эффективное управление метаданными требует чёткого определения процессов, связанных с их жизненным циклом (рисунок 9).



Рисунок 9 – Процессы жизненного цикла управления метаданными

Процесс инициализации метаданных начинается с определения источников событий, таких как внешние сервисы, базы данных, системы управления процессами и т.д. На данном этапе могут определяться схемы и контекст поступающих событий, описания интерфейсов и протоколов взаимодействия. Далее происходят процессы конструирования метаданных для определения активных правил, условий их активации и создания управляющих воздействий.

Следующей группой процессов управления является мониторинг и аудит изменения метаданных. Системы мониторинга необходимы для оповещения внутренним и внешним элементам системы об изменениях в метаданных. Зачастую применяется механизмы версионирования метаданных, которые предоставляют возможность отката к предыдущим версиям. Системы аудита отражают все изменения, связанные с метаданными, в иерархической последовательности, что помогает администраторам системы или соответствующим алгоритмам легче находить причинно-следственные связи в случаях возникновения каких-либо проблемных ситуаций.

Процессы тиражирования и синхронизации являются одними из самых сложных и необходимых для поддержания консистентности и актуальности метаданных по всей системе. Тиражирование включает в себя распределение метаданных из центрального репозитория в локальные репозитории микросервисов, что позволяет микросервисам работать с актуальной версией метаданных с высокой производительностью даже при отсутствии прямого доступа к центральному репозиторию. Синхронизация обеспечивает, что все изменения в метаданных, произведенные в одном месте системы, будут отражены во всех соответствующих компонентах. В условиях, когда изменения метаданных могут производиться параллельно в разных частях системы, необходимо обеспечить механизмы разрешения конфликтов для поддержания единого состояния данных. Существуют различные подходы к синхронизации: централизованные и децентрализованные. Централизованные подходы предполагают, что все изменения проходят через центральный узел, который контролирует обновления и в дальнейшем рассылает эти обновления каждому элементу системы. Децентрализованные подходы позволяют микросервисам непосредственно синхронизировать изменения между собой, это особо актуально, когда нет необходимости дополнительно верифицировать изменения и выполнять с ними дополнительные операции.

Процессы управления необходимо спроектировать и реализовать таким образом, чтобы минимизировать задержки, сохранить производительность и отказоустойчивость ДС ИУС на требуемом уровне.

1.5 Выводы по разделу 1

В данном разделе были рассмотрены аспекты применения метаданных в реагирующих дискретно-событийных динамических информационно-управляющих системах, с акцентом на реагирующие ИУС, основанные на потоковой обработке событий и исполнении активных правил. Рассмотрена актуальность и проблематика исследуемой предметной области.

На основе материалов данного раздела можно сделать вывод, что возникает необходимость эффективного управления метаданными на всём их

жизненном цикле, включая их инициализацию, изменения, тиражирование и синхронизацию.

Для улучшения производительности и отказоустойчивости системы были выдвинуты требования к управлению метаданными в микросервисной архитектуре, предполагающей наличие центрального и локальных репозиториев.

В качестве критически важных процессов для поддержания актуальности и консистентности метаданных были выделены процессы тиражирования и синхронизации. Было исследовано, что данные процессы напрямую влияют на корректную обработку событий и исполнение активных правил.

2 Архитектура, модели и алгоритмы управления метаданными в реагирующей ИУС потоковой обработки событий и исполнения активных правил в рамках единой метамодели

Данный раздел затрагивает процессы управления метаданными на основе разработанной единой метамодели для ДС ИУС. Приведена схема единой метамодели с детальным описанием каждого элемента. Рассмотрены особенности процессов управления при использовании распределённой архитектуры метаданных.

2.1 Единая метамодель реагирующей ИУС

Работа с метаданными реагирующей ИУС осуществляется на основе её метамодели. Схему метамодели, на которой изображены её основные элементы, можно видеть на рисунке 10.

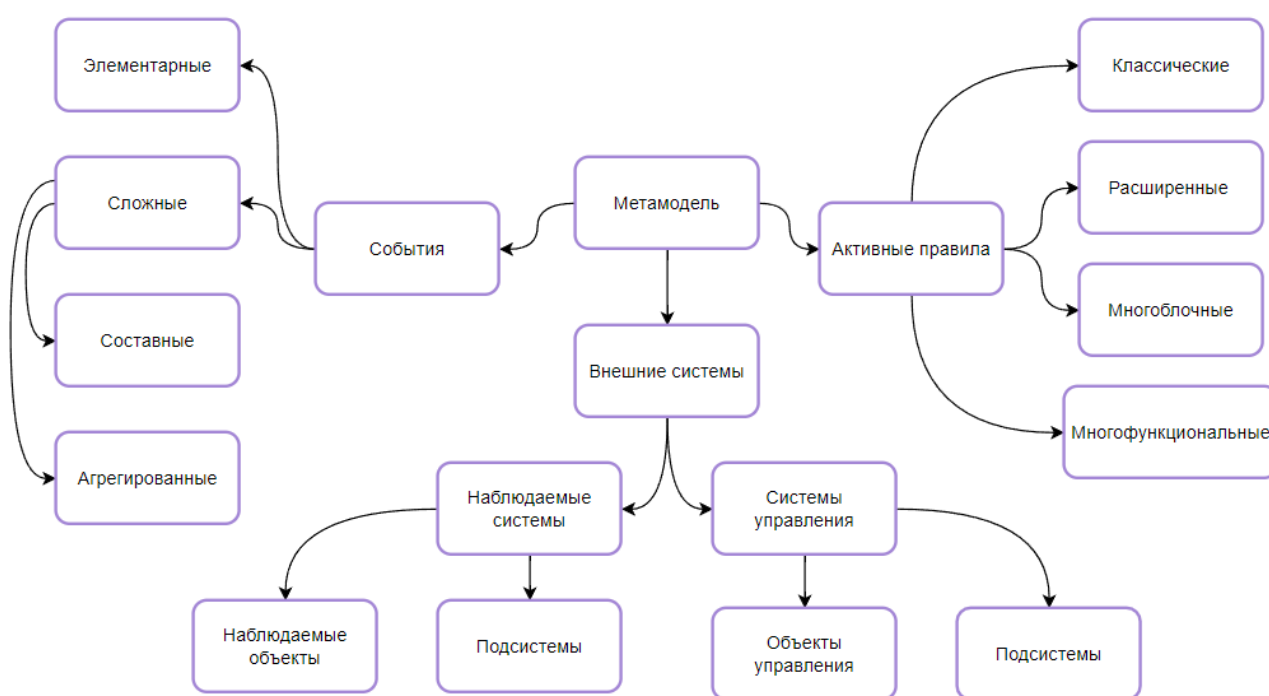


Рисунок 10 – Структура единой метамодели

Метамодель включает в себя спецификации событий и активных правил, наблюдаемые системы и системы управления, наблюдаемые объекты и объекты управления, правила обработки событий и формирования управляющих воздействий, а также другие виды объектов.

2.1.1 Мета модель внешних систем

ИУС взаимодействует с множеством сторонних систем. Рассмотрим метамодель внешних систем, которые подразделяются на наблюдаемые (Sm) и управляемые системы (Sc). Системы представляются следующим множеством

$$Sm = \{Sm_1, Sm_2, ..., Sm_n\} \quad (1),$$

где n – количество наблюдаемых систем.

Любые внешние системы могут иметь сложную иерархию и зависеть от других подсистем.

Множество внешних наблюдаемых подсистем ($Ssubm$) определяются как

$$Ssubm_i = \{Ssubm_i^1, Ssubm_i^2, ..., Ssubm_i^p\} \quad (2),$$

где p – количество подсистем, образующих наблюдаемую систему Sm_i из (1).

Системы мониторинга и их составляющие подсистемы могут включать в себя множество наблюдаемых объектов (Om), именуемые как объекты мониторинга. Эти объекты служат первоисточниками данных и различных событий, формируя множество

$$Om_i = \{Om_i^1, Om_i^2, ..., Om_i^k\} \quad (3),$$

где k – количество подключённых объектов мониторинга в рамках i -ой системы или её подсистемы.

Собранные данные от этих объектов интерпретируются как базовые (или первичные) события (Ep), формируя входящий поток событий

$$E_p = (e_1, e_2, \dots, e_i, \dots) \quad (4),$$

где e_i – конкретное событие в этом потоке.

Различные внешние управляемые системы, зарегистрированные в ИУС, определяются как

$$Sc = \{Sc_1, Sc_2, \dots, Sc_l\} \quad (5),$$

где l – количество подключённых систем управления.

Аналогично, каждая из этих систем может включать в себя несколько управляемых подсистем (S_{subc}), которые представляются как

$$S_{subc_i} = \{S_{subc_i}^1, S_{subc_i}^2, \dots, S_{subc_i}^t\} \quad (6),$$

где t – количество подсистем, формирующих систему управления Sm_i .

В рамках каждой управляемой системы или её подсистем можно выделить многочисленные объекты управления (Oc), к которым могут быть применены различные управляющие действия. Такие объекты обозначаются как

$$Sc_i = \{Oc_i^1, Oc_i^2, \dots, Oc_i^r\} \quad (7),$$

где r – количество объектов управления в i -ой системе.

Управляющие действия (A), применяемые к объектам управления, создают выходной поток действий

$$A = (a_1, a_2, \dots, a_i, \dots) \quad (8),$$

где каждое a_i – отдельное управляющее действие, являющееся результатом активации определённого правила.

Внешние системы могут одновременно выступать в роли систем мониторинга и систем управления, что приводит к пересечению их функций и сущностей. В качестве примера можно привести базы данных, которые одновременно могут выполнять функции отслеживания изменений и различные управленческие операции. Таким образом, каждый объект в такой системе может быть одновременно и объектом мониторинга, и объектом управления.

2.1.2 Мета модель событий

ДС ИУС принимают и обрабатывают огромное количество событий, поступающих от внешних источников в дискретные моменты времени, а также события, возникающие внутри системы в результате выполнения различных операций. События делятся на элементарные (простые) и комплексные (сложные), которые могут быть либо агрегированными, либо составными. (рисунок 11).

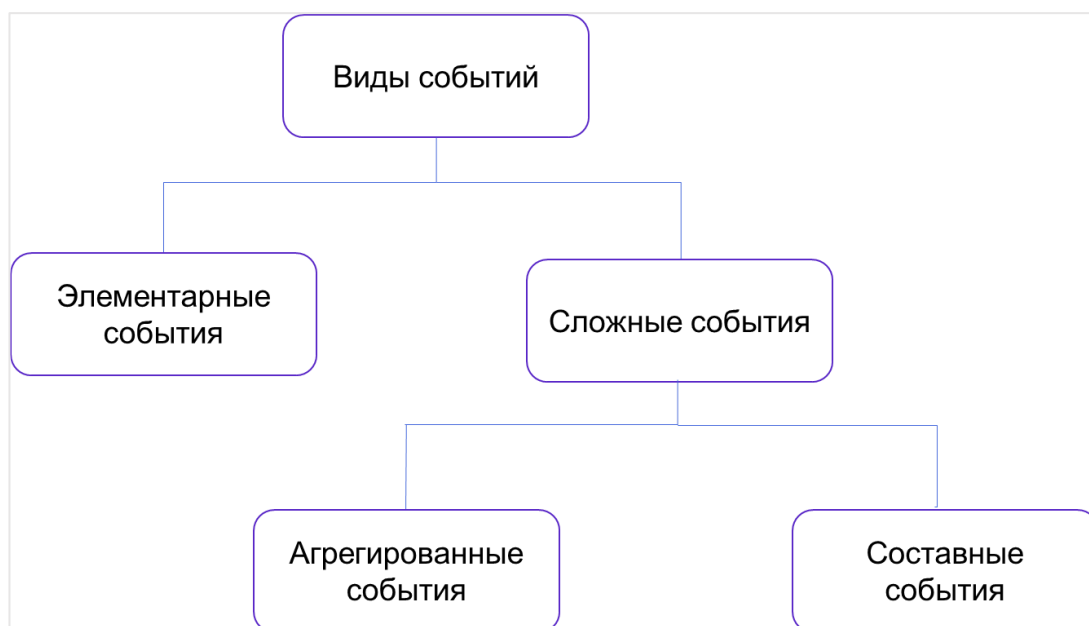


Рисунок 11 – Классификация событий

Наиболее распространёнными являются элементарные события. Они возникают при взаимодействии с внешними системами мониторинга, которые в определённый момент времени отправляют различную информацию. Кортеж элементарных событий (`e_primitive`) представляется следующим образом

$$e_primitive = \langle PEid, Context, S, T, t \rangle \quad (9),$$

где *PEid* – уникальный идентификатор этого события;

Context – набор параметров, описывающих природу события;

S – объект мониторинга;

T – тип события;

t – временная метка момента возникновения события.

Вычисляемыми структурными элементами являются сложные события, которые фиксируются системой на этапах выполнения операций над элементарными и/или другими сложными событиями. Эти сложные события можно дополнительно классифицировать на агрегированные и составные события.

Агрегированное событие формируется на основе агрегирующего запроса к входному потоку элементарных событий. Кортеж агрегированных (*e_aggregated*) событий описывается как

$$e_aggregated = \langle AEid, Context, Q, T, tb, te \rangle \quad (10),$$

где *AEid* – уникальный идентификатор события;

Context – набор параметров, описывающих природу события;

Q – запрос, выполняемый к входному потоку и агрегирующий атрибуты входных событий в конкретный промежуток времени;

T – тип события;

tb, te – временные метки, указывающие на начало и конец события.

Составное событие представляет собой комбинацию элементарных и/или сложных событий, с возможным использованием ограничений, логических операций и комбинаций в различных последовательностях. Кортеж составного события (*e_composite*) выглядит следующим образом

$$e_composite = \langle CEid, Context, C, T, tb, te \rangle \quad (11),$$

где $CEid$ – уникальный идентификатор события;

$Context$ – набор параметров, описывающих природу события;

C – комбинация элементарных и сложных событий, которые образуют составное событие;

T – тип события;

tb, te – временные метки, указывающие на начало и конец события.

Важно отметить, что контекст составного события определяется исключительно параметрами контекста входящих событий.

Контекст события ($Context$) формируется набором параметров, значения которых определяют содержательную информацию о событии, т.е. атрибутами события

$$Context = \{P_1, P_2, \dots, P_m\} \quad (12),$$

где P_i – отдельный параметр события.

Каждый элемент ИУС обрабатывает своё подмножество событий. На входящем потоке осуществляется обработка элементарных событий, после чего проводится анализ для выявления сложных событий.

2.1.3 Мета модель активных правил

В основе ИУС лежит модель Event-Condition-Active (ECA). Данная модель состоит из трех частей:

- событийная часть, которая подает в систему сигналы, исходящие от сторонних наблюдаемых систем,
- часть условия, представляющая собой логический блок, который при выполнении определенных условий вызывает заранее назначенное действия,
- часть действий, состоящая из операций по взаимодействию с системой.

Как было сказано ранее, реакция на определённые события происходит в следствии исполнения активного правила. Активное правило может

порождать дополнительные события, требующие обработки. Активные правила можно классифицировать на классические и расширенные (рисунок 12). Расширенное активное правило может быть дополнительно классифицировано на альтернативное, многоблочное и многофункциональное правила.

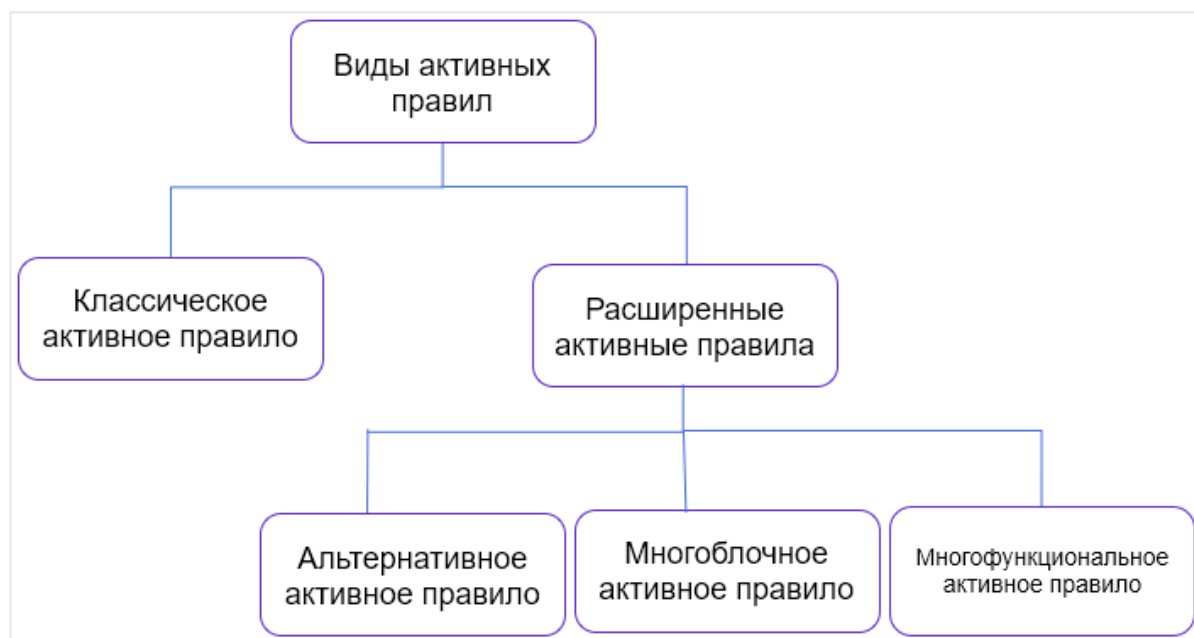


Рисунок 12 – Классификация активных правил

Классическое (элементарное) активное правило (r) в модели ЕСА описывается кортежем, который представляется как

$$r = \langle e, c, a \rangle \quad (13),$$

где e – тип экземпляра события (event type), обрабатываемого системой;

c – условие (condition), логическое выражение над контекстом типа события, при истинности которого для экземпляра события выполняется исполнение активного правила;

a – действие (action), набор операций, выполняемые над объектами управления в случае истинности условия c .

Появление расширенных активных правил обусловлено растущей сложностью бизнес-процессов и необходимостью повышения быстродействия ИУС, вследствие минимизации множества активных правил.

Расширенное активное правило представляется как

$$r = \langle e, c, \{a_{base}, a_{alt}\} \rangle \quad (14),$$

где a_{base} – действие, выполняемое в случае истинности условия;

a_{alt} – альтернативное действие, выполняемое в случае, если условие ложно.

Данное правило необходимо для объединения двух противоречащих друг-другу результирующих воздействия. Аналогом данного правила является конструкция «if-then-else», которая широко используется во множестве языков программирования.

В случае возникновения комплексного события с различным множеством результирующих событий необходимо использовать многоблочные активные правила, для которых определены несколько вариантов обнаружения

$$r = \langle \{e_1, c_1, a_1\}, \dots, \{e_n, c_n, a_n\} \rangle \quad (15),$$

где $e_1, \dots, e_n$ – множество результирующих событий;

$c_1, \dots, c_n$ – множество условий, соответствующих множеству результирующих событий;

$a_1, \dots, a_n$ – множество действий, соответствующих множеству условий.

Многофункциональное активное правило необходимо в случае, когда в качестве реакции на возникновения события и истинности условия необходимо выполнить множество результирующих воздействий. Например, при событии «пожар» необходимо сразу включить систему пожаротушения и отправить соответствующий сигнал в службу спасения. Данное правило позволяет минимизировать набор править и ускорить процесс реакции на возникшее событие.

Данное правило представлено кортежем

$$r = \langle e, c, A \rangle \quad (16),$$

где $A = \{a_1, \dots, a_k\}$ – множество экземпляров результирующих воздействий, применяемых к управляемым объектам.

Классическое активное правило является частным случаем многофункционального активного правила, т.к. множество действий состоит из одного элемента.

2.1.4 Отображение единой метамодели в модели данных

Метамодель в ИУС должна быть реализована в виде различных хранилищ данных, которые могут быть построены на разных моделях представления.

Необходимо будет отобразить единую метамодель в трёх моделях представления:

- модель «key-value»,
- объектная модель,
- реляционная модель.

Метамодель для информационных управляющих систем (ИУС) может быть представлена в виде «key-value» что позволяет эффективно хранить и обрабатывать данные в распределенных системах. Например, такая модель может быть использована в локальных репозиториях микросервисов. таких как микросервисы.

В качестве ключей (key) могут использоваться ключевые слова объектов совместно с уникальными идентификаторами. В качестве значения могут содержаться соответствующие объекты. На листинге 1 показан пример использования данной модели для события с его контекстом.

Листинг 1 – Пример представления единой метамодели в «key-value»

```
{
  "Event:1": {
    "type": "Login",
    "description": "User login event"
```

```

    }
  }
  {
    "ContextParam:1": {
      "event": "Event:1",
      "paramName": "UserLocation",
      "paramValue": "USA"
    }
  }
}

```

Для удобства проектирования центрального репозитория может использоваться модель в объектном формате (рисунок 13).

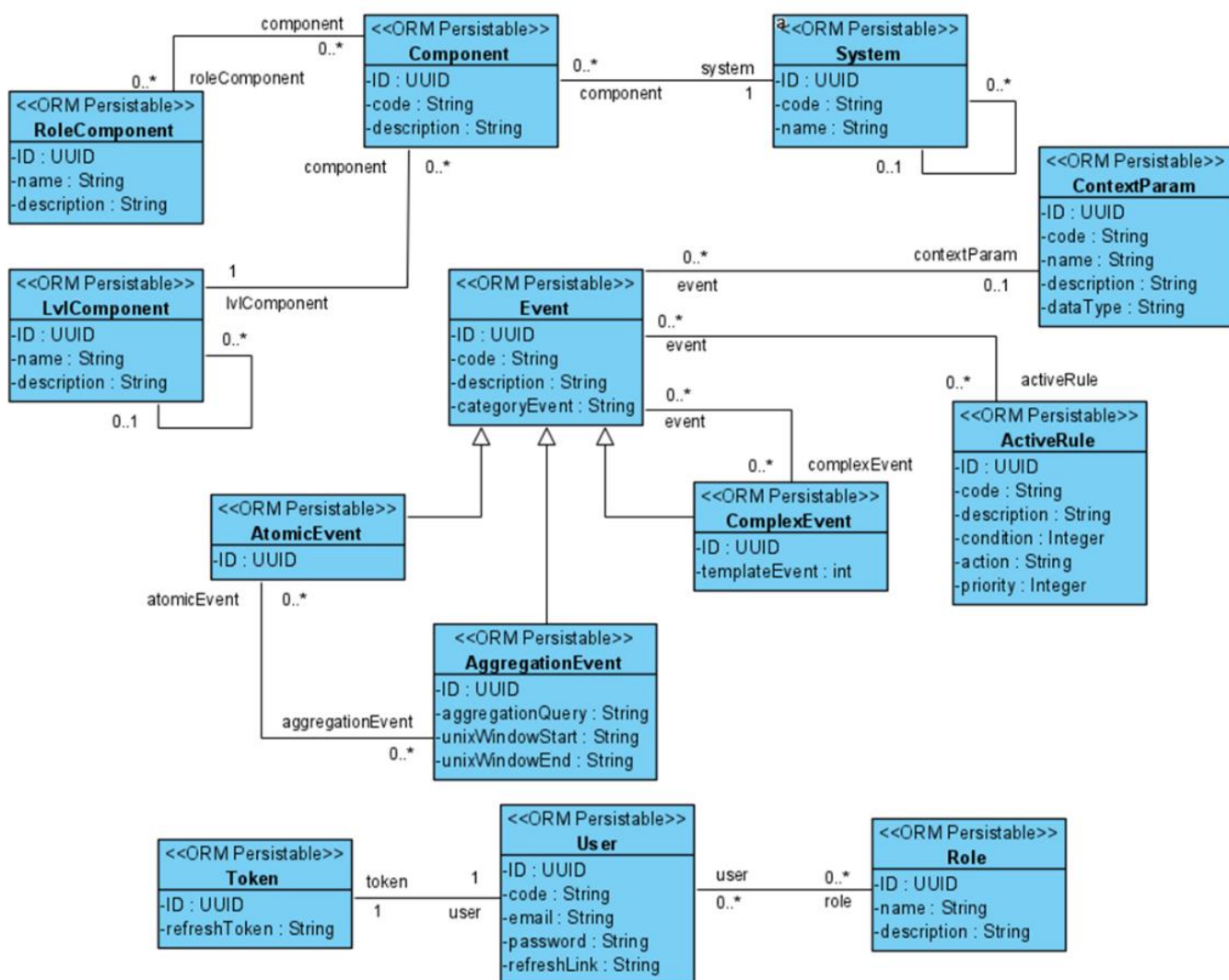


Рисунок 13 – Представление единой метамодели в рамках объектной метамодели данных

Спецификации основных объектов данной модели представлены в таблице 1.

Таблица 1 – Спецификации объектов модели

Объект	Описание
System	Содержит объекты систем наблюдения и систем контроля, а также их подсистемы
Component	Содержит компоненты наблюдения и компоненты контроля
Role Component	Содержит метаданные, отвечающие за роль компонента в системе
Lvl Component	Содержит метаданные, отвечающие за уровень компонента
Event	Содержит общую информацию о всех событиях в системе
Atomic Event	Содержит объекты поступающих элементарных событий
Complex Event	Содержит объекты сложных событий, которые были созданы в системе
Aggregate Event	Содержит объекты агрегированных событий
Context Param	Содержит контекст события
Active Rule	Содержит объекты активных правил и условий их активации
User	Содержит метаданные пользователя
Role	Содержит метаданные ролей пользователя, которые позволяют производить соответствующие действия в системе
Token	Содержит специальные токены пользователей, необходимые для предоставления доступа к ресурсам

Центральный репозиторий был развернут при помощи средств PostgreSQL. Благодаря данной СУБД обеспечивается надежность хранения метаданных и быстрая скорость обмена большим набором информации. Он поддерживает расширенные типы данных, такие как массивы, hstore (хранилище ключей и значений) и JSON, что делает его хорошо подходящим для хранения метаданных под сложные и агрегированные события.

Диаграмма репозитория метаданных, созданного при помощи СУБД PostgreSQL, проиллюстрирована на рисунке 14.

Репозиторий состоит из 19 сущностей. Планируется дальнейшее расширение репозитория для улучшения внутренних процессов.

2.2 Алгоритмы первичной валидации метаданных при добавлении их в центральный репозиторий реагирующей ИУС

Необходимо специфицировать алгоритмы первичной валидации метаданных при попытке их добавления в центральный репозиторий. Необходим алгоритм, который будет автоматически запускаться и делать первичную проверку, которая включает в себя проверку прав пользователя на возможность управления объектами и анализ корректно заданных элементов метаданных. Диаграмма состояний первичной проверки изображена на рисунке 15. За основу взят сценарий создания сложного события.

Описанная первоначальная проверка состоит из трёх ключевых шагов: подтверждение авторизации, проверка прав доступа в зависимости от роли и анализ корректно заданных элементов метаданных.

Данный анализ выполняется при помощи главного компонента ИУС и способен выявить следующие аномалии:

- несоответствия между экземплярами метаданных,
- избыточность,
- смысловые несоответствия.

Проверка на несоответствия между экземплярами метаданных необходима для того, чтобы убедиться, что все объекты метаданных заданы правильно и логически соответствуют друг другу. Примером неправильного сочетания между событием и условием может служить попытка связать условие активации, предполагающее наличие воды, с событием от датчика, измеряющего температуру. Такое несоответствие между контекстом события (температурой) и условием для его активации (наличием воды) является явной аномалией. Это несоответствие легко обнаруживается на этапе конструирования правил, предотвращая включение подобных некорректных правил в систему.

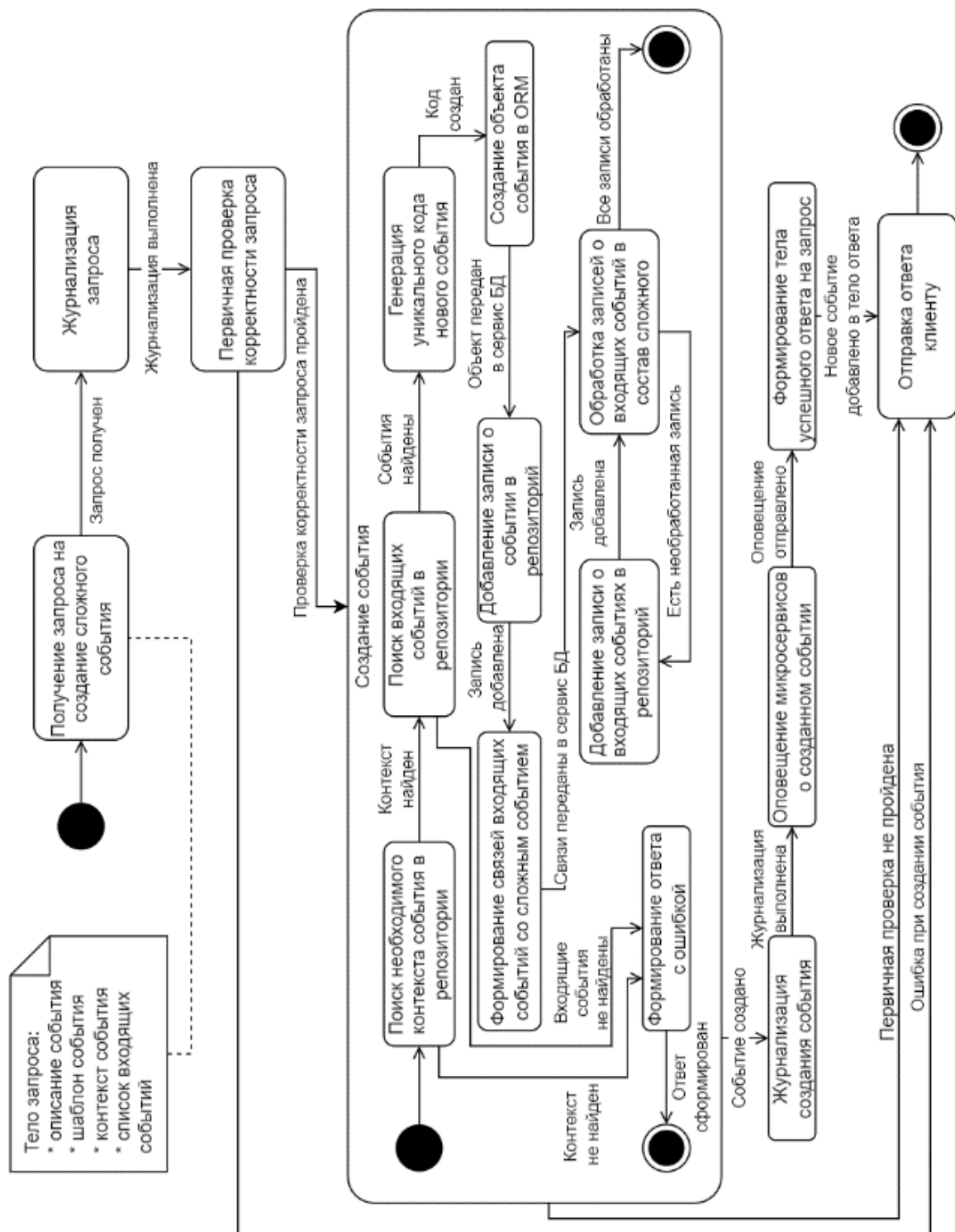


Рисунок 15–Диаграмма состояний создания сложного события с первичной проверкой

Под избыточностью в ИУС понимается излишнее дублирование событий, правил или иных классов метаданных. Эта проблема может привести не только к проблемам повышения ресурсоёмкости и снижения эффективной обработки событий, но и к аномалиям в сложных событиях. Например, в событиях с агрегацией данных крайне важно не допустить дубликатов элементарных событий. К характеристикам избыточности относятся одинаковая категория, контекст исполнения и источник для базовых событий, а также дополнительные параметры для более сложных или агрегированных событий, например, шаблон исполнения для сложных событий или критерии выборки для агрегирования. Определение избыточных событий может происходить с помощью специализированных алгоритмов на этапах создания, редактирования событий и проектирования системы.

Смысловая несогласованность ИУС происходит, когда установленные правила ведут к последовательности действий, которые логически противоречат друг другу или неэффективны в достижении целей системы. Это противоречие может существенно снижать эффективность системы, ведя к пустой трате ресурсов или к действиям, которые взаимно нейтрализуют результаты друг друга. Рассмотрим пример с действиями «открыть окна» и «включить обогреватель». Эти действия могут быть безопасны, если использовать их независимо друг от друга, но при использовании в общем контексте, например для достижения цели поддержания тепла в помещении, они могут оказаться противоречивыми. При неверно заданных условиях активации открытие окон приведет к потере тепла, в то время как включение обогревателя будет направлено на его генерацию. Если диапазон значений в условии будут совпадать, то оба действия будут выполняться одновременно. Это приведёт к неэффективному использованию ресурсов системы и к тупиковой ситуации. Зачастую определение противоречивых действий требует глубокого понимания контекста и условий, в которых эти действия выполняются. Это означает, что для выявления смысловой несогласованности

необходимо анализировать не только сами правила, но и информацию о внешней среде и текущих условиях.

Можно сделать вывод, что подобные аномалии могут существенно снижать эффективность и надежность ИУС. Каждая из рассмотренных аномалий в метаданных имеет потенциал привести к серьезным сбоям или непреднамеренным реагирующим воздействиям.

Данные средства анализа позволяют своевременно выявлять и устранять потенциальные проблемы, тем самым обеспечивая стабильность и надёжность работы ИУС.

2.3 Распределенная архитектура метаданных реагирующей ИУС потоковой обработки событий и исполнения активных правил

Система управления потоковой обработкой событий и выполнения активных правил, использующая событийно-ориентированный подход, построена на архитектуре, включающей множество микросервисов (рисунок 16).

Такая архитектура имеет множество преимуществ, среди которых наиболее значимыми являются высокая степень масштабируемости, отказоустойчивость, работа в режиме реального времени и высокая доступность.

ИУС получает данные через очередь брокера сообщений, который собирает и распределяет события в соответствии с установленными правилами. Благодаря такой архитектуре повышается автономность каждого элемента системы, а логика независимых компонентов отделяется друг от друга. Этот подход также позволяет создавать микросервисы на разных языках программирования, взаимодействие между которыми осуществляется с помощью общедоступных протоколов и средств. Микросервисы могут независимо друг от друга масштабироваться и обновляться. Сбой или выход из строя одного компонента не окажет значительного влияния на работу всей ИУС.

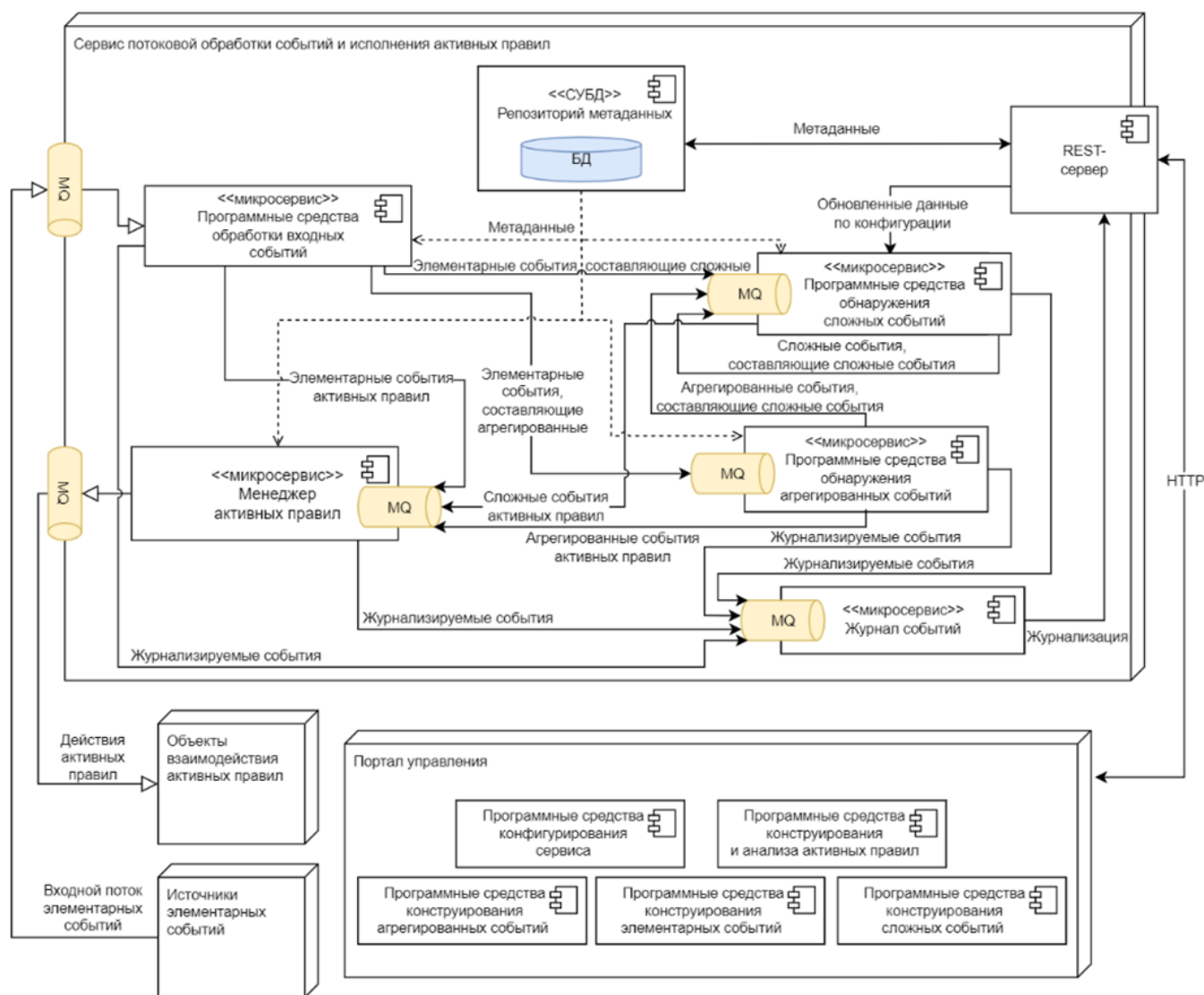


Рисунок 16 – Архитектура ИУС потоковой обработки событий

В микросервисных архитектурах проблема быстрой обработке данных и высокой пропускной способности имеет первостепенное значение. Взаимодействие с центральным репозиторием метаданных будет вызывать большую задержку выполнения операций и может негативно сказаться на скорости и корректности обработки поступающих событий.

Для решения этой проблемы на микросервисах реализуют паттерн «База данных на сервис» (Database per Service) [17], основная суть которого – реализация локального репозитория рядом с каждым микросервисом. Часто для таких репозиториев применяются резидентные (in-memory) СУБД, которые также служат для кэширования микросервисов. Локальный репозиторий позволяет значительно сократить время получения метаданных, необходимых для обработки событий.

Однако создание такой распределённой архитектуры может стать серьёзной проблемой при тиражировании и синхронизации метаданных. В таких системах поддержание согласованного состояния во всех микросервисах является важной задачей, для реализации которой требуется разработать специальные алгоритмы и программные средства.

2.4 Обоснование выбора подходов синхронизации метаданных

Из-за распределенного характера архитектуры метаданных обеспечение согласованности является сложностей задачей. Необходимо найти решение, которое обеспечит синхронизацию метаданных без ухудшения работоспособности ИУС.

2.4.1 Модели согласованности в распределённой архитектуре

Из-за распределенного характера архитектуры обеспечение согласованности данных является сложностей задачей. Существует несколько моделей, которые обеспечивают основу для понимания компромиссов между строгой актуальностью данных и быстродействию системы [18].

Строгая (немедленная согласованность) гарантирует, что любая операция чтения данных извлекает самую последнюю запись. Применимо к микросервисной архитектуре такая модель может быть реализована с помощью методов синхронной репликации, при которых каждая операция записи подтверждается только после того, как она была зафиксирована во всех репликах (рисунок 17).

Эта модель упрощает разработку микросервисов, обеспечивая единое согласованное представление данных в любой момент времени, однако она может существенно повлиять на доступность и быстродействие системы.

Конечная согласованность – это более простая модель, которая допускает временные расхождения в реплицируемых данных, но при этом гарантирует, что все копии данных будут сходиться к одной и той же записи при отсутствии новых обновлений по истечении определенного периода (рисунок 18).



Рисунок 17 – Строгая согласованность

Эта модель обеспечивает более высокую доступность и меньшую задержку, однако требуя механизмов, которые способны работать с устаревшими данными. При использовании данной модели также может потребоваться дополнительная логика для разрешения конфликтов во время работы системы.



Рисунок 18 – Конечная согласованность

Важное отличие между строгой и конечной согласованностями заключается в том, что строгая согласованность может существенно повлиять на производительность: задержки в работе системы и пропускную способность [19].

Строгая согласованность не всегда требуется, и в некоторых случаях может быть достаточно конечной согласованности. Например, предположим, что пользователь сменил иконку или цвет активного правила в пользовательском интерфейсе, а в центре обработки данных произошел сбой. Повторное изменение не является катастрофой, поэтому в этом случае было бы достаточно конечной согласованности. Но если пользователь изменит тело

активного правила, которое отвечает за бизнес-логику обработки события, то конечная согласованность может привести к некорректной работе сервиса.

Еще одной моделью является причинно-следственная согласованность. Это компромисс между строгой и конечной согласованностями, который гарантирует, что причинно-связанные обновления отображаются всеми процессами в одинаковом порядке. Эта модель полезна в сценариях, где логика приложения требует согласованного порядка операций и может допускать работу с устаревшими данными. Причинно-следственная согласованность может быть реализована с помощью временных меток и использования версий для метаданных.

В реальной практике реализуются сразу несколько моделей, каждая под свои требования. Необходимо отталкиваться от требований к быстродействию и гарантий актуальности данных в каждом элементе системы.

2.4.2 Управление версиями с использованием временных меток для синхронизации метаданных

В распределенных системах, особенно при использовании микросервисной архитектуры, управление версиями на основе меток является популярным методом поддержания согласованности между репликами метаданных. Данный подход предполагает присвоение уникального значения каждому обновлению метаданных.

Существует две популярных стратегии назначения меток: использование системного времени, в которое произошло обновление метаданных (timestamp), либо использование логических часов (монотонно увеличивающегося счетчика). В базовом представлении реализация первой стратегии в микросервисной архитектуре лучше, поскольку с высокой долей вероятности и без дополнительной логики обеспечивается уникальность метки, практически исключены те случаи, когда несколько источников одновременно задают метку.

При внесении изменений в метаданные предполагается сценарий, который включает 4 основных шага:

1) ответственный элемент системы присваивает обновленной версии метаданных новую временную метку, гарантируя, что она больше, чем любая предыдущая временная метка,

2) обновленные метаданные и связанная с ними временная метка распространяются на все микросервисы,

3) микросервис получает обновление и сравнивает временную метку обновления с временной меткой своих текущих метаданных,

4) если набора метаданных с указанной временной меткой не содержится на микросервисе, то происходит добавление этого набора в локальный репозиторий.

Наличие сразу нескольких наборов метаданных с разными временными метками необходимо для корректной обработки уже «поступивших» в систему событий на момент редактирования правил. Возможен случай, когда во время обработки события произошло изменение сценария обработки. Событие, которое поступило до обновления, а об этом свидетельствует временная метка, будет обработано в соответствии с прежним набором метаданных.

Управление версиями на основе временных меток является полезным методом синхронизации обновлений метаданных между микросервисами, обеспечивая баланс между согласованностью, доступностью и производительностью.

2.4.3 Обработка конфликтов при тиражировании и синхронизации метаданных

В распределённой архитектуре конфликты при тиражировании и синхронизации метаданных могут возникать из-за различных аномалий, поэтому необходимо иметь механизмы для нахождения и разрешения конфликтов.

Для корректной работы приложения конфликт необходимо обнаружить и разрешить сразу в процессе тиражирования метаданных на различные микросервисы. Как только конфликт обнаружен, система должна

использовать стратегию для его разрешения. Существует несколько подходов к разрешению конфликтов [20-22].

Глобальная блокировка – это самый простой способ для соблюдения строгой согласованности данных, но крайне неэффективный из-за большого количества задержек. Суть подхода заключается в том, что если произошло какое-либо изменение, то процессы обработки приостанавливаются до момента получения актуального набора данных. Также блокируется возможность обновления данных со стороны. Сам по себе подход нарушает главный принцип обработки данных в реальном времени, так как в определенный момент система простаивает и не обрабатывает данные.

Побеждает последняя запись (LWW) – это стратегия, которая принимает в качестве «истинной» только самую последнюю запись, которой соответствует ближайшая временная метка или последний номер версии набора данных. Несмотря на простоту, LWW может привести к потере обновлений, а также он не сохраняет предыдущие наборы данных, которые могут понадобиться для обработки «опаздывающих» событий.

Управление многовариантными записями (MVCC) – это стратегия, которая, в контексте обработки событий в реальном времени, означает, что локальные репозитории могут хранить несколько версий метаданных, позволяя системе выбирать правильную версию на основе контекста обработки.

Оперативное преобразование (OT) – это стратегия, которая используется в системах, когда один набор данных могут редактировать одновременно несколько источников. Авторитетный сервер получает сразу все обновления, сравнивая их на наличие конфликтов. В случае возникновения конфликта OT позволяет преобразовывать одновременные обновления таким образом, чтобы их можно было применять в любом порядке и при этом сохранять согласованность.

Бесконфликтный тип реплицируемых данных (CRDT) – относительно новая концепция, в основе которой лежит создание таких типов данных, для

которых не понадобится разрешение конфликтов. Каждое обновление должно быть разработано таким образом, чтобы быть коммутативным, идемпотентным и ассоциативным, гарантируя, что независимо от порядка применения обновлений конечное состояние будет одинаковым.

В рамках разрабатываемого сервиса важным требованием является сохранение версий для обновлений метаданных, быстроедействие и сохранение нескольких актуальных наборов в локальных репозиториях микросервисов. Данным требованиям удовлетворяет MVCC, которая без лишних затрат производительности позволит соблюсти указанные требования.

При реализации тиражирования с разрешением конфликтов необходимо также обеспечить наличие надежного канала связи. Необходимо создать механизм согласования и отката (rollback) обновлений. Механизм согласования позволит каждому микросервису получать актуальный набор данных от центрального репозитория. Механизмы отката позволят, в случае конфликта, вернуть прежнее состояние данных.

В данном случае могут использоваться транзакции для управления обновлениями метаданных. Транзакции могут гарантировать, что все микросервисы видят согласованное представление метаданных, либо фиксируя обновление во всех репликах, либо откатывая его при обнаружении конфликта.

2.5 Алгоритм тиражирования и синхронизации метаданных

Алгоритмы тиражирования и синхронизации необходимы для своевременного оснащения микросервисов метаданными, необходимыми для работы. Важным аспектом является то, что обновление метаданных должно происходить без какого-либо простоя системы. Должна присутствовать гарантия, что поступившее в поток событие будет обработано по единственному набору метаданных.

2.5.1 Алгоритм тиражирования метаданных

Алгоритм тиражирования отвечает непосредственно за саму доставку наборов метаданных в локальные репозитории микросервисов. Для основы

понимания алгоритма необходимо сперва рассмотреть его структурные элементы.

Главный элемент ИУС. Главный компонент, которые отвечает за хранение, управление и предоставление метаданных всем заинтересованным микросервисам. Данный компонент инициирует процессы тиражирования, а также выполняет функции по восстановлению состояния системы в случаях сбоя. Компонент имеет непосредственной взаимодействие с центральным репозиторием метаданных.

Программная служба тиражирования с возможностью отслеживания состояний метаданных. Данное программное обеспечение поддерживает постоянный канал связи с микросервисами и предоставляет им возможность подписываться на события об обновлениях.

Механизм синхронизации. Данный функциональный элемент необходим для гарантии того, что все микросервисы в системе будет обрабатывать определённый экземпляр события в рамках единого состояния метаданных.

Механизм кеширования и локальные репозитории. Для уменьшения зависимости от централизованного репозитория и снижения задержек при обращении к метаданным, микросервисы могут использовать локальное кеширование актуальной информации.

Система контроля версий метаданных. Для обеспечения совместимости и возможности отката к предыдущим версиям метаданных, необходима система управления версиями. Это позволяет микросервисам работать с различными версиями метаданных без риска нарушения работы ИУС. Поступившее на вход в микросервис событие должно содержать уникальную метку – номер версии метаданных, с которой необходимо обрабатывать это событие.

Данные структурные элементы требуют тщательного планирования, что включает в себя не только технические аспекты, такие как выбор технологий и проектирование интерфейсов, но и управленческие, например, разработку

процессов управления изменениями и контроль за версиями, включая обогащение событий текущей версией метаданных.

Рассмотрим концептуальную схему алгоритма тиражирования (рисунок 19). Данный алгоритм можно разделить на несколько этапов.

Этап подготовки метаданных. Главный элемент ИУС делает запрос в центральный репозиторий и получает все текущие метаданные с указанием версии. Этот набор служит отправной точкой для последующей идентификации изменений.

Этап получение первичного набора метаданных микросервисами. Первичным называется тот набор метаданных, который микросервис получает при первом запуске или восстановлении работы, в случае перезагрузки с обнулением кэша. Далее главный элемент ИУС формирует уникальные небольшие наборы метаданных, которые необходимы соответствующим микросервисам для их работы. В этот момент микросервис должен установить связь с главным компонентом, подписаться на обновления и получить первичный набор.

Способ получения первичного набора отличается от вида устанавливаемой связи. Если соединение является двусторонним, как при использовании WebSocket, то набор данных может быть получен после успешного установления соединения. Однако, такой тип связи часто требует дополнительных затрат и времени при запуске микросервиса. В случае однонаправленного соединения, например, в модели публикации/подписки, необходимо создать дополнительный механизм для отправки первичного набора метаданных параллельно процессу установления соединения, например, через HTTP-протокол. В таком случае возрастают требования и трудозатраты на реализацию данного подхода, но в конечном счёте это положительно повлияет на скорость запуска микросервиса и готовности к процессам обработки событий.

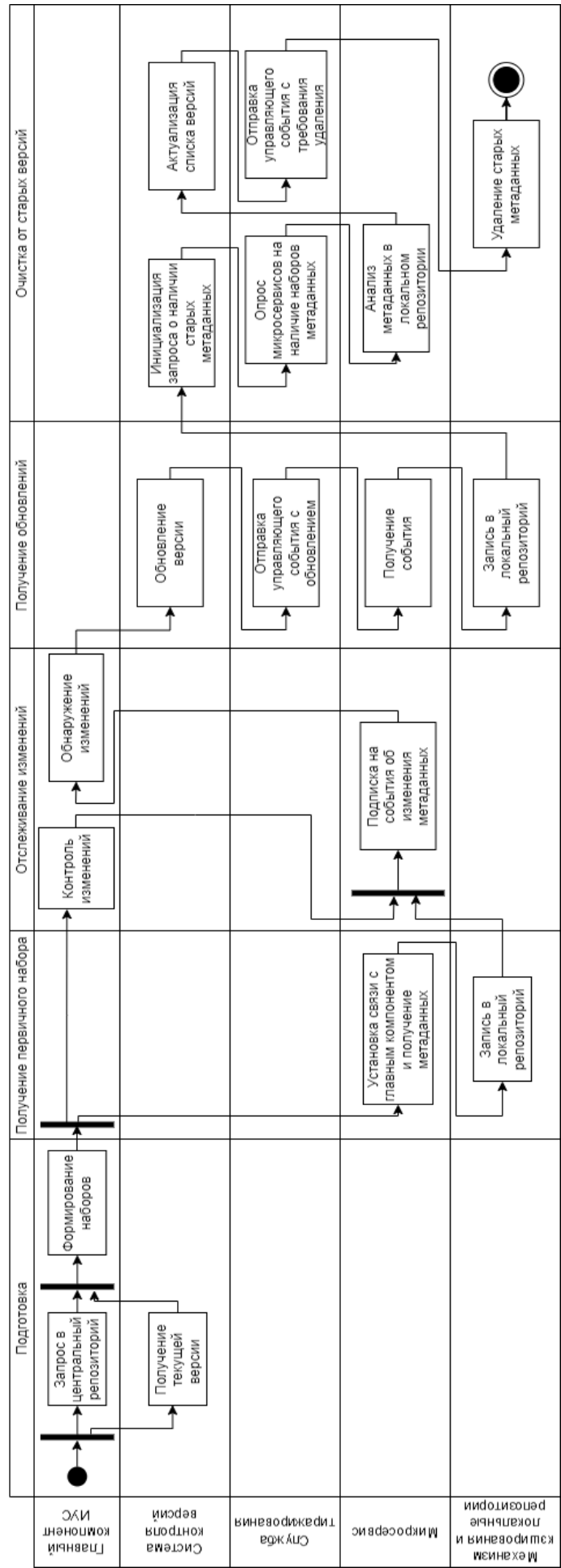


Рисунок 19 – Процесс тиражирования метаданных

Этап отслеживания изменений и оповещения об обновлениях. Так как все пользовательские процессы управления метаданными протекают через главный компонент ИУС, то он, в свою очередь, способен отследить изменения в текущих наборах метаданных. Это достигается с помощью различных методов и подходов, таких как опрос, уведомления, управляющие события и потоковая передача данных в реальном времени. При обнаружении изменения система контроля версиями, в зависимости от выбранной стратегии, присваивает метаданным новую временную метку или помечает изменения как критически важные. Затем главный компонент формирует наборы метаданных, специфичные для каждого микросервиса, с учётом новых изменений, которые публикуются в соответствующие каналы (топики), предназначенные для каждого микросервиса.

Этап получения обновлений. На данном этапе каждый микросервис получает соответствующие и релевантные метаданные, адаптированные к его конкретным требованиям или контексту. Актуальные метаданные вместе с временной меткой кэшируются в локальные репозитории микросервисов.

Этап очистки от старых метаданных. Данный этап необходим для предотвращения проблемы утечки памяти, когда нерелевантные наборы метаданных остаются в локальных репозиториях. Система контроля версий производит опрос среди элементов ИУС и вычисляет с какими версиями метаданных обрабатываются события на текущий момент. В случае если в системе отсутствуют события, в контексте которых содержится старая версия метаданных, то система помечает этот набор как неактуальный. Далее главный компонент ИУС отправляет соответствующее сообщение каждому микросервису с необходимостью удаления данного набора. Сам набор остается в архивном разделе репозитория на заранее заданный промежуток времени, что необходимо для обработки критически важных ситуаций и аудита со стороны администрации.

Во время работы микросервисы считывают метаданные непосредственно из своего локального репозитория. Информацию о том,

какие наборы необходимо использовать, микросервисы получают из контекста обрабатываемого события, в котором указана необходимая временная метка.

2.5.2 Алгоритм синхронизации метаданных

Алгоритм синхронизации необходим для осуществления гарантии того, что все микросервисы получили необходимые для обработки событий наборы метаданных. Когда в процессе тиражирования метаданных в распределённой микросервисной архитектуре ошибки не произошли, алгоритм синхронизации должен уведомить об этом главный компонент ИУС и систему контроля версии метаданных. Рассмотрим схему работы алгоритма (рисунок 20).

В этом контексте алгоритм синхронизации выступает как механизм, гарантирующий консистентность и актуальность данных во всех микросервисах.

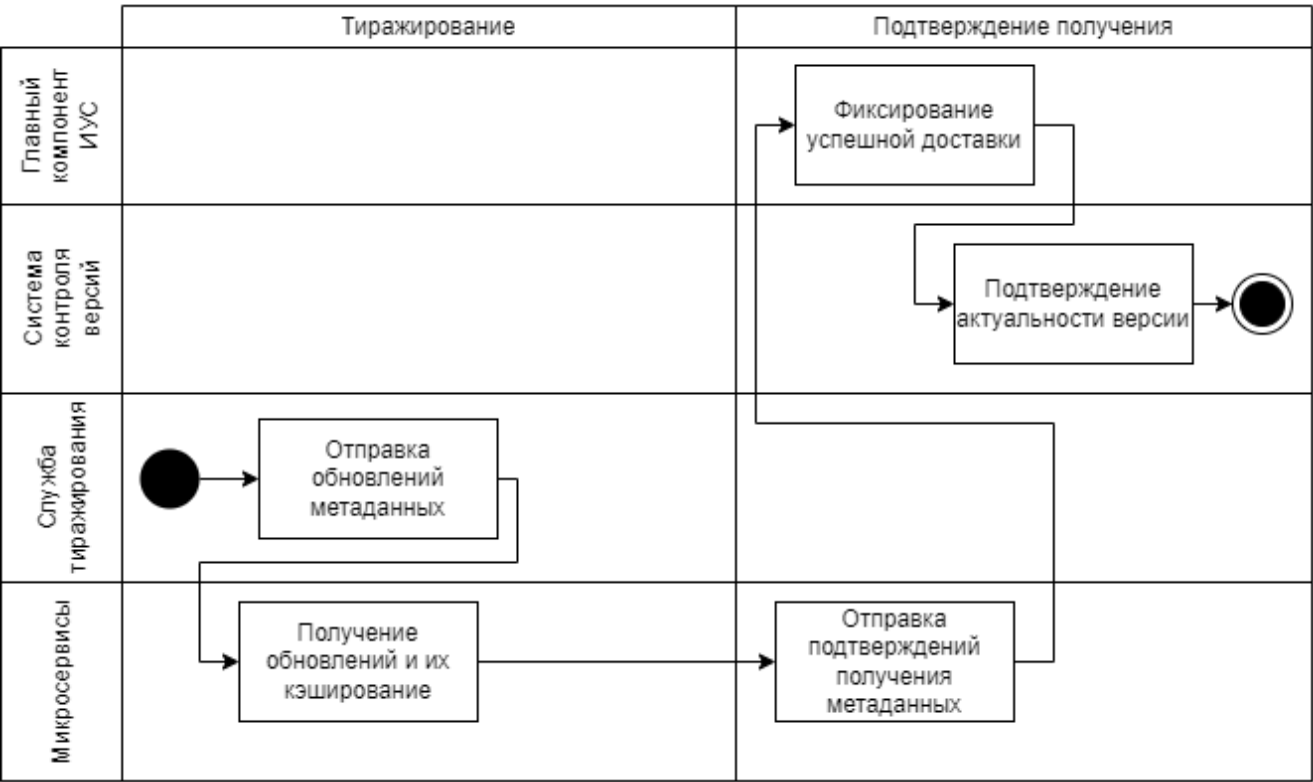


Рисунок 20 – Процесс синхронизации при успешном тиражировании

Архитектура системы имеет распределённых характер, поэтому при потоковой обработке событий, вследствие высокой нагрузки на ИУС, могут

происходить различные описанные ранее аномалии в процессе тиражирования метаданных.

Важным аспектом алгоритма является способность системы продолжать работу при частичных сбоях без необходимости перезагрузки микросервисов. Это достигается за счет разработанного алгоритма тиражирования метаданных между различными узлами системы и возможностью переключения на резервные копии данных в случае выхода возникновения проблем.

Алгоритм синхронизации включает механизм отката (Rollback) для устранения любых несоответствий, которые могут возникнуть в процессе обновления метаданных. При сбое или ошибке при доставке метаданных процесс отката направлен на отмену или возврат любых изменений или модификаций состояния.

В отличие от баз данных с полной поддержкой транзакций, событийно-ориентированные системы не имеют встроенных механизмов автоматического отката. Они сосредоточены на обеспечении высокопроизводительной доставки данных и изначально не поддерживают требования ACID к транзакциям. Процессы отката необходимо реализовывать на прикладном уровне. При разработке архитектуры был придуман и реализован данный процесс (рисунок 21).

Рассмотрим основные этапы процесса отката.

Обнаружение аномалий или ошибок. В распределенной ИУС необходимо непрерывно мониторить систему на предмет аномалий или ошибок, которые могут указывать на некорректное применение обновлений метаданных. Это может быть реализовано через системы логирования и мониторинга, которые запускают автоматическое восстановление или отправляют соответствующее предупреждение администратору/пользователю сервиса.

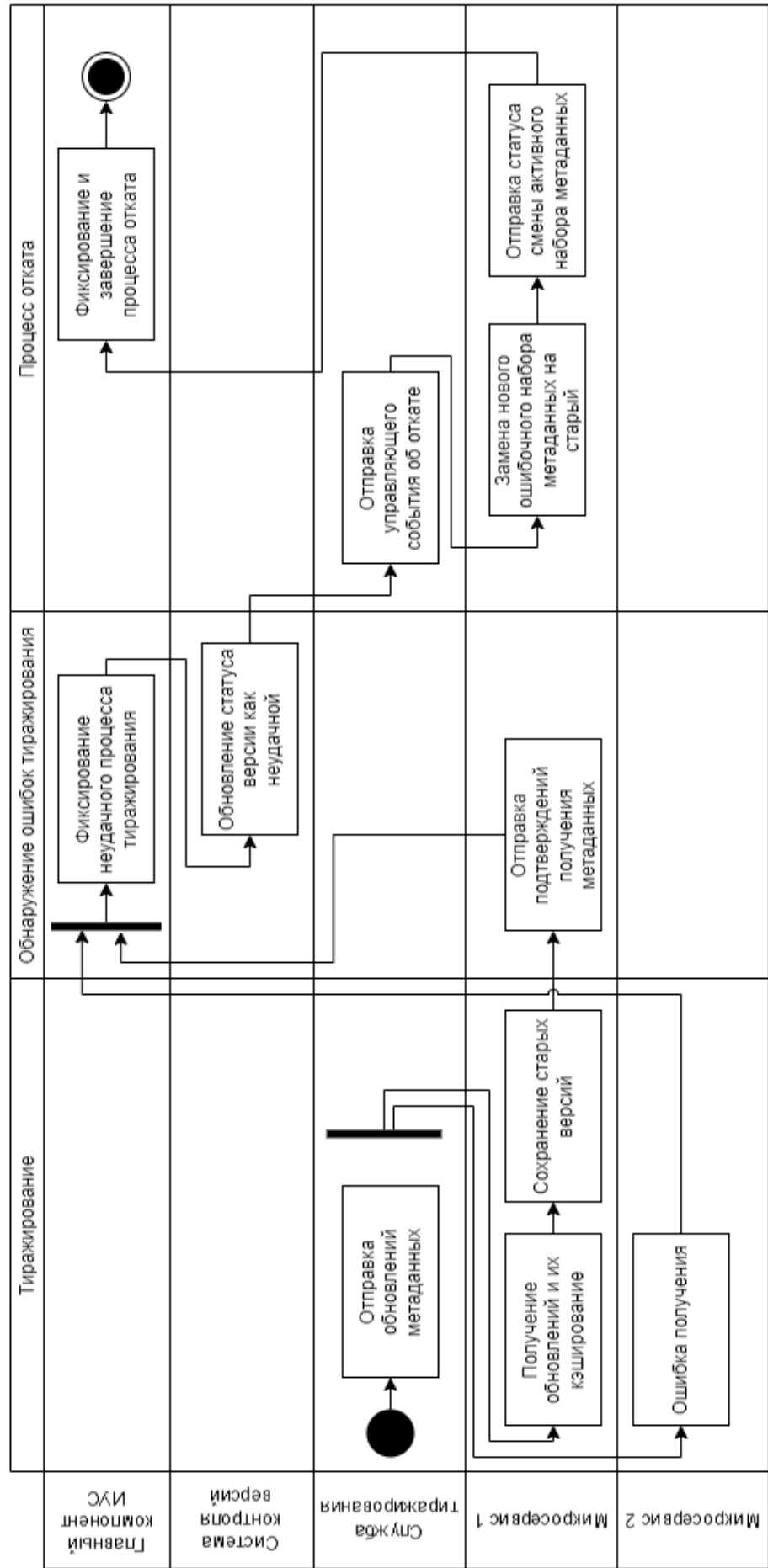


Рисунок 21 – Процесс отката до предыдущего состояния метаданных

Инициирование отката и восстановление предыдущего состояния. Как только аномалия обнаружена, и принято решение, что наиболее эффективным решением будет возврат к предыдущей версии метаданных, начинается процесс отката. Главный компонент ИУС сообщает всем микросервисам, что новое обновление метаданных не было корректно зафиксировано, поэтому эти метаданные удаляются из локальных репозитория. Система контроля версий, отслеживая процессы тиражирования, понимает, что обновление не было успешно завершено, поэтому новая метка версии метаданных не будет присвоена в контекст поступающего события.

Синхронизация отката с микросервисами. После восстановления предыдущего состояния метаданных главный компонент ИУС опрашивает все микросервисы и гарантирует, что процесс отката прошёл успешно.

Анализ и устранение причин аномалий. После успешного отката и восстановления стабильного состояния ИУС предоставляет необходимые метрики и зависимости, которые привели к возникновению ошибки. Основываясь на этом анализе, разрабатываются и внедряются корректирующие меры для предотвращения повторения подобных инцидентов в будущем.

Процесс отката является важной частью алгоритма синхронизации и позволяет управлять рисками в микросервисной распределённой архитектуре реагирующей ИУС. Он обеспечивает возможность быстро реагировать на ошибки и минимизировать их влияние на работу системы. При разработке программного обеспечения важно учитывать, что процесс отката должен выполняться эффективно и не блокировать работу системы.

2.6 Выводы по разделу 2

В данной главе была описана и разработана единая метамодель ИУС, которая включает в себя метамодель внешних систем, событий и активных правил. Представлена объектная модель репозитория метаданных. Описаны особенности использования распределённой архитектуры метаданных, приведены возможные проблемы и их пути решения. Рассмотрены подходы и

алгоритмы синхронизации метаданных, в том числе модели согласованности в распределённой архитектуре и модели, основанные на версионировании данных с использованием временных меток. Были указаны алгоритмы для разрешения проблемных ситуаций при тиражировании метаданных.

3 Программные средства управления метаданными в составе сервиса потоковой обработки событий и исполнения активных правил

В данной главе будут рассмотрены программные средства для реализации описанных ранее механизмов и алгоритмов управления метаданными в реагирующей ИУС. Затрагиваются аспекты архитектуры и функциональные возможности серверных и клиентских программных средств [23]. Рассматриваются средства для взаимодействия между элементами архитектуры, средства для анализа корректности и непротиворечивости метаданных. Для средств тиражирования и синхронизации представлен подход к разработке с применением шаблонов проектирования.

3.1 Назначение, архитектура и функциональные возможности программных средств управления метаданными сервиса потоковой обработки событий и исполнения активных правил

Возникает необходимость в выборе правильных программных средств для организации, обеспечения целостности и тиражирования метаданных. От корректности выбора программных средств зависит быстроедействие, возможность масштабирования и отказоустойчивость системы. Архитектура программного обеспечения для управления метаданными будет включать в себя следующие составляющие: централизованное хранилище метаданных, локальные хранилища метаданных, средства конструирования, интерфейсы взаимодействия, средства кэширования для повышения производительности и средства обмена метаданными.

Выделение локальных репозиторий, помимо одного централизованного, необходимо, чтобы каждый отдельный микросервис быстрее обращался к собственному специфичному набору данных. Также это повысит способность масштабирования системы за счёт увеличения инкапсуляции бизнес-логики каждого микросервиса. Для этой цели будут

использованы такие средства, как PostgreSQL и Redis, в сочетании с такими инструментами, как Sequelize ORM, для доступа к данным и управления ими [24-26].

В качестве клиентского средства конструирования метаданных будет выступать портал управления, который представляет собой пользовательский интерфейс, позволяющий системным администраторам, разработчикам и аналитикам создавать, редактировать и удалять объекты метаданных.

Необходимо разработать собственный набор API-интерфейсов, который необходим для каждого отдельного микросервиса. Также необходимо создать внешний набор API для сторонних систем, которые будут интегрироваться к ДС ИУС.

Также возникает необходимость в реализации отдельного механизма, предназначенного для тиражирования и синхронизации метаданных. Благодаря единому механизму, который будет интегрироваться в каждый микросервис отдельно, можно повысить отказоустойчивость системы и скорость разработки новых микросервисов, которые не будут зависеть от деталей реализации этого механизма. Разработанная технология должна будет самостоятельно определять новые объекты метаданных и изменять существующие с учётом поступивших изменений. Основная задача – это гарантия своевременного тиражирования и синхронизация метаданных без требования к перезагрузки или простоя системы. Помимо прочего, будут отслеживаться различные версии объектов метаданных для обеспечения возможности отката к предыдущим состояниям.

Для оптимизации производительности необходимо будет сделать кэширование часто используемых метаданных. Для этого также будет использоваться Redis, который предоставляет возможность хранения локальной копии метаданных прямо в ОЗУ.

Кроме того, система должна включать в себя комплексные функции аудита, поэтому необходимо записывать каждое действие, выполняемое с метаданными, с указанием подробностей, например, с указанием временной

метки и идентификатором пользователя. Эти функции необходимы для мониторинга безопасности и облегчения выявления любых проблем или нарушений, которые могут возникнуть.

Основное приложение будет функционировать на платформе NodeJS. В качестве основного фреймворка для REST-сервера будет использоваться NestJS. Этот выбор обусловлен несколькими важными преимуществами, которые NestJS предоставляет по умолчанию. Прежде всего, NestJS полностью интегрирован с TypeScript, что обеспечивает строгую типизацию и, таким образом, снижает вероятность возникновения ошибок. Во-вторых, архитектура позволяет использовать модульный подход к разработке, облегчая тем самым разделение приложения на переиспользуемые компоненты, сервисы и модули. В-третьих, NestJS обладает мощной экосистемой и поддерживает множество интеграций, включая те, которые необходимы для микросервисной архитектуры.

Nest имеет встроенную поддержку взаимодействия с SQL и NoSQL базами данных. Для взаимодействия с PostgreSQL будет использоваться SequelizeORM. Данный инструмент обеспечивает абстракцию объектно-реляционного отображения (ORM), что позволит взаимодействовать с базой данных, используя высокоуровневые объектные модели, вместо прямых запросов SQL. Модель в Sequelize содержит информацию о сущности: название, атрибуты, типы данных и прочее. В качестве примера рассмотрим модель одной из сущности центрального репозитория (листинг 2).

Листинг 2 – Модель таблицы Active Rule

```
@Table({ tableName: 'ActiveRule' })
export class ActiveRule extends Model<ActiveRule,
IActiveRuleCreate> {

  @Column({
    type: DataType.UUID,
    defaultValue: DataType.UUIDV4,
    primaryKey: true,
  })
  id: string;

  @Column({
    type: DataType.STRING,
```

```

        unique: true,
        allowNull: false,
    })
    code: string;

    @Column({
        type: DataType.STRING,
        allowNull: false,
    })
    description: string;

    @Column({
        type: DataType.INTEGER,
        defaultValue: 5,
        validate: {
            min: 1,
            max: 10,
        },
    })
    priority: number;

    @Column({
        type: DataType.JSONB,
        defaultValue: {},
    })
    condition: JSON;

    @Column({
        type: DataType.JSONB,
        defaultValue: {},
    })
    action: JSON;

    @BelongsToMany(() => Event, () => ActiveRuleEvent)
    event: Event[];
}

```

Эти модели интегрируются в соответствующий модуль с использованием шаблона программирования «Внедрение зависимостей» (Dependency Injection). Еще одним из ключевых преимуществ SequelizeORM является его система миграций, позволяющая контролировать и версионировать изменения в структуре базы данных.

Стоит подчеркнуть, что REST-сервер не имеет жесткой связи с Redis или PostgreSQL. Для всех компонентов ИУС будут разработаны индивидуальные инструкции для запуска их в изолированных Docker-контейнерах. Кроме того, с помощью инструментов Docker будут настроены механизмы

автоматического перезапуска контейнеров в случае их аварийного завершения. Важно соблюсти порядок перезагрузки и подключение к внутренней сети для корректного взаимодействия.

На основании поставленных требований была разработана архитектура развертывания, которая изображена на рисунке 22.

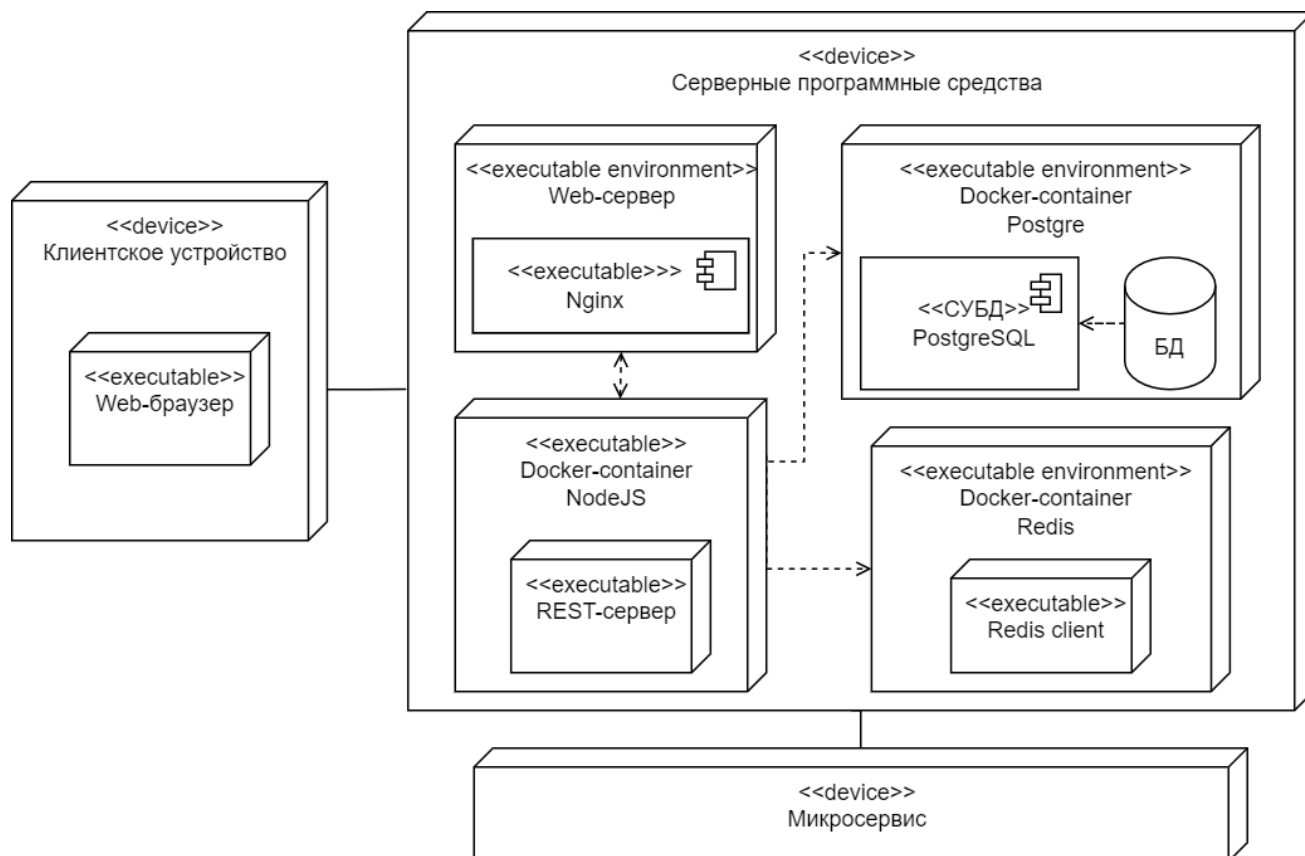


Рисунок 22 – Диаграмма развертывания

Для обеспечения взаимодействия между сервисными программными средствами необходимо организовать внутреннюю защищённую сеть. Центральный репозиторий в этой сети будет иметь проброшенные порты для управления через внешние клиентские СУБД.

В качестве web-сервера будет использоваться Nginx, который отличается низким потреблением процессорных ресурсов и оперативной памяти, а также имеет высокую производительность при большом числе активных запросов от программных средств, занимающихся конструированием метаданных. Для повышения отказоустойчивости Nginx будет размещён в Docker-контейнере с конфигурацией, осуществляемой непосредственно на ОС Ubuntu.

3.2 Программные средства конструирования метаданных

В контексте реагирующей ИУС с применением активных правил под термином конструирование обозначается процесс создания, модификации и организации метаданных. Программные средства конструирования подразделяются на серверные и клиентские.

С серверной стороны бизнес-логику конструирования метаданных инкапсулируют различные сервисы главного компонента – REST-сервера. Для каждого класса метаданных был разработан соответствующий сервис с набором сценариев. Для всех классов предусмотрены базовые операции, такие как создание, изменение, удаление и чтение. Кроме того, некоторые классы имеют уникальные сценарии, которые подробно описаны в таблице 2.

Таблица 2 – Дополнительные сценарии управления метаданными

Сервис	Сценарий	Пояснение
1	2	3
Active Rule	Bind Event	Назначение события активному правилу
Active Rule	Unbind Event	Отсоединение события от активного правила
Active Rule	Find Events	Поиск присоединенных событий по всем типам связи
Component	Add Role	Назначение роли компоненту
Component	Remove Role	Удаление роли у компонента
Component	Add Lvl	Назначение уровня компоненту
Component	Remove Lvl	Удаление уровня у компонента
Component	Add System	Добавление компонента в систему
Component	Remove System	Отсоединение компонента от системы
Event	Add Atomic To Aggregation	Добавление ассоциации элементарного события с агрегированным
Event	Remove Atomic To Aggregation	Удаление ассоциации элементарного события с агрегированным
Event	Add Inbound	Добавление ассоциации между элементарным событием и сложным
Event	Remove Inbound	Удаление ассоциации между элементарным событием и сложным
System	Attach	Назначение дочерней системы родительской в соответствующей иерархии
System	Detach	Удаление дочерней системы из родительской в соответствующей иерархии

Одним из немаловажных преимуществ в созданной архитектуре REST-сервера является просто внедрение зависимостей (Dependency injection). Это означает, что между собой сервисы могут взаимодействовать без какой-либо

жёсткой привязки друг к другу. В качестве примера рассмотрим сценарий конструирования ассоциаций между активным правилом и событием (рисунок 23).

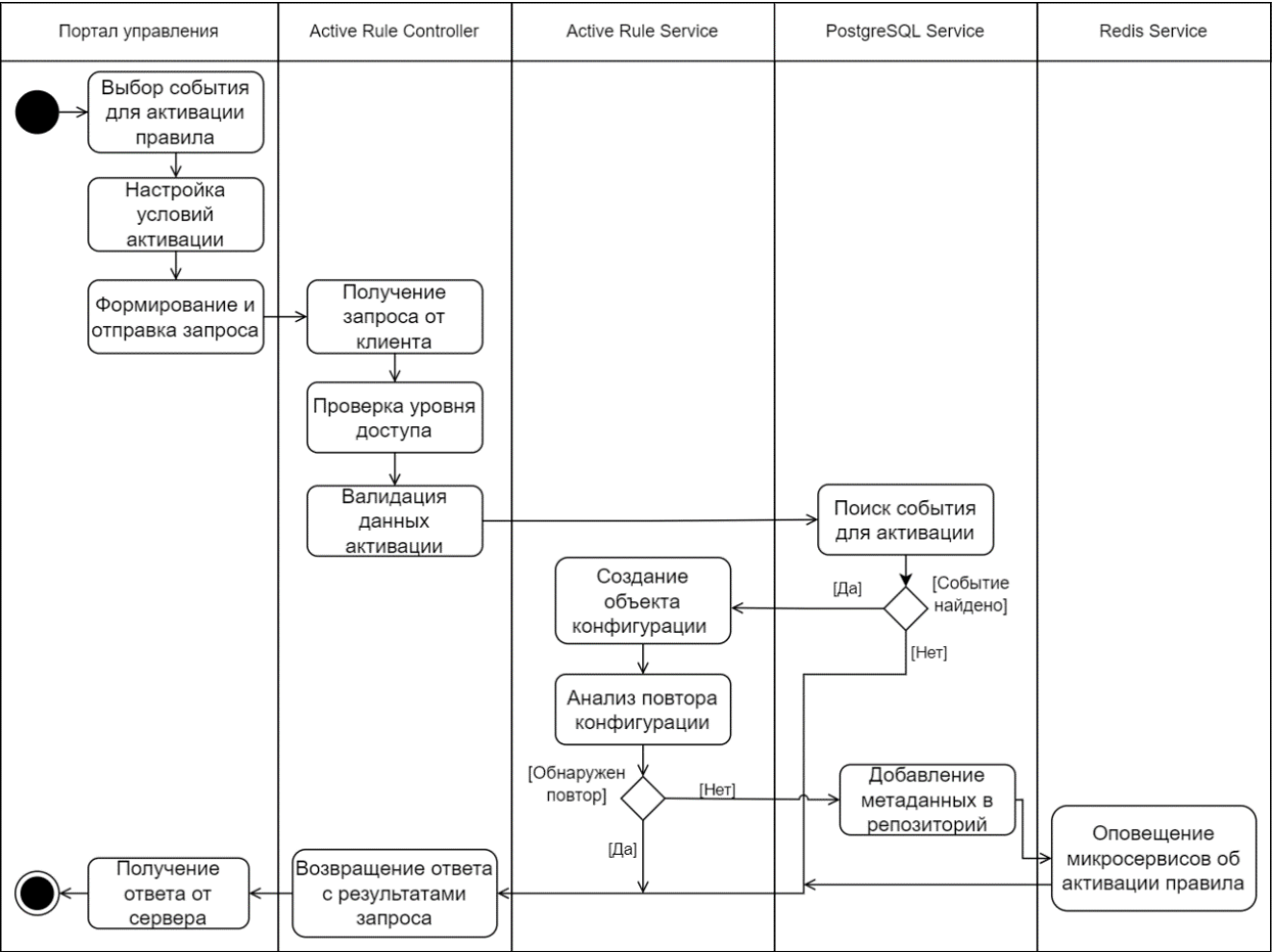


Рисунок 23 – Диаграмма последовательности в сценарии привязки активного правила к событию

Запрос на создание ассоциаций активного правила с событием отправляется клиентским приложением и обрабатывается специализированным контроллером. Этот контроллер взаимодействует с необходимыми сервисами для выполнения бизнес-логики: сервисом активных правил, сервисом работы с репозиторием и сервисом взаимодействия с микросервисами. Затем микросервисы получают уведомление о внесённых изменениях в метаданные. Серверные средства обладают множеством контроллеров для взаимодействия с клиентским приложением. Каждый контроллер обрабатывает свои тематические точки доступа.

Перечень основных контроллеров:

- User, отвечающий за данные пользователей и их роли,
- System, отвечающий за управление метаданными наблюдаемых и управляемых систем,
- Component, отвечающий за объекты компонентов системы, их роли и уровни,
- JWT Auth and Registration, отвечающий за авторизацию и регистрацию новых пользователей,
- Event, отвечающий за метаданных событий и их внутреннего контекста,
- Active Rule, отвечающий за метаданные активных правил.

Документация была создана с использованием инструментов Swagger. В ней представлены все API для взаимодействия с REST-сервером через протокол HTTP(S), а также указаны параметры, необходимые для запросов, и возможные варианты ответов. Пример этой документации показан на рисунке 24.



Рисунок 24 – Точки доступа для работы с событиями

У каждого элемента в документации имеется более подробное описание.

На примере работы с точкой доступа `/events/{id}` для обновления событий (рисунок 25) можно увидеть:

- запрос использует метод PATCH, который указывает на возможность частичного обновления данных,
- идентификация объекта типа event происходит с помощью динамического параметра,
- схема возможных полей для обновления объекта (DTO) с указанием типов данных,
- возможные ответы сервера.



Рисунок 25 – Подробное описание API /events/{id}

Вся документация доступна по адресу сервера с точкой доступа /api. Более того, Swagger на этой странице позволяет отправлять сами запросы на сервер, что упрощает процесс разработки клиентского приложения.

В качестве основного клиентского приложения, которое взаимодействует с описанным REST-сервером, выступает портал управления. Он был разработан по технологии одностраничного приложения (SPA), что повышает удобство работы пользователей, предлагая плавную навигацию и взаимодействие без необходимости перезагрузки страницы.

Для визуального интерфейса были разработаны собственные компоненты на React [27]. Использование React облегчает создание динамичного и интерактивного интерфейса для пользователей, благодаря компонентам, которые можно повторно использовать. Такой подход не только

ускоряет процесс разработки, но и обеспечивает согласованность во всем портале, обеспечивая единый внешний вид. Например, страница с выбором системы для взаимодействия показана на рисунке 26.

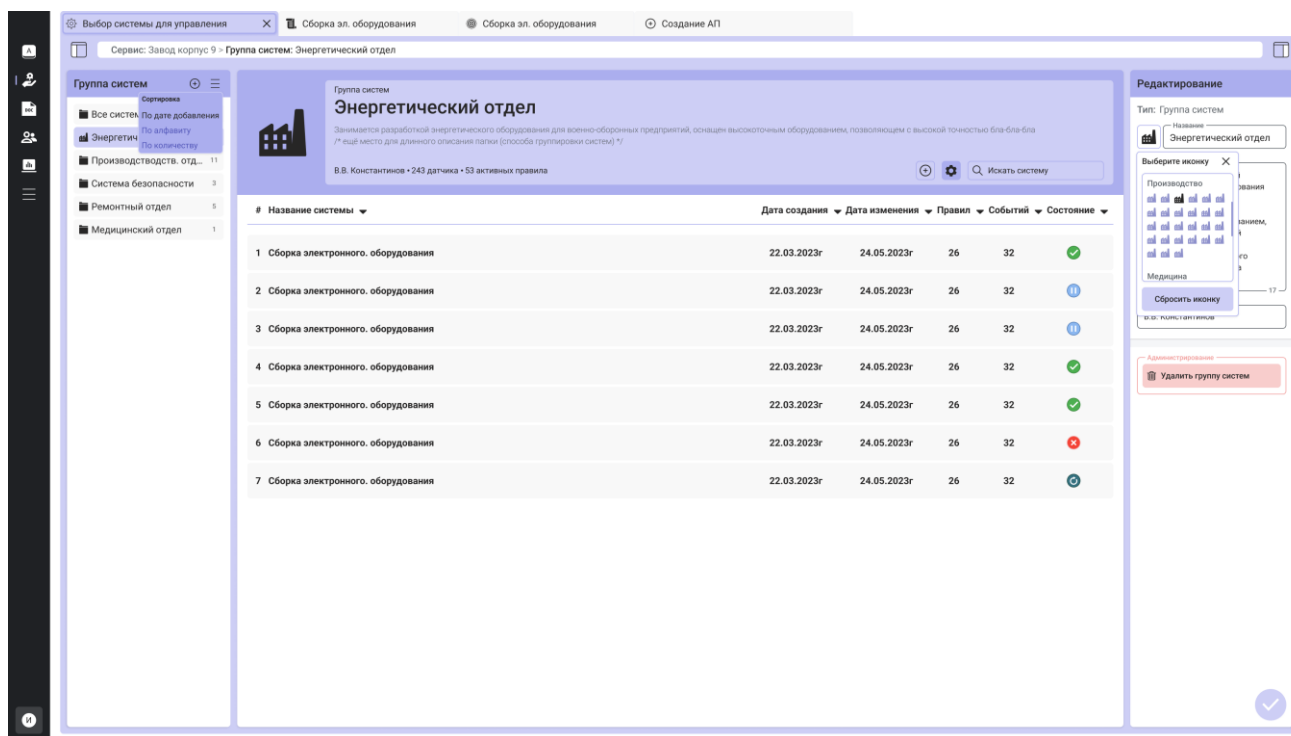


Рисунок 26 – Страница с выбором системы

После выбора системы пользователь попадает на страницу управления элементами системы для конструирования под них активных правил (рисунок 27).

Для управления состоянием, кэшированием и оптимизацией REST запросов используется Redux Toolkit [28]. Кэшируя часто используемые данные, Redux сводит к минимуму необходимость в избыточных вызовах API, тем самым сокращая сетевой трафик и повышая производительность портала.

Также на портале управления используется NoCode-подход при конструировании различных правил и событий [29-30]. Этот подход позволяет пользователям создавать, настраивать и управлять метаданными без необходимости традиционного программирования. Отпадает необходимость к технической реализации систем, тем самым повышается акцент на бизнес-процессах.

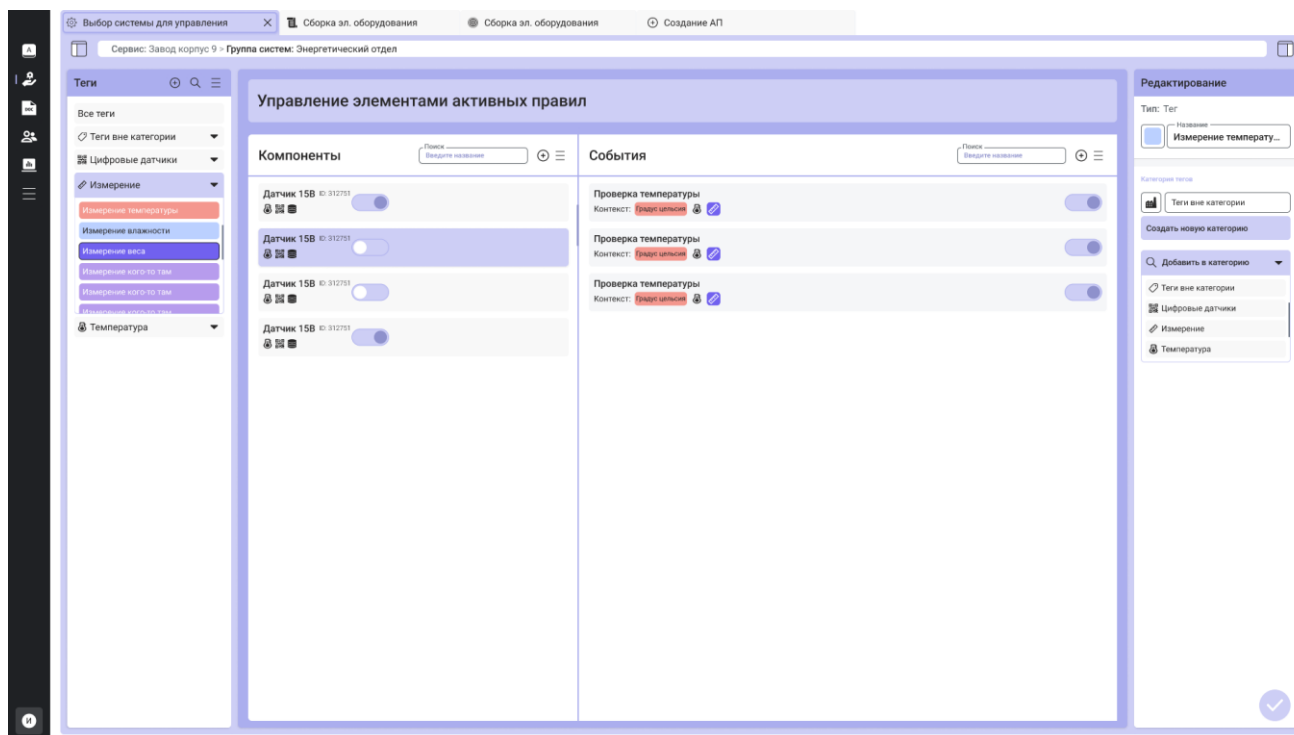


Рисунок 27 – Выбор датчика для создания активного правила

Пример конструирования активного правила с использованием NoCode-подхода показан на рисунке 28.

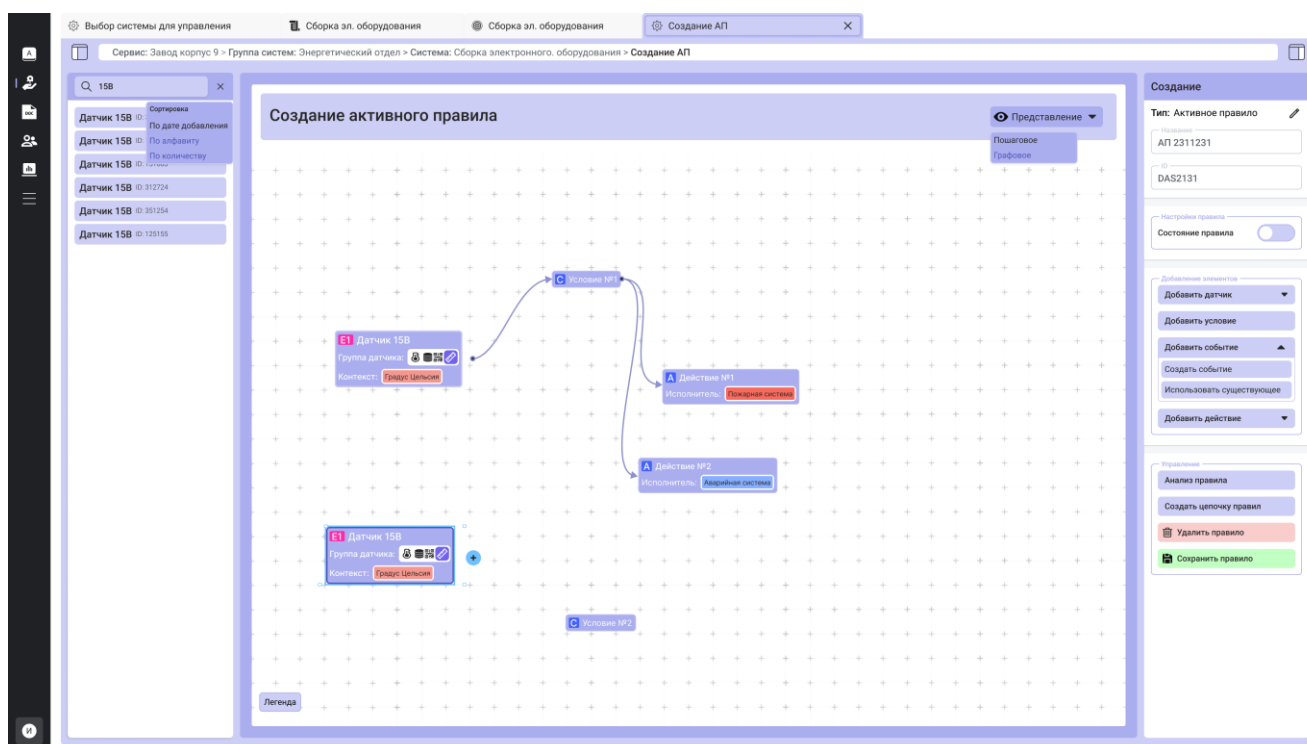


Рисунок 28 – Конструирование правил с использованием NoCode-подхода в графовом представлении

Помимо графового представления, конструирования активных правил может производиться в поаом представлении (рисунок 29). Каждый

пользователь имеет возможность выбрать наиболее подходящее для него представление.

Рисунок 29 – Конструирование правил с использованием NoCode-подхода в пошаговом представлении

На портале управления реализованы все поставленные требования, но для взаимодействия с тем или иным элементом метамодели у пользователя должна быть соответствующая роль. Роли предоставляют индивидуальные права доступа к отдельным частям ИУС (рисунок 30).

Разработчик имеет возможность создавать новые объекты или изменять существующие. Он имеет возможность настраивать взаимосвязи с различными объектами, например установить какое событие должно происходить в результате выполнения активных правил.

Системный аналитик имеет различные инструменты, необходимые для организации элементов, созданных разработчиком. Пользователь с этой ролью также имеет возможность задавать иерархию существующих систем и определять сами объекты управления и наблюдения.

Бизнес-аналитик имеет доступ к информации, которая ранее была записана в аудит микросервисами, о работе системы и её объектов. Основная задача пользователя с этой ролью заключается в анализе выполнения бизнес-процессов и выявлении ошибок или проблемных правил.

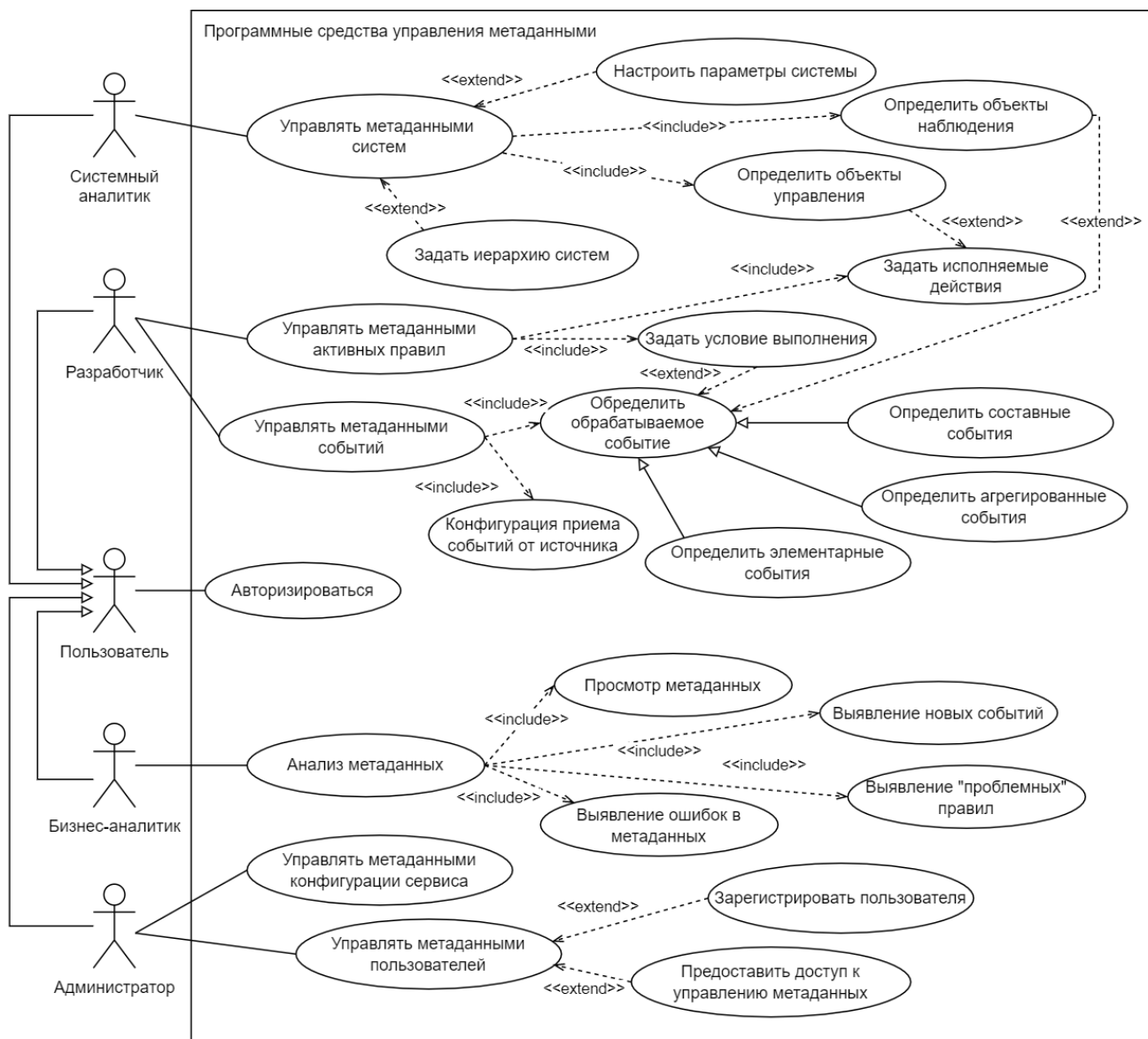


Рисунок 30 – Разграничение прав доступа пользователей на основе ролей

У администратора портала имеются уникальные возможности по управлению метаданными конфигурации ИУС и управлением пользователями (рисунок 31). Регистрация новых пользователей выполняется напрямую через администратора. Также администратор имеет возможность менять определённые аспекты у выбранной роли, тем самым повышается удобство и безопасность процессов управления для каждой группы пользователей.

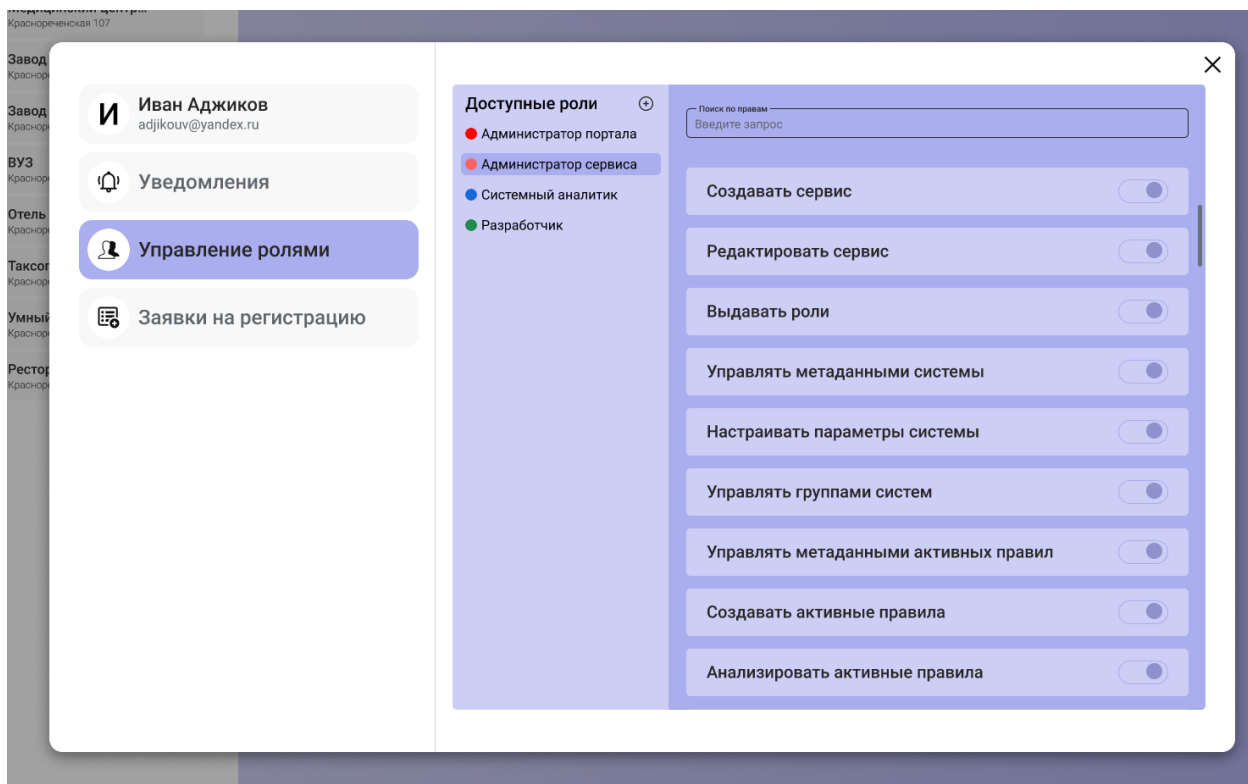


Рисунок 31 – Настройка ролей для пользователей

Пользовательская аутентификация и авторизация были реализованы с применением JSON Web Token (JWT), который основывается на формате JavaScript Object Notation (JSON). Токены создаются на REST-сервере при помощи алгоритма шифрования с использованием секретного ключа. Для усиления безопасности было принято решение задействовать два токена: Access и Refresh. Первый токен передается на сервер вместе с каждым запросом, хранится в локальном хранилище браузера и имеет срок действия два часа. Второй токен предназначен для периодического обновления первого, чтобы пользователь не был вынужден повторно проходить авторизацию при каждом входе в систему. Этот токен находится в HTTP-only cookie браузера и имеет срок действия четырнадцать дней. Если пользователь не будет заходить в систему и не будет совершать никаких действий в течение двух недель, оба токена утратят свою актуальность, и потребуются повторная авторизация.

На диаграмме (рисунок 32) показан запрос к REST-серверу с использованием токенов.

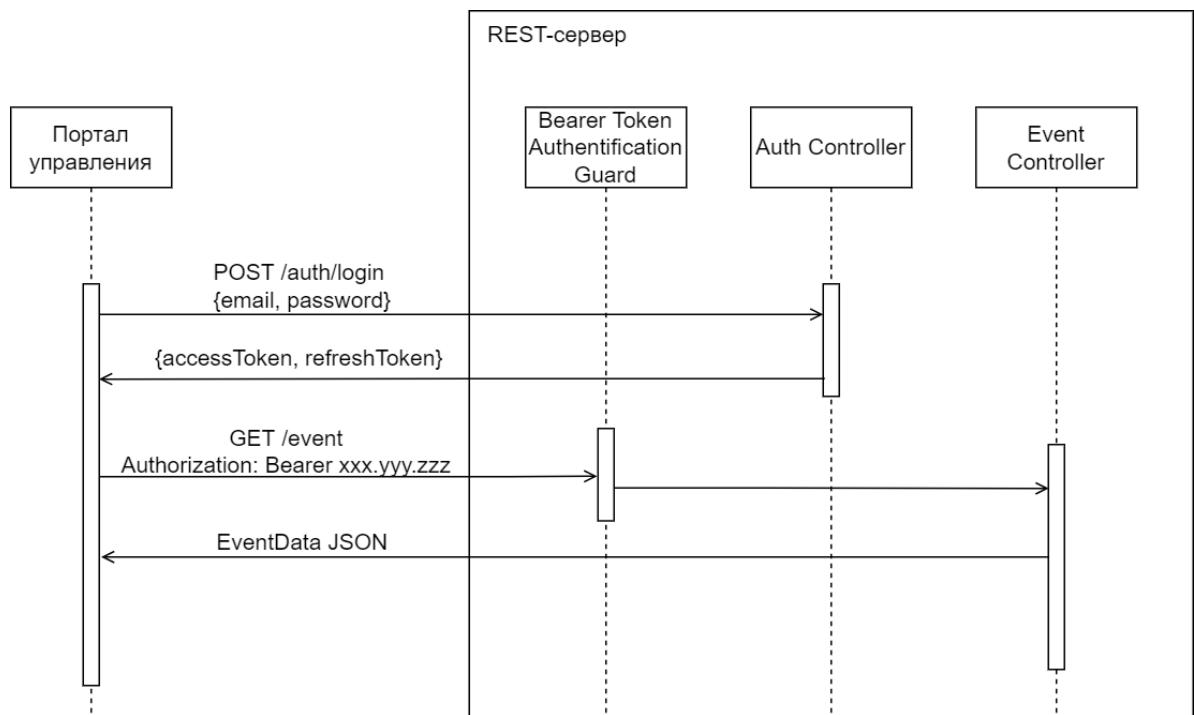


Рисунок 32 – Диаграмма взаимодействия с клиентским приложением

Кроме аутентификации и авторизации, существуют дополнительные программные инструменты для проверки запросов, отправляемых на REST-сервер. Механизм защиты (Guard) отвечает за контроль уровня доступа пользователя к различным операциям управления, а также имеется дополнительный механизм – Class Validator, который отвечает за корректность объекта Data Transfer Object (DTO), отправленного в запросе. В случае обнаружения ошибки доступа, сервер вернёт соответствующую ошибку (рисунок 33).

```

1  {
2    "status": 400,
3    "message": [
4      {
5        "property": "description",
6        "message": "Неверная длина. Мин: 4, макс: 32; Должно быть строкой"
7      },
8      {
9        "property": "contextParamCode",
10       "message": "Должно быть строкой"
11     },
12     {
13       "property": "templateEvent",
14       "message": "Должно быть JSON`ом"
15     }
16   ]
17 }
  
```

Рисунок 33 – Ответ от сервера с ошибкой валидации DTO

На листинге 3 продемонстрировано применение декораторов для определения параметров класса, включаемых в тело запроса на привязку активного правила.

Листинг 3 – Класс для валидации DTO

```
export class BindEventDto {  
    @ApiProperty({ type: String, description: 'Code правила' })  
    @IsString()  
    readonly codeRule: string;  
  
    @ApiProperty({ type: String, description: 'Code события' })  
    @IsString()  
    readonly codeEvent: string;  
  
    @ApiProperty({ type: String, description: 'Type Bind' })  
    @IsEnum(TypeBind)  
    readonly typeBind: TypeBind;  
}
```

Данные механизмы помогают избежать обработки неверных объектов метаданных прямо на этапе обработки самого запроса, тем самым защищая процессы бизнес-логики от некорректных данных.

3.3 Программные средства тиражирования и синхронизации распределенных метаданных

Программное средство для тиражирования и синхронизации метаданных должно быть эффективным с точки зрения производительности. Быстрая доставка метаданных на микросервисы очень важна, поскольку, если процесс тиражирования будет занимать значительное время, то метаданные могут потерять свою актуальность.

3.3.1 Экспериментальное обоснование выбора технологий для реализации алгоритма

Эффективность микросервисной архитектуры в значительной степени зависит от базовых технологий, которые отвечают за обмен и синхронизацию данных между сервисами. Первостепенной задачей является определение подходящего посредника (брокера) сообщений. Этот выбор зависит от нескольких факторов, включая скорость доставки, надежность, масштабируемость и способность удовлетворять уникальные требования процесса тиражирования метаданных.

Был проведен эксперимент для сравнения двух популярных брокеров сообщений, Kafka и Redis, с акцентом на показатель скорости доставки метаданных. Эксперимент проводился на одной ЭВМ, которая имеет процессор с тактовой частотой 4,6 ГГц, размер ОЗУ 32 ГБ и ОС Ubuntu.

Приложения были развернуты с использованием средств Docker, а для организации микросервисов использовался Docker-compose, позволяющий динамически масштабировать количество микросервисов от 1 до 100 экземпляров.

Ключевыми измеряемыми показателями эффективности были минимальное, среднее и максимальное время доставки метаданных, записанное в миллисекундах. Этот диапазон обеспечивал всестороннее представление о производительности каждого брокера при различных условиях загрузки.

Redis продемонстрировал свою способность доставлять метаданные с поразительной скоростью, поддерживая минимальное время доставки в пределах от 1 до 2 миллисекунд и максимальное – от 1 до 5 миллисекунд (рисунок 34). Примечательно, что среднее время доставки постоянно колебалось в пределах 1-2 миллисекунд, независимо от количества микросервисов. Даже при увеличении масштаба Redis продемонстрировал лишь незначительное увеличение максимального времени доставки, что означает стабильную производительность под нагрузкой.

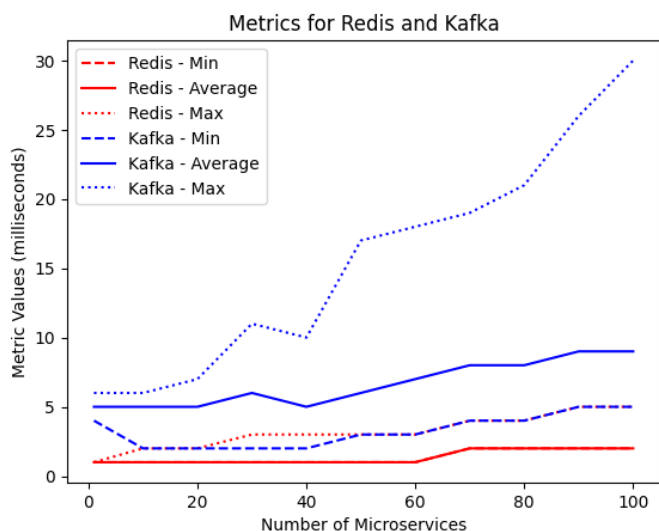


Рисунок 34 – Сравнение времени доставки метаданных

Kafka, являясь надежным брокером сообщений с мощным набором функций, включая хранение данных и отказоустойчивость, продемонстрировал более высокое время доставки метаданных. Минимальное время доставки варьировалось от 2 до 5 миллисекунд, а максимальное - от 6 до 30 миллисекунд. По мере расширения числа микросервисов среднее время доставки Kafka также увеличивалось, начиная с 5 миллисекунд и достигая 9 миллисекунд, что указывает на негативное влияние масштаба на производительность Kafka с точки зрения скорости доставки.

Таким образом, Redis стал лучшим выбором для доставки метаданных в рамках предлагаемой архитектуры благодаря стабильно низкому времени доставки во всех протестированных сценариях. Не смотря на свои сильные стороны, такие как возможность потоковой обработки и хранение сообщений, Kafka оказалась слабее в текущем контексте. Задержки в синхронизации метаданных могут привести к неправильной обработке событий, что потенциально может вызвать системные ошибки и снижение производительности.

Вдобавок к результатам тестирования можно отметить ещё одну очень сильную сторону Redis, которая применима к текущей предметной области. Redis еще является мощным и одним из самых популярных средств для кэширования данных, при помощи него можно разворачивать локальные репозитории напрямую в ОЗУ.

Использование Redis позволяет эффективно инкапсулировать алгоритмы тиражирования и синхронизации метаданных в отдельный контейнер, что в дальнейшем облегчает его интеграцию в микросервисную архитектуру.

3.3.2 Архитектура программных средств в процессе тиражирования и синхронизации

Выбранный брокер сообщений разворачивается внутри Docker-контейнера. Тиражирование метаданных осуществляется посредством

механизма Pub/Sub, который реализует модель публикации и подписки на сообщения. (рисунок 35).

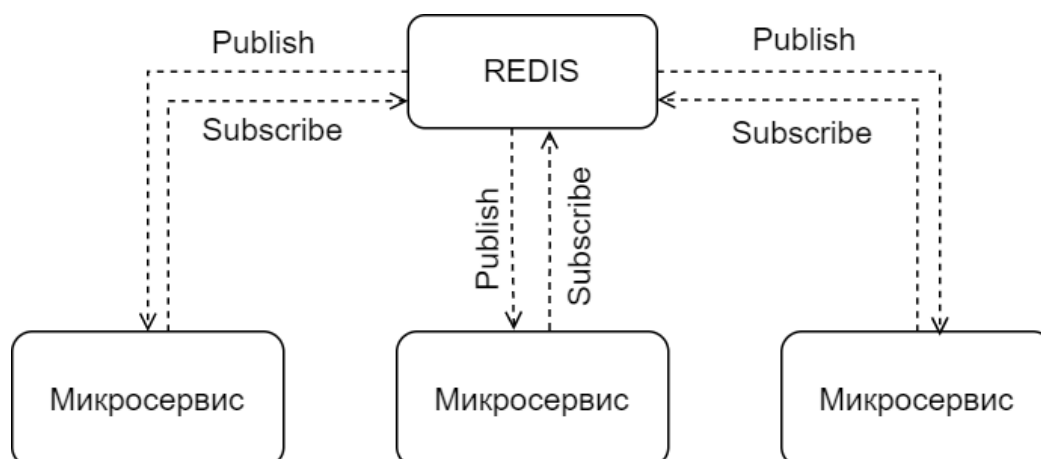


Рисунок 35 – Схема взаимодействия при использовании Redis Pub/Sub

Для передачи данных необходимо опубликовать их в канал брокера сообщений. Такие сообщения доставляются исключительно клиентам, подписанным на данный канал. Сообщение включает как минимум два основных элемента: данные, полученные при публикации, и имя исходного канала.

Реализация механизма Redis Pub/Sub поддерживает сопоставление по шаблонам. Клиенты могут подписываться на шаблоны в стиле glob, чтобы получать все сообщения, отправленные в каналы, соответствующие этим шаблонам. Кроме того, клиент может получить одно и то же сообщение несколько раз, если он подписан на несколько одинаковых шаблонов.

Для начала взаимодействия требуется подключение к транспорту Redis. В листинге 4 демонстрируется подключение микросервиса к Redis.

Листинг 4 – Подключение к Redis

```
this.client = ClientProxyFactory.create({
  transport: Transport.REDIS,
  options: {
    url: 'redis://localhost:6379',
  },
});
```

Фреймворк NestJS, применяемый для разработки REST-серверов и микросервисов, не имеет встроенной поддержки Redis. Однако её можно легко добавить с помощью сторонних библиотек, таких как «nestjs-redis» или

«ioredis». Используя эти библиотеки, можно легко реализовать механизм Pub/Sub в NestJS.

Для этой технологии был разработан алгоритм тиражирования метаданных. Процесс начинается с того, что каждый микросервис при запуске получает необходимый «начальный» набор метаданных, отправляя GET-запрос на REST-сервер (рисунок 36). Этот «начальный» набор служит отправной точкой для обработки событий и сохраняется в собственном локальном репозитории микросервиса.

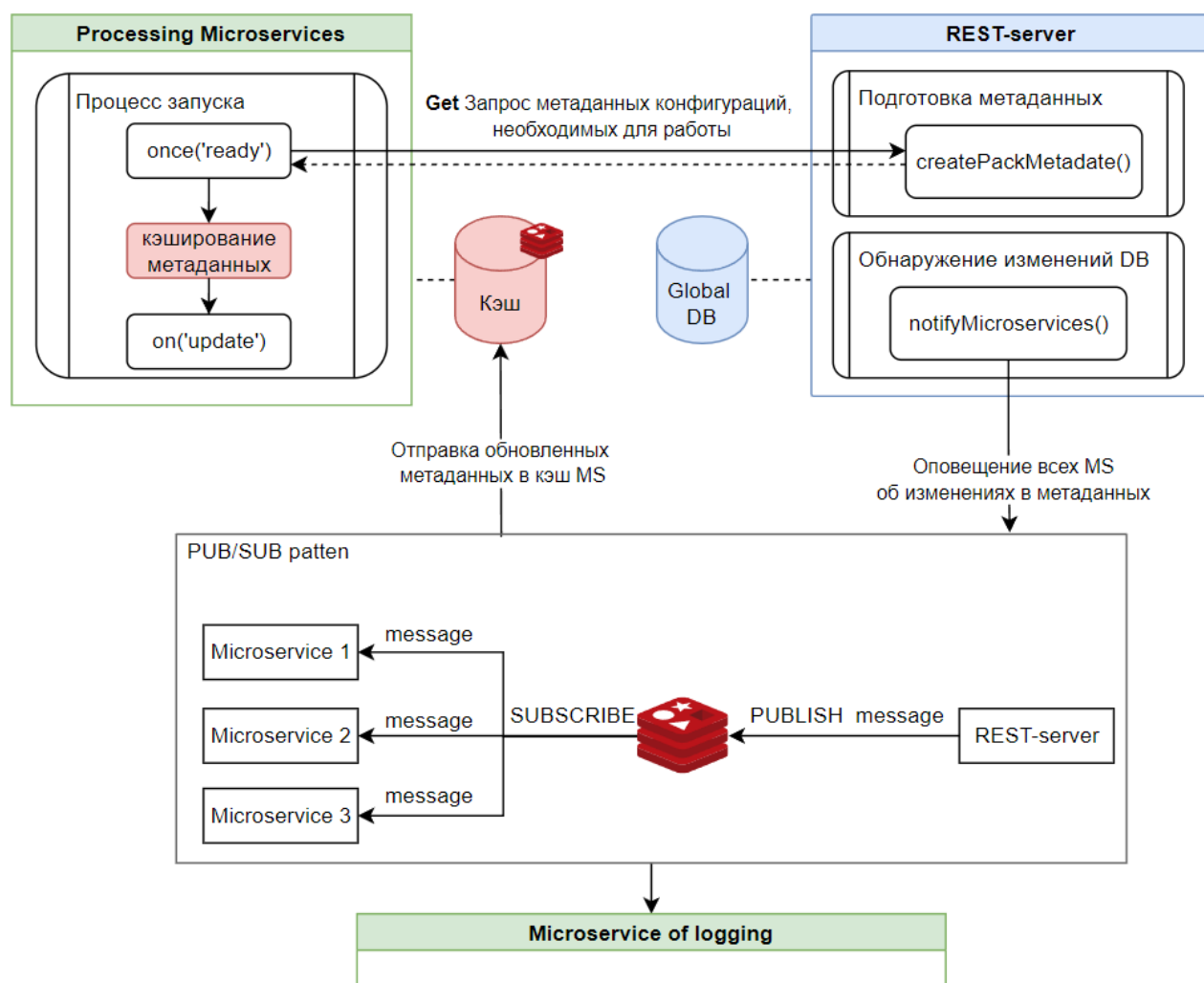


Рисунок 36 – Схема процесса получения и актуализации метаданных для каждого микросервиса

В течение всего жизненного цикла метаданных REST-сервер отслеживает изменения с помощью описанных механизмов. При обнаружении изменений метаданным присваивается новая версия, и отправляется уведомление с уникальными наборами метаданных для каждого

микросервиса. Эти уведомления передаются через заранее установленные каналы Redis Pub/Sub. Такой механизм гарантирует, что каждый микросервис своевременно получит актуальные метаданные, адаптированные к его конкретным требованиям или контексту. Далее микросервис также сохраняет метаданные с своём локальном репозитории.

В процессе работы микросервисы извлекают метаданные непосредственно из своего кэша Redis. Это позволяет эффективно получать доступ к актуальной информации без дополнительных сетевых затрат и задержек, связанных с запросами к REST-серверу.

Для обеспечения актуальности и согласованности был реализован алгоритм синхронизации, который включает механизм отката для устранения любых несоответствий, возникающих в процессе обновления метаданных. В случае сбоя или ошибки при доставке метаданных процесс отката направлен на отмену любых изменений, произведенных в рамках неудачной попытки доставки. Данный процесс показан на рисунке 37.

Алгоритм включает в себя следующие 5 шагов:

- 1) происходят изменения метаданных,
- 2) REST-сервер отправляет обновленные метаданные каждому микросервису,
- 3) при получении новых метаданных каждый микросервис сохраняет их в локальный репозиторий, параллельно отправляя соответствующий статус назад, REST-серверу,
- 4) если все микросервисы сообщают об успешной доставке, процесс считается завершенным и никаких дальнейших действий не требуется,
- 5) если не все микросервисы сообщают об успешной доставке в течение указанного периода времени, что указывает на сбой в применении новых метаданных, инициируется процесс отката.

Процесс отката подразумевает аннулирование изменений в метаданных и восстановление их предыдущей версии.

инициируется процесс отката для поддержания согласованности данных и отмены любых изменений, внесенных во время неудачной попытки доставки.

3.3.3 Применение Sidecar паттерна

В разработке микросервисов для распределённой ИУС возникает проблема дублирования методов при реализации алгоритмов тиражирования и синхронизации метаданных. Это приводит к увеличению сложности поддержки и обновления системы, а также к потенциальному увеличению числа ошибок из-за необходимости поддерживать одинаковое поведение в разных частях системы. Одним из решений, позволяющим уменьшить дублирование кода и централизовать управление общими задачами, является использование шаблона проектирования Sidecar в сочетании с контейнеризацией, например, с использованием средств Docker [31]. Данный шаблон проектирования полезен в микросервисной архитектуре, особенно сильно, если микросервисы написаны с помощью разных технологий или языков программирования.

Каждый микросервис, как правило, разрабатывается и поддерживается разными командами. При использовании Pub/Sub на каждом микросервисе должны быть реализованы механизмы подписки на соответствующие темы, обработки полученных оповещений об обновлениях и корректного обновления своего локального репозитория. Это не только увеличивает объем трудозатрат, но и повышает риск ошибок из-за различий в реализации этих механизмов в разных сервисах. Риск становится ещё больше если микросервисы разрабатываются на разных языках. Из-за технических особенностей отдельных программных средств могут появляться различные проблемы.

Паттерн позволяет выделить определенные функциональные возможности в отдельный контейнер, упрощая разработку и обслуживание каждого микросервиса. Используя шаблон Sidecar, разработчики могут сосредоточиться на основной бизнес-логике своих микросервисов, не заботясь о деталях реализации специфических алгоритмов и механизмов.

В контексте тиражирования и синхронизации метаданных Sidecar может инкапсулировать в себе следующие задачи:

- взаимодействие с механизмами Redis Pub/Sub и обработку входящих оповещений,
- применение обновлений к микросервису, путем их кэширования в резидентную СУБД,
- логику алгоритма синхронизации с операцией отката.

Для реализации Sidecar можно использовать средства Docker, которые позволяют размещать разработанные средства в виртуализированных контейнерах.

Создание Sidecar контейнера включает в себя следующие 3 шага:

- 1) разработка приложения Sidecar, которое будет инкапсулировать описанный ранее функционал,
- 2) настройка Dockerfile для Sidecar, в котором определяется среда выполнения, необходимые зависимости и команды для запуска,
- 3) конфигурация таких средств как Docker Compose или Kubernetes, для организации развертывания сразу нескольких контейнеров.

Sidecar контейнер разворачивается рядом с каждым микросервисом и тесно с ним связывается с точки зрения жизненного цикла, но остается слабо связанным с точки зрения функциональности. Фрагмент архитектуры развертывания микросервисов с использованием Sidecar представлен на рисунке 38.

Данное улучшение для управления метаданными в архитектуре ИУС несёт в себе следующие преимущества:

- общая функциональность выносится в отдельный компонент, что сокращает объем кода в каждом микросервисе,
- упрощается обновление и поддержка, так как изменения фиксируются в одном месте,
- общий код тестируется централизованно, что повышает качество и надежность системы,

– повышается устойчивость архитектуры за счет локализации и сдерживания сбоев в отдельных контейнерах, которые могут быть перезапущены или заменены без ущерба для самого микросервиса.

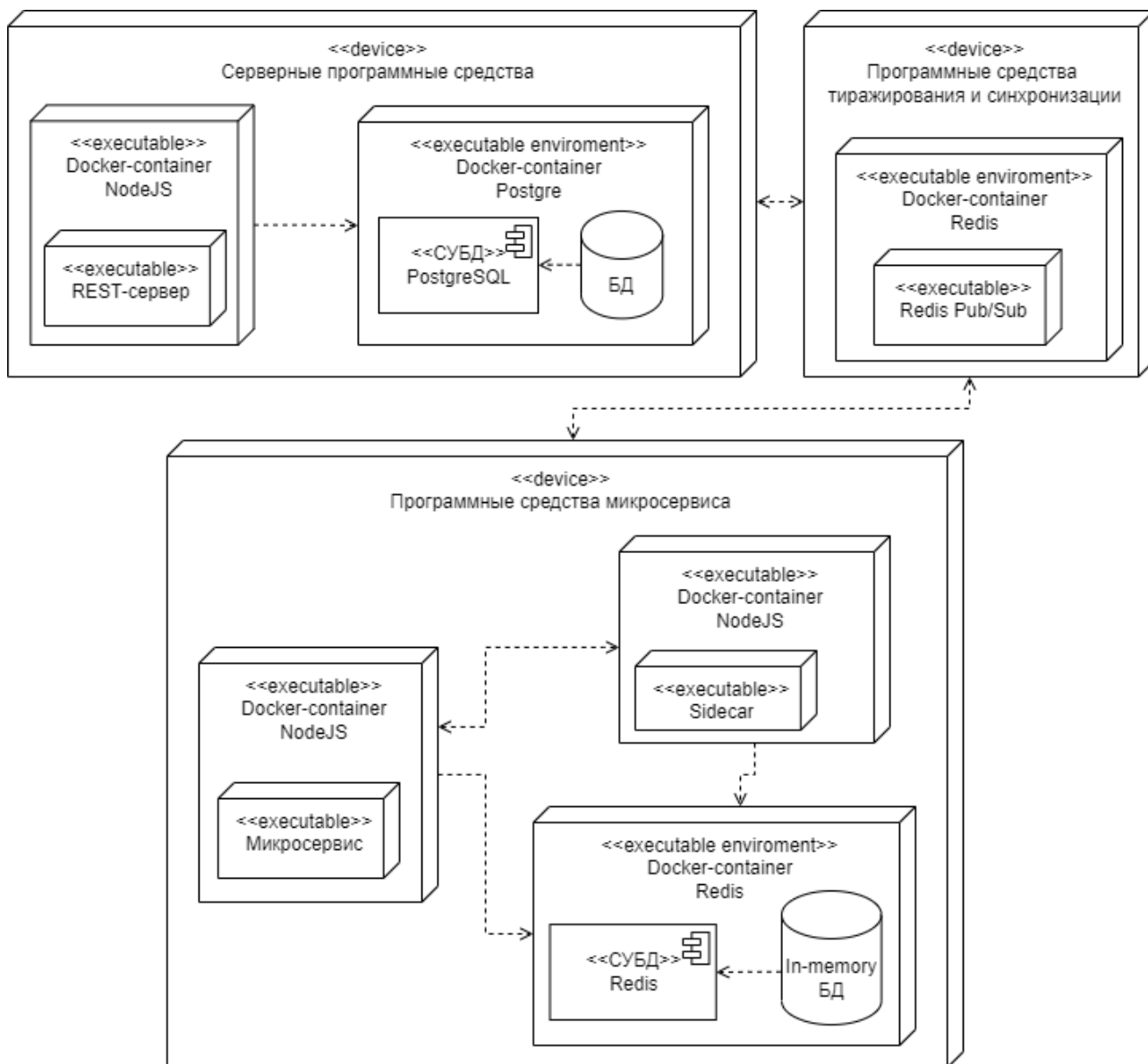


Рисунок 38 – Фрагмент архитектуры развертывания микросервисов

Важно заметить, что Sidecar может быть легко адаптирован или заменен на другой, не влияя на разработанные микросервисы.

3.4 Выводы по разделу 3

По итогу данной главы были рассмотрены аспекты применения программных средств для тиражирования и синхронизации метаданных. Были детально описаны клиентские и серверные средства конструирования. С серверной стороны за данные механизмы отвечает REST-сервер, с клиентской – разработанное SPA приложение с применением NoCode-

подхода. Для взаимодействия используется HTTP(S) протокол с передачей данных в формате JSON.

Были рассмотрены различные аномалии при некорректном конструировании метаданных, представлены программные средства выполнения статического анализа.

Для процессов тиражирования была выбрана технология Redis Pub/Sub. Был проведен эксперимент для более корректного обоснования выбора. Разработанные алгоритмы были реализованы с применением данной технологии.

Для улучшения ряда архитектурных особенностей был рассмотрен паттерн проектирования Sidecar, который лег в основу реализации инкапсулированных виртуализированных контейнеров для реализации алгоритмов.

4 Экспериментальная оценка разработанных моделей, алгоритмов и программных средств управления метаданными

Нужно организовать и провести эксперимент, который будет направлен на оценку эффективности, отказоустойчивости и способности к масштабированию созданных алгоритмов и программных средств для управления метаданными. Для проведения эксперимента необходимо смоделировать сценарии, отражающие сложности и требования, с которыми обычно сталкиваются такие системы. Необходимо оценить, насколько хорошо разработанные решения справляются с высоконагруженными потоками событий в режиме реального времени, обеспечивая бесперебойное выполнение активных правил и поддержание синхронизации метаданных.

4.1 Планирование и постановка эксперимента

Эксперимент необходимо проводить в управляемой виртуализированной среде, которая имитирует производственные условия. Таким образом эксперимент позволит оценить производительность и степень отказоустойчивости разработанных моделей, алгоритмов и программных средств управления метаданными в условиях высоконагруженной информационно-управляющей системы.

4.1.1 Сценарии и условия проведения эксперимента

Необходимо создать несколько сценариев с различной пропускной способностью шлюза (Gateway) и различной частотой поступающих в поток событий. В экспериментальных сценариях необходимо наличие нескольких обязательных компонентов.

Сервис генерации событий. Данный сервис будет имитировать наблюдаемые объекты системы и производить отправку в шлюз событий с различной частотой и контекстом. Он позволит смоделировать высоконагруженные потоки событий в режиме реального времени.

Шлюз ИУС. Он позволит обрабатывать входящие потоки событий и обогащать конкретные экземпляры событий текущей версией метаданных. Далее шлюз будет перенаправлять события в соответствующие микросервисы.

Средства транспортировки событий. Для транспортировки событий между микросервисами будет выступать вспомогательное программное обеспечение – брокер сообщений.

Главный компонент ИУС. В качестве данного компонента будет выступать отдельный сервер, который будет напрямую взаимодействовать с центральным репозиторием метаданных. Для целей эксперимента будет разработан модуль, который имитирует обновление метаданных через заданный интервал времени. Обновления будут поступать в сторонний модуль, который отвечает за отслеживание изменений, поддержание целостности различных версий и наборов метаданных. В процессе тиражирования метаданных микросервисам главный компонент ИУС будет отвечать за алгоритмы синхронизации, и в случае успешного тиражирования будет обновлять текущую версию метаданных.

Набор микросервисов, имитирующих процесс обработки событий. Микросервисы не должны обладать реальными алгоритмами обработки событий, чтобы гарантировать чистоту сбора метрических показателей. Вместо методов обработки будет реализована их имитация (Mock) с фиксированных временем работы.

Программные средства тиражирования и синхронизации. Они должны позволить передавать обновления метаданных на соответствующие Sidecar контейнеры.

Sidecar контейнеры. Разработанные средства будут инкапсулировать в себе всю логику тиражирования и синхронизации метаданных. Контейнеры будут спроектированы таким образом, что смогут определять какие именно наборы метаданных необходимы их собственным микросервисам и будут обновлять именно эти метаданных в локальных репозиториях.

Средства сбора метрических показателей и их визуализации. Для более наглядного и точного представления результатов эксперимента необходимо развернуть средства, которые позволят собирать метрические показатели со всех компонентов инфраструктуры. В дальнейшем эти показатели будут визуализированы в различные графические представления.

Обобщенная структура описанных компонентов для эксперимента представлена на рисунке 39.

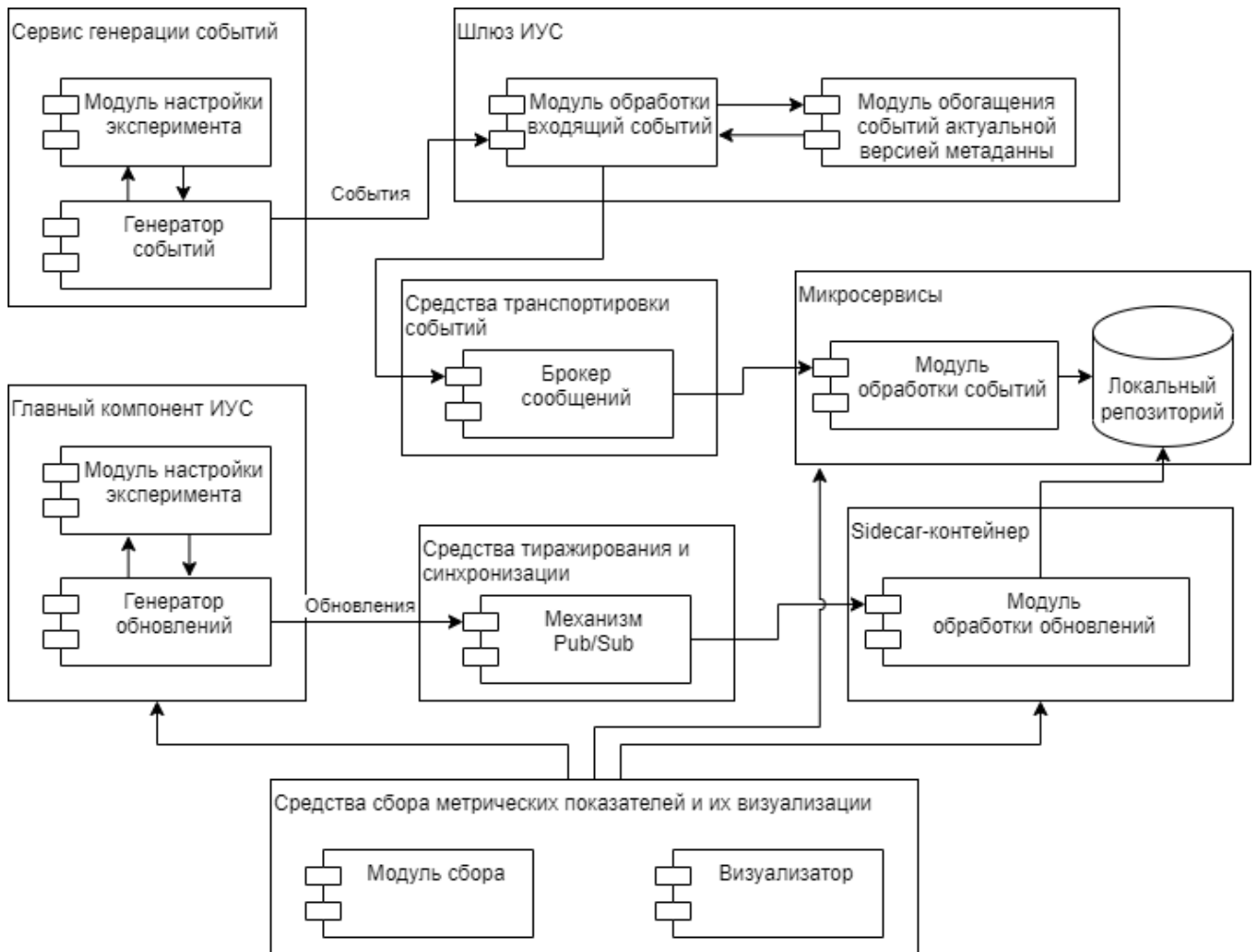


Рисунок 39 – Обобщенная структура элементов архитектуры для эксперимента

Каждый этап эксперимента будет фиксироваться в журнале обработки событий и в журнале процессов обновления метаданных. Администратор сможет отслеживать состояние ИУС в режиме реального времени. Данные будут представлены при помощи различных диаграмм и прочих компонентов визуализации.

Для повышения актуальности эксперимента необходимо будет использовать последние стабильные версии вспомогательного программного обеспечения. Разработанные программные средства необходимо будет развернуть в виртуальных контейнерах с возможностью ведения журнала и мониторинга.

4.1.2 Параметры экспериментальной оценки программных средств

Для оценки эффективности разработанных решений необходимо измерять показатели при различных условиях нагрузки и модификациях конфигурации. Необходимо будет измерять время обработки события на каждом микросервисе, а также общее время обработки цепочки событий. Необходимо оценить отказоустойчивость и способность к масштабируемости системы, убедиться, что она может обрабатывать увеличивающийся объем событий или обновлений при сохранении приемлемой пропускной способности. В качестве параметров экспериментальной оценки могут выступать:

- общее количество обработанных событий на каждом микросервисе,
- количество событий, находящихся в обработке на данный момент,
- среднее время обработки событий на каждом микросервисе,
- количество обновлений метаданных за определённый промежуток времени,
- время тиражирования обновлений метаданных, включая время доставки на отдельные микросервисы и шлюз ИУС,
- дополнительные флаги верификации, свидетельствующие на наличие расхождений в версиях метаданных при обработке событий.

Для комплексного мониторинга и анализа производительности ИУС будут применяться современные средства сбора и визуализации, такие как Prometheus и Grafana [32-33]. Средства позволят не только отображать состояние системы в реальном времени, но и будут сохранять исторические данные в собственной системе хранилищ, которая основана на модели

временных рядов, что позволит проанализировать возникшие аномалии в определённый интервал времени.

Данные инструменты облегчат детальный анализ системы и позволят выявить узкие места, требующие улучшения. Экспериментальная оценка позволит получить четкое представление об эффективности разработанных решений по управлению метаданными в рамках динамичной архитектуры, управляемой событиями.

4.1.3 Предметная область ИУС для экспериментальной оценки

Для проведения эксперимента была выбрана сфера розничной торговли и электронной коммерции, поскольку она идеально подходит для оценки эффективности, масштабируемости и отказоустойчивости разработанных инструментов.

Существующие программные решения, разработанные для данной предметной области, зачастую сталкиваются с целым рядом проблем. Например, резкий рост объема транзакций, нестандартное поведение пользователей, сбои в автоматической корректировке запасов на складах, некорректный персонализированный маркетинг и так далее.

В данной предметной области активные правила отлично подойдут для автоматизации различных бизнес-процессов в режиме реального времени. Рассмотрим некоторые примеры активных правил, которые могут быть использованы в данной предметной области:

- правила корректировки цен, которые могут автоматически изменять цену в зависимости от количества запасов на складе, цен конкурентов или колебаний спроса,
- правила пополнения запасов, которые позволяют производить закупку, когда уровень запасов товаров, пользующихся большим спросом, падает ниже заданного порога,
- правила обнаружения мошенничества позволят выявлять потенциально мошеннические транзакции, например быстрая последовательность множества транзакций от одного пользователя,

– правила выявления нестандартных закономерностей, отличающихся от заданных шаблонов,

– правила динамической сегментации клиентов на основе их истории заказов.

Для данной предметной области необходимо будет смоделировать несколько подробных сценариев, которые будут отличаться друг от друга и затрагивать «проблемные» ситуации.

Например, можно смоделировать множественную закупку в праздничных сезон, при которой резко будет возрасть количество событий на покупку определённых позиций. События должны быть корректно обработаны за приемлемый срок.

Также время тиражирования обновлений никак не должно зависеть от таких всплесков. Зачастую в таких случаях требуется динамическая корректировка цен на товары или иные операции, которые могут осуществляться администратором, поэтому обновление метаданных должно происходить максимально быстро. Данный случай может позволить оценить способность системы управлять большим объемом данных и точно применять сложную бизнес-логику.

Для более точного анализа эксперимента можно декомпозировать этот случай на несколько сценариев и проверять отдельно увеличение числа событий и увеличение числа обновлений метаданных. Каждый сценарий будет разработан таким образом, чтобы изучить производительность, масштабируемость и точность системы в момент управления метаданными и выполнения множества активных правил.

4.2 Разработка, развертывание и настройка информационного и программного обеспечения эксперимента

Для целей проводимого эксперимента была разработана микросервисная архитектура, которая также включает различные средства

СУБД и средства мониторинга и анализа производительности. Рассмотрим разработанные элементы архитектуры.

Генератор событий. Данный микросервис отвечает за генерацию событий по заранее заданной конфигурации. Так как контекст события не имеет значения, то для эксперимента будет использоваться самый простой экземпляр события. Данный микросервис будет напрямую подключаться к шлюзу ИУС по протоколу Message Queuing Telemetry Transport (MQTT). Это легковесный протокол обмена событиями, который преимущественно используется в системах «Интернета вещей» для обеспечения надежной и эффективной связи между устройствами с ограниченными ресурсами. Он реализован на основе модели «издатель-подписчик», что позволяет устройствам публиковать сообщения без необходимости поддерживать постоянное соединение с сервером, тем самым сокращая использование сетевых ресурсов. Фрагмент кода для отправки событий в шлюз представлен на листинге 5.

Листинг 5 – Отправка сгенерированного события в шлюз

```
import mqtt from 'mqtt';
const clientId = `mqtt_${Math.random().toString(16).slice(3)}`;
const client = mqtt.connect('mqtt://mqtt-broker:1883', {
  clientId,
  clean: true,
  connectTimeout: 4000,
  reconnectPeriod: 1000,
});
// Тема, куда будут отправляться события
const topic = 'gateway:data';
// Событие
const message = JSON.stringify({ value: 22.4, timestamp:
Date.now() });
// Время между отправкой сообщений, мс
const messageSendTime = 1000;

client.on('connect', () => {
  console.log('Connected');
  setInterval(() => {
    client.publish(topic, message, { qos: 0, retain: false },
(error) => {
      if (error) {
        console.error('Publish error: ${error}');
      }
    });
  });
});
```

```

    }, messageSendTime);
  });

client.on('error', (err) => {
  console.error('Connection error: ', err);
  client.end();
});

```

Шлюз ИУС. Данный микросервис принимает входящие события по MQTT протоколу. Событие на входе обогащается соответствующей версией метаданных, которая хранится в кэше шлюза, и после этого отправляются на микросервисы.

Главный компонент ИУС. В реализации для эксперимента роль компонента состоит в отслеживании изменений в метаданных. Генерации этих изменений происходит при помощи модуля, который был написан для целей эксперимента. Модуль работает на основе интервалов, то есть в заданный момент времени происходит обновление самой версии метаданных, что приводит к вызову процесса отслеживания обновлений. Сами обновления отправляются на микросервисы при помощи Redis Pub/Sub в полном соответствии с разработанными алгоритмами тиражирования и синхронизации. После подтверждения обновлений на каждом микросервисе главный компонент взаимодействует со шлюзом при помощи системы удалённого вызова процедур – gRPC. Данная технология была разработана Google и позволяет эффективно взаимодействовать между парой микросервисов в распределённых системах. Принцип работы заключается в следующем: главный компонент вызывает напрямую метод обновления версии метаданных, который реализован на шлюзе, как будто он реализован локально. Таким образом шлюзу нет необходимости заботиться об отслеживании версий метаданных, ему просто необходимо обогащать события и отправлять дальше в соответствующий микросервис.

Микросервисы. Был разработан микросервис, который клонируется на несколько экземпляров, тем самым имитируя работу нескольких разных микросервисов. Для целей эксперимента процесс обработки событий происходит последовательно между всеми микросервисами.

Sidcar-контейнер. Данный контейнер разворачивается рядом с каждым микросервисом. Он инкапсулирует логику по обработке входящих обновлений и записи их в локальный репозиторий. Фрагменты кода данных операций представлены на листингах 6-7.

Листинг 6 – Фрагмент контроллера для обработки входящих событий
@Controller()

```
export class AppController {
  constructor(private readonly appService: AppService) {}

  private readonly logger = new Logger(AppController.name);

  @EventPattern('metadata:update')
  handleDataEventRedis(@Payload() data: any, @Ctx() context:
RedisContext) {
    this.logger.log('Received data: ${JSON.stringify(data)} on
topic: ${context.getChannel()}');
    // Обновление метаданных
    this.appService.updateMetadata(data);
    return;
  }
}
```

Листинг 7 – Фрагмент сервиса для сохранения набора метаданных
@Injectable()

```
export class AppService {
  constructor(
    @Inject(CACHE_MANAGER) private cacheManager: Cache,
    @Inject('REDIS_PUB_SUB_SERVICE') private clientRedis:
ClientProxy,

    private readonly metricsService: MetricsService,
  ) {}

  async updateMetadata(data: any) {
    // Обновление метаданных
    this.metricsService.recordReceivedMetadataUpdate();
    await
this.cacheManager.set('metadata:version:${data.version}', data,
0);

    // Обновление метаданных завершено
    this.metricsService.recordProcessedMetadataUpdate();
    this.emitConfirmation(data.version, 'success');

    // Обновление метаданных завершилось ошибкой
    // this.metricsService.recordFailedMetadataUpdate();
  }
}
```

```

    async emitConfirmation(version: string, status: 'success' |
'failed') {
      this.clientRedis.emit('metadata:update:confirmed', {
        version,
        microservice: process.env.MICROSERVICE_NAME,
        status,
      });
    }
  }
}

```

При сохранении каждый набор метаданных сохраняется под своим ключом в локальном репозитории. В конце каждого ключа указывается уникальная версия, которая служит главным критерием для идентификации набора с которым необходимо обрабатывать входящие события

Вся система развернута в виртуализированной среде с использованием средств Docker и Docker Compose для обеспечения изоляции и управляемости компонентов. Диаграмма развёртывания представлена на рисунке 40.

Для взаимодействия между контейнерами была использована Docker сеть. Сеть была настроена как мост (bridge), что позволило контейнерам взаимодействовать друг с другом как независимые устройства, обеспечивая изоляцию от хост-машины и других сетей. Использование данной сети также упростило настройку сетевых параметров, таких как порты и IP-адреса, что позволило с лёгкостью интегрировать средства мониторинга на каждый контейнер.

Окружение каждого контейнера настраивалось при помощи переменных окружения, которые были переданные в соответствующий файл для средств Docker Compose. Переменные окружения – это набор значений, которые определяют поведение и настройки операционной системы, а также других программ, работающих в этой среде. Они позволяют легко изменять конфигурацию приложения, при этом не затрагивая программный код. Также хранение всех секретных данных в переменных окружения позволит избежать их случайного разглашения, так как они не включаются в исходный код проекта.

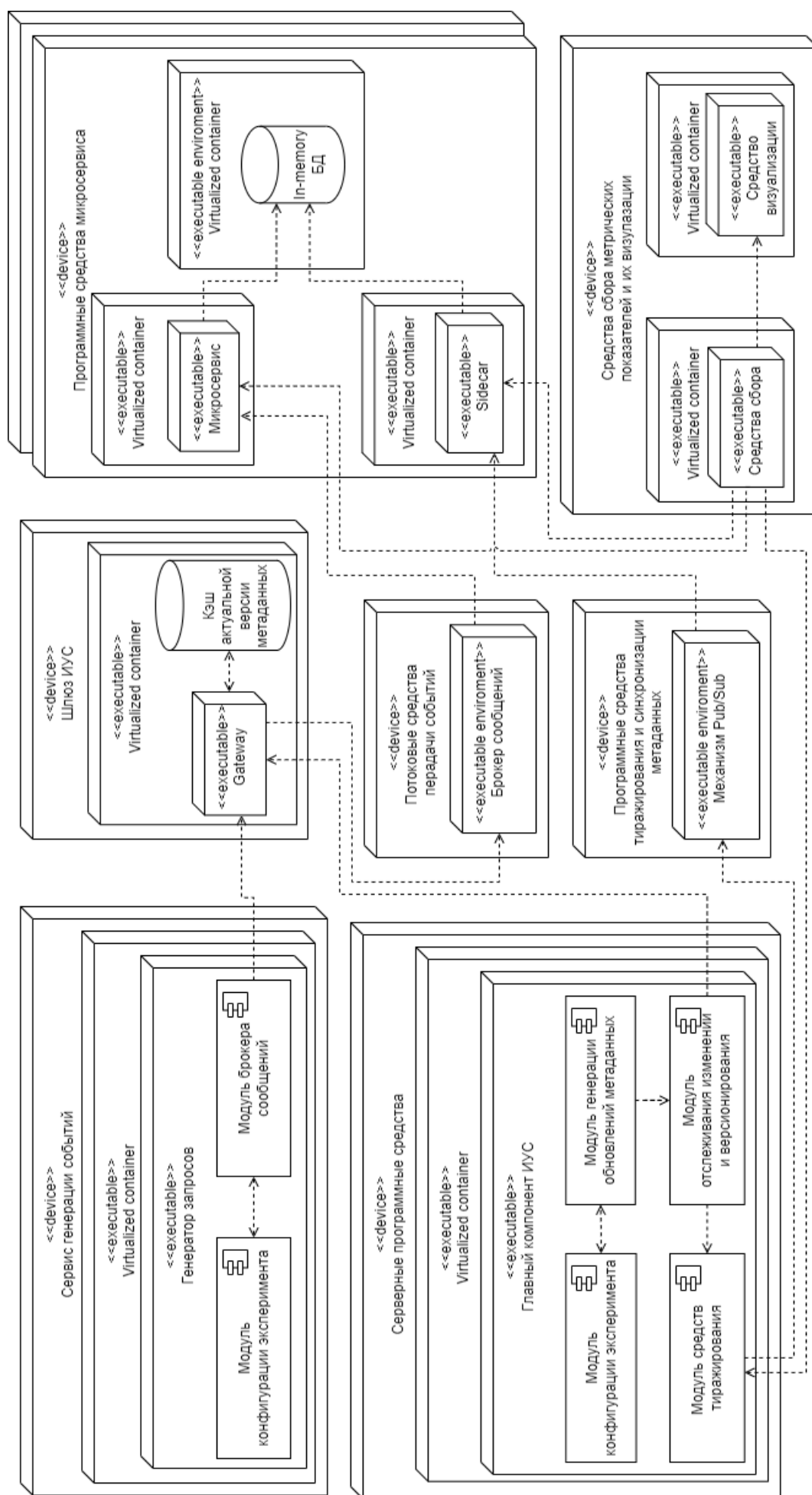


Рисунок 40 – Диаграмма развёртывания архитектуры для проведения эксперимента

Программные средства микросервиса были развёрнуты дважды, чтобы имитировать работу двух независимых микросервисов с самостоятельным инкапсулированным окружением.

В общей сложности было развёрнуто 15 виртуализированных контейнеров. Детальное описание контейнеров представлено в таблице 3.

Таблица 3 – Виртуализованные контейнеры для проведения эксперимента

Название	Описание
Backend Core	Контейнер, содержащий центральный компонент системы, который управляет основной бизнес-логикой и взаимодействием между различными сервисами. Также данный контейнер отслеживает изменения метаданных.
Migrations	Сервис, задачей которого является управление миграциями баз данных, что позволяет обновлять и поддерживать структуру базы данных в актуальном состоянии.
Gateway	Контейнер, содержащий шлюз, выступает в качестве точки входа в систему, распределяя входящие запросы по другим сервисам. Это первый уровень обработки и фильтрации событий перед их дальнейшей обработкой.
Processor 1	Контейнер, имитирующий работу первого микросервиса. В него сообщения поступают напрямую из шлюза.
Processor 2	Второй контейнер с микросервисом в последовательности обработки события.
Sidcar Processor 1	Sidcar-контейнер для первого микросервиса. Обеспечивает функции, связанные с тиражированием данных и синхронизацией метаданных между центральной базой и локальными хранилищами каждого микросервиса
Sidcar Processor 2	Sidcar-контейнер для второго микросервиса.
Redis Processor 1	Экземпляр Redis для реализации локального хранилища первого микросервиса. Разворачивается рядом с микросервисом и его Sidcar-контейнером.
Redis Processor 2	Экземпляр Redis для реализации локального хранилища второго микросервиса.
Event Generator	Контейнер, содержащий генератор событий для имитации работы датчиков.
MQTT Broker	Брокер сообщений, использующий протокол MQTT для обмена сообщениями между компонентами системы.
Postgre	Реляционная система управления базами данных, используемая для организации центрального репозитория метаданных.

Продолжение таблицы 3

Redis Pub Sub	Экземпляр Redis, используемый специально для реализации механизма публикации и подписки, который позволяет микросервисам получать обновления метаданных.
Prometheus	Контейнер, в котором развёрнута система мониторинга Prometheus для сбора метрик со всех других контейнеров в сети Docker.
Grafana	Контейнер, внутри которого развёрнут инструмент визуализации данных Grafana. В контейнер данные импортируются из Prometheus. Grafana доступен для подключения из локальной сети и предоставляет веб-интерфейс для удобного отслеживания метрик.

Для повышения отказоустойчивости и сохранности данных были использованы volumes. Volumes в Docker позволяют хранить данные вне контейнеров, обеспечивая их сохранность даже после удаления или перезапуска самих контейнеров.

Было создано два volumes: «ars-pgdata» и «grafana_data», которые используются для хранения данных соответствующих сервисов – PostgreSQL и Grafana. Volume «ars-pgdata» предназначен для сохранения данных в центральном репозитории, с которым взаимодействует главный компонент ИУС. Аналогично, volume «grafana_data» используется для хранения конфигураций, дашбордов и других данных Grafana.

На каждом контейнере предварительно были настроены соответствующие классы метрик, которые понадобятся на этапе проведения и анализа. Данные настройки были указаны в специальном конфигурационном файле (листинг 8).

Листинг 8 – Конфигурационный файл для Prometheus

```
global:
  scrape_interval: 10s

scrape_configs:
  - job_name: 'main'
    static_configs:
      - targets: ['backend:3000']
  - job_name: 'gateway'
    static_configs:
      - targets: ['gateway:3001']
  - job_name: 'processor1'
```

```

static_configs:
  - targets: ['processor1:3002']
- job_name: 'processor2'
  static_configs:
    - targets: ['processor2:3003']
- job_name: 'sidecar_processor1'
  static_configs:
    - targets: ['sidecar-processor1:4002']
- job_name: 'sidecar_processor2'
  static_configs:
    - targets: ['sidecar-processor2:4003']

```

Для более точного анализа сбор метрических показателей будет производиться Prometheus каждые 10 секунд. Были указаны адреса и порты каждого контейнера, на котором производится сбор метрик.

Разработанная архитектура позволит провести эксперимент с учётом заданных требований и собрать все метрические показатели, необходимые для оценки качества разработанных алгоритмов и средств.

4.3 Проведение эксперимента

Эксперимент должен проводиться на рассмотренной архитектуре и включать в себя несколько сценариев, которые позволят полноценно сделать выводы, касаясь разработанных алгоритмов и средств, на этапе анализа. Существует три гипотезы, которые необходимо проверить в ходе эксперимента:

- время тиражирования не должно зависеть от количества поступающих событий,
- время обработки событий и время обращение к локальному репозиторию не должно зависеть от моментов тиражирования,
- все микросервисы должны обрабатывать события с единой версией метаданных, даже при большом потоке обновлений микросервисы должны корректно работать со всеми соответствующими версиями метаданных.

С целью детальной проверки первой гипотезы необходимо смоделировать соответствующий сценарий, в котором будет происходить динамическая смена времени отправки событий в ИУС. Данный сценарий представлен на рисунке 41.

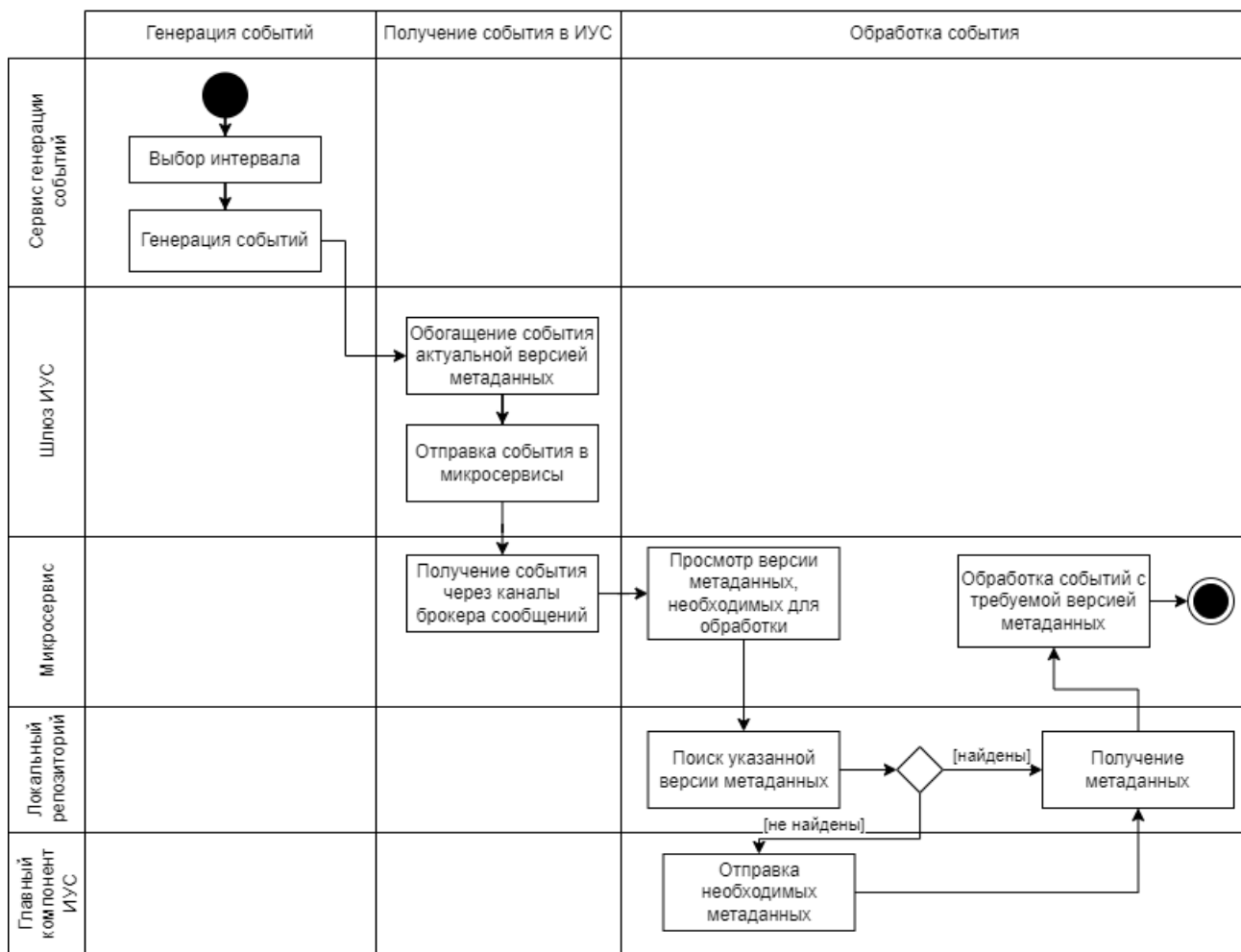


Рисунок 41 – Первый сценарий эксперимента

В ходе эксперимента по текущему сценарию события будут генерироваться на протяжении 10 минут. Постепенно сервис будет увеличивать частоту генерации событий от 20 событий/минуту до 2000 событий/минуту. Для данного сценария необходимо добавить в сервис генерации рекурсивный метод, который позволит динамически изменять значение интервала (листинг 9).

Листинг 9 – Динамическое обновление интервала для генерации событий

```
client.on('connect', () => {
  console.log('Connected');

  const startTime = Date.now();
  const duration = 10 * 60 * 1000; // 5 минут в миллисекундах
  let intervalId;

  const updateInterval = () => {
    const elapsed = Date.now() - startTime;
    if (elapsed > duration) {
      clearInterval(intervalId);
      console.log('Experiment ended.');
```

```

    return;
}

// Расчёт нового интервала
const rate = 20 + 1980 * (elapsed / duration); // Плавное
изменение от 20 до 2000 событий в минуту

const messageSendTime = 60000 / rate; // Переводим частоту
событий в минуту в интервал в миллисекундах

// Публикация сообщения
const message = JSON.stringify({ temperature: 22.4 +
Math.random() * 5, timestamp: Date.now() }); // Добавим немного
случайности к температуре
client.publish(topic, message, { qos: 0, retain: false },
(error) => {
    if (error) {
        console.error('Publish error: ${error}');
    }
});

// Перезапуск таймера с новым интервалом
clearInterval(intervalId);

intervalId = setInterval(updateInterval, messageSendTime);
};

intervalId = setInterval(updateInterval, 60000 / 20); //
Начнем с 20 событий в минуту
});

```

Данный фрагмент кода напрямую будет затрагивать настройку интервала в сервисе генерации событий. После завершения заданного времени события перестанут генерироваться.

Так как данные об обработке событий будут фиксироваться в Prometheus в формате временных рядов, то удобно будет агрегировать их и сопоставить с моментами тиражирования обновлений на графиках. Время тиражирования и синхронизации обновлений не должно изменяться, оно должно происходить в рамках приемлемого временного окна.

Для проверки второй и третьей гипотезы будет смоделирован ещё один сценарий (рисунок 42). Доступ к локальному репозиторию необходим во время обработки поступающих событий, поэтому мониторинг времени доступа к этому репозиторию включён в метрику времени обработки. Для данной модели эксперимента все процессы обработки событий на всех

Процессы тиражирования и синхронизации в эксперименте затрагивают следующие этапы:

- отслеживание изменений,
- отправка изменений на все микросервисы,
- ожидание подтверждения доставки и применения обновлений,
- фиксирование новой версии метаданных и отправка на шлюз,
- подтверждение доставки и завершение процесса синхронизации.

Так как данные процессы затрагивают модификацию репозитория, то легко произвести проверку третьей гипотезы. Метаданные должны оставаться в консистентном и доступном для микросервисов состоянии даже при максимально возможном числе обновлений.

При проведении эксперимента в обоих моделях также фиксируется вспомогательная метрика: версия, с которой было обработано событие тем или иным микросервисом. Диаграмма последовательности, отображающая этот процесс показана на рисунке 43.

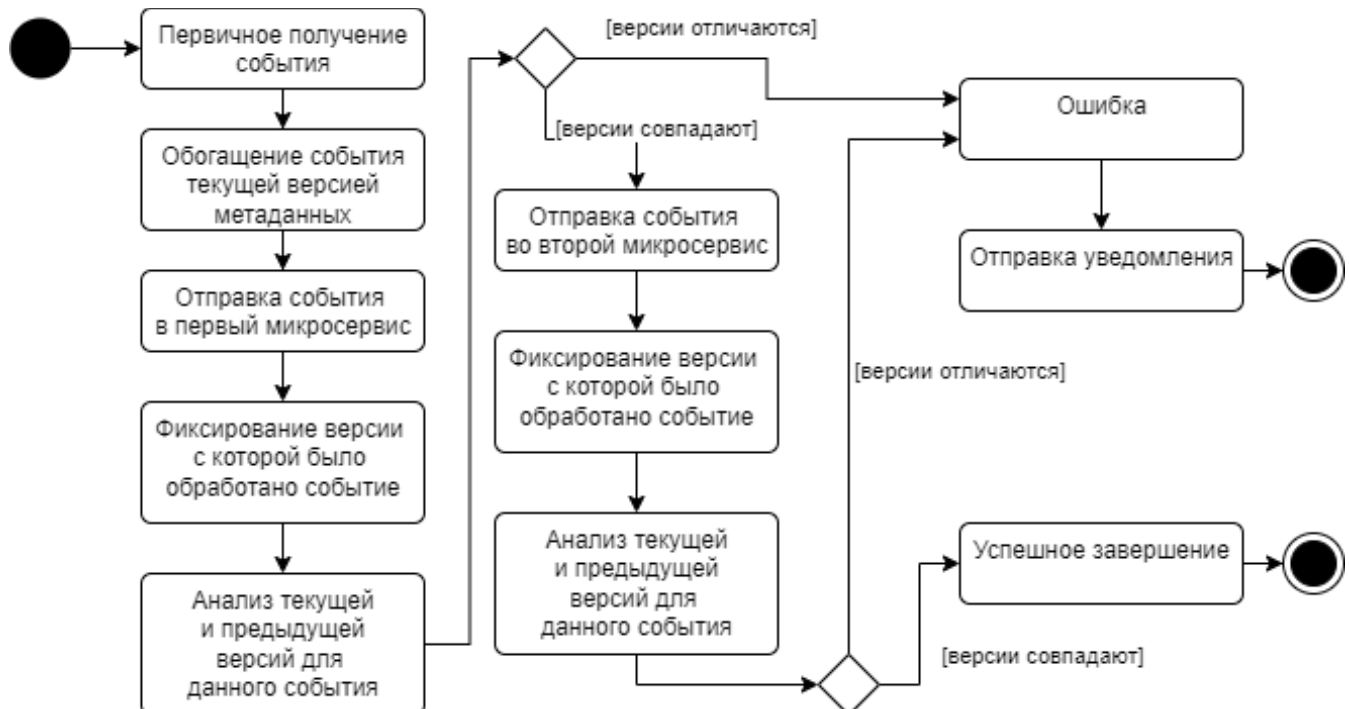


Рисунок 43 – Процесс фиксирования и проверки версий, используемых на каждом этапе обработки события

Благодаря данному функционалу можно создать правила оповещений для Alertmanager в Prometheus. Каждое правило содержит запрос на языке

PromQL, который периодически выполняется Prometheus. Если результаты запроса соответствуют заданным условиям, в данном случае будут найдены разные версии метаданных для одного события, то будет сгенерировано соответствующее оповещение. На следующем этапе Alertmanager обработает оповещение и может заняться их группировкой и маршрутизацией. Alertmanager предоставляет функционал отправки уведомлений через различные каналы связи, такие как электронная почта, мессенджеры и т.п.

Все метрические показатели при разных моделях эксперимента экспортируются с разными метками (labels). Метки в Prometheus служат для добавления контекста к метрикам, в данном случае, помимо источника данных, будет задан дополнительный атрибут, который соответствует номеру проводимого сценария. Это позволит эффективно систематизировать и анализировать данные, полученные в рамках разных сценариев эксперимента. Программный код для проведения эксперимента представлен в приложении Б.

4.4 Анализ и систематизация экспериментальных данных

Полученные в ходе эксперимента данные были успешно импортированы через Prometheus в Grafana, которая позволит визуализировать их для анализа.

Для анализа данных необходимо выбрать варианты представления собранных метрических показателей, которые позволят максимально эффективно отобразить важные тенденции и выявить узкие места разработанных алгоритмов и средств. Было выбрано два типа компонент: диаграмма временных рядов (Time Series) и панель статистики (Stat Panel).

Диаграмма временных рядов – это один из самых популярных и информативных типов диаграмм для мониторинга и анализа метрик в реальном времени. При помощи данной диаграммы можно наблюдать тенденции, пики или аномалии в данных на протяжении выбранного интервала времени. На оси абсцисс будут находиться временные отметки, а на оси ординат значения метрических показателей.

Панель статистики – это панель, которая представляет собой отображение последнего полученного значения. Также встроенные средства

предоставляют возможность дополнительного отображения хронологии изменений значений на временном ряду. Этот компонент особенно полезен для представления основных показателей производительности в компактном виде.

Для удобного представления всех компонентов был разработан Grafana Dashboard, который позволит на одном экране сразу отслеживать несколько диаграмм, которые размещаются в отдельных блоках. Каждый блок отвечает за визуализацию отдельно взятых частей ДС ИУС. Данные в эти блоки передаются при помощи средств через Prometheus.

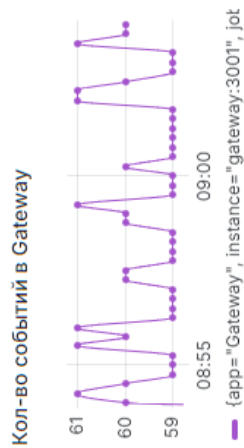
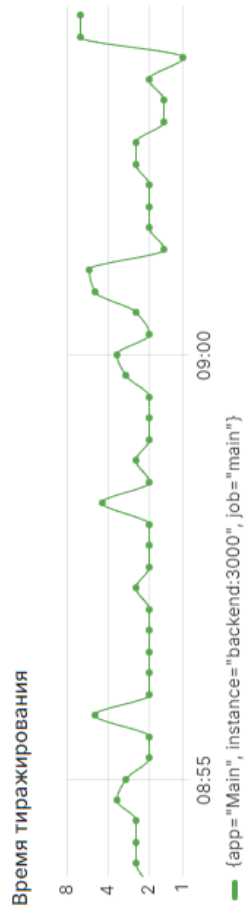
Было разработано 9 блоков, рассмотрим каждый из них:

- количество событий в Gateway, который отображает количество поступающих событий в шлюз в указанный момент времени,
- время тиражирования, которое отображает время, за которое происходит весь жизненный цикл тиражирования и синхронизации обновлённых метаданных,
- количество обновлений метаданных за минуту,
- общее количество обработанных событий, представленное отдельными диаграммами для обоих микросервисов,
- общее количество обработанных событий за минуту, представленное отдельными диаграммами для обоих микросервисов,
- среднее время обработки событий, представленное отдельными диаграммами для обоих микросервисов.

Для начала анализа сперва необходимо изучить стандартные показатели, которые измерялись при нормальных условиях проведения эксперимента (рисунок 44). Данные показатели были измерены при следующих условиях:

- равномерная генерация событий на протяжении минуты, каждое событие генерировалось один раз в одну секунду,
- равномерная генерация обновлений метаданных, каждое новое обновление происходило один раз в десять секунд.

▼ Gateway и программные средства тиражирования



▼ Microservice 1

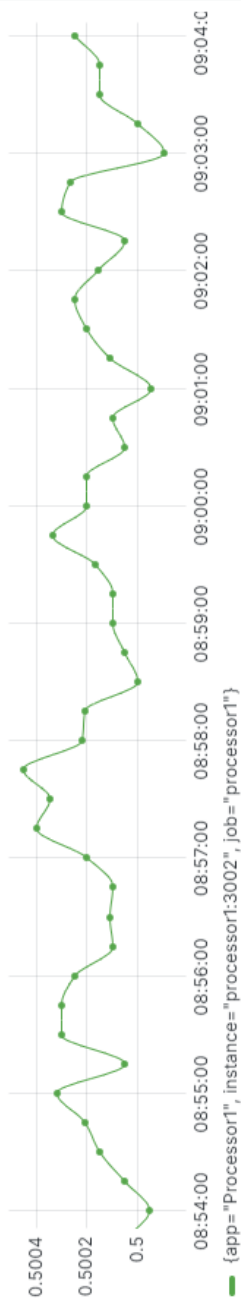
Общее количество обработанных событий

642

Количество обработанных событий (1 мин)

60

Среднее время обработки событий



▼ Microservice 2

Общее количество обработанных событий

639

Количество обработанных событий (1 мин)

60

Среднее время обработки событий

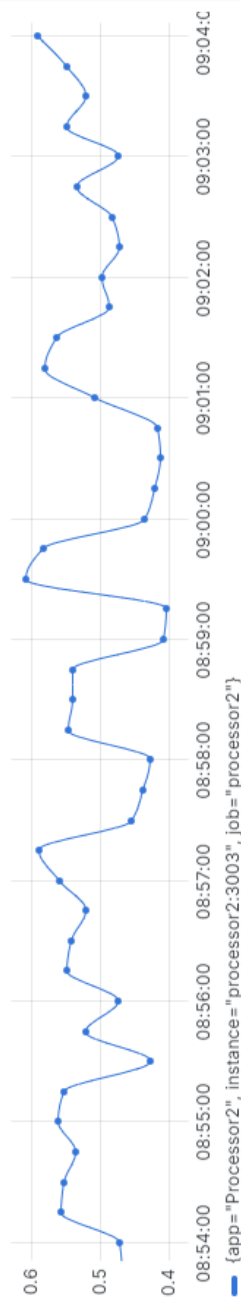


Рисунок 44 – Стандартные показатели

На диаграмме можно сразу заметить небольшие отклонения в данных на диаграммах временных рядов, например, когда показывается 59 событий в одну минуту, а 61 в следующую, при том, что в минуту генерируется ровно 60 событий. Системы мониторинга, такие как Prometheus, собирают данные в определенные интервалы времени, которые могут не всегда точно совпадать с границами каждой секунды, когда генерируется событие. Например, если время генерации события и время сбора данных не синхронизированы с точностью до микросекунд, то возможно возникновение ситуации, при которой одно событие фиксируется в предыдущем временном интервале, а следующее – уже в новом. Данные погрешности не несут в себе значимой проблемы, поэтому для временных рядов будут использованы средние показатели.

Показатели основных для сравнения метрик при стандартном сценарии использования представлены в таблице 4.

Таблица 4 – Стандартные показатели метрик

Название	Значение
Количество событий в Gateway	60 событий/мин
Время тиражирования	2,5 миллисекунды
Общее количество обновлений метаданных	6 обновлений/мин
Общее количество событий (за 10 мин)	600 событий
Среднее время обработки в первом микросервисе	500,1 миллисекунд
Среднее время обработки во втором микросервисе	500,1 миллисекунд

Время обработки событий на каждом микросервисе имеет одинаковую продолжительность, что является хорошим показателем. Данное время не должно изменять в зависимости от количества обновлений метаданных и количества поступающих событий.

Рассмотрим результаты эксперимента, полученные в ходе первого экспериментального сценария (рисунок 45).

На диаграмме можно отчётливо наблюдать, как частота поступления событий в шлюз линейно возрастает на протяжении 10 минут от 20 событий в минуту до 20000 событий в минуту.

Сведём итоговые показатели в таблицу и сравним их с показателями при обычном сценарии использования. Данное сравнение представлено в таблице 5.

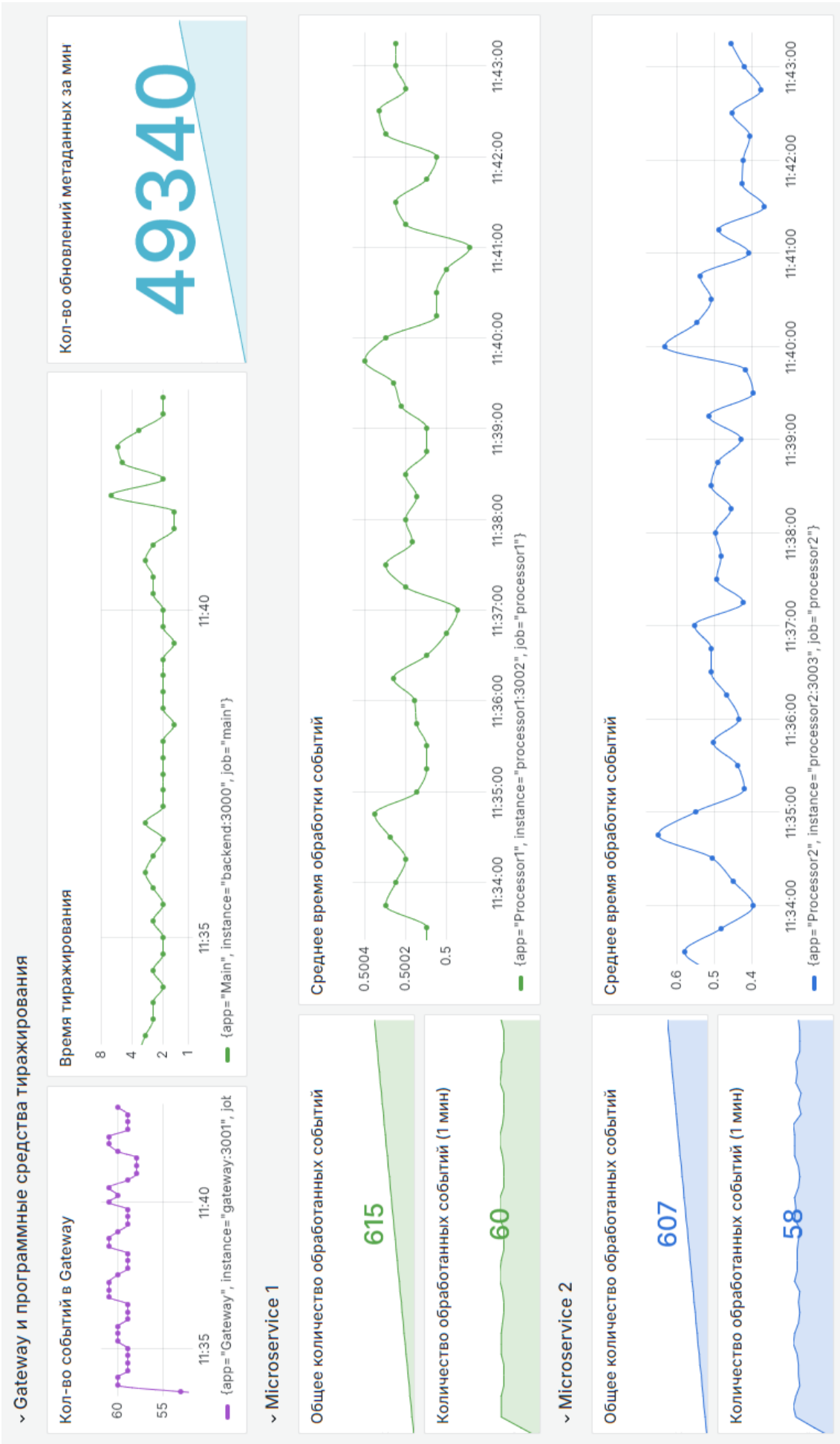
Таблица 5 – Показатели первого экспериментального сценария

Название	Стандартное значение	Экспериментальное
Количество событий в Gateway (последняя минута)	60 событий/мин	20000 событий/мин
Время тиражирования	2,5 миллисекунды	2,5 миллисекунды
Общее количество обновлений метаданных	6 обновлений/мин	6 обновлений/мин
Общее количество событий (за 10 мин)	600 событий	95000 событий
Среднее время обработки в первом микросервисе	500,1 миллисекунд	500,1 миллисекунд
Среднее время обработки во втором микросервисе	500,1 миллисекунд	500,1 миллисекунд

Исходя из полученных показателей можно сделать вывод о том, что изменение количества поступающих событий никаким образом не влияет на время тиражирования и синхронизации метаданных. Это связано с тем, что все процессы тиражирования и синхронизации инкапсулируются в отдельном Sidecar-контейнере, который никак не связан с процессами обработки поступающих событий.

Для проверки следующей гипотезы рассмотрим результаты, полученные в ходе второго экспериментального сценария (рисунок 46). Второй сценарий предполагает динамическое увеличение числа обновлений метаданных.

На диаграмме можно отчётливо наблюдать, как частота обновлений метаданных линейно возрастает на протяжении 10 минут до 50000 обновлений за минуту.



Сведём итоговые показатели в таблицу и сравним их с показателями при обычном сценарии использования. Данное сравнение представлено в таблице 6.

Таблица 6 – Показатели второго экспериментального сценария

Название	Стандартное значение	Экспериментальное
Количество событий в Gateway (последняя минута)	60 событий/мин	60 событий/мин
Время тиражирования	2,5 миллисекунды	2,5 миллисекунды
Общее количество обновлений метаданных	6 обновлений/мин	50000 обновлений/мин
Общее количество событий (за 10 мин)	600 событий	600 событий
Среднее время обработки в первом микросервисе	500,1 миллисекунд	500,1 миллисекунд
Среднее время обработки во втором микросервисе	500,1 миллисекунд	500,1 миллисекунд

Исходя из полученных показателей можно сделать вывод о том, что изменение количества обновлений метаданных никаким образом не влияет на время тиражирования и время обработки событий, не возникает никаких блокировок процессов обработки. Также после проведения всех сценариев эксперимента все события были обработаны с корректной версией метаданных на каждом этапе жизненного цикла.

Стабильные показатели производительности не исключают роста объема потребляемой памяти в локальных хранилищах микросервисов. Во время интенсивной генерации событий и увеличения числа обновлений метаданных, количество данных в локальных репозиториях может быть большим, так как в системе находится много событий с разной требуемой версией метаданных. После выхода события из системы, локальные хранилища оперативно очищаются от устаревшей версии.

Таким образом, несмотря на высокую отказоустойчивость и производительность системы, увеличение нагрузки приводит к более интенсивному использованию ресурсов Redis, что, в свою очередь, требует большего объема оперативной памяти. Однако, на данный момент времени,

это является небольшой проблемой по сравнению с необходимостью обеспечения высокой производительности и стабильности системы.

4.5 Выводы по разделу 4

В результате работы в данном разделе был спланирован и разработан эксперимент сразу по нескольким сценарием, чтобы изучить все возможные проблемы разработанных технологий. Эксперимент прошел удачно, данные были собраны для последующего анализа. В результате проведенного анализа можно уверенно сказать, что разработанные алгоритмы и программные средства показали себя с хорошей стороны, аномалий во время высоких нагрузок выявлено не было.

ЗАКЛЮЧЕНИЕ

В ходе магистерской работы были исследованы и разработаны модели, алгоритмы и программные средства управления метаданными в дискретно-событийных информационно управляющих системах, построенных на базе микросервисной программной платформы потоковой обработки и исполнения активных правил.

Была описана и разработана единая метамодель ДС ИУС, которая включает в себя метамодель внешних систем, событий и активных правил. Разработана единая архитектура метаданных и модель распределенного репозитория.

Были разработаны алгоритмы управления метаданными, включая конструирование, анализ, тиражирование и синхронизацию метаданных в компонентах микросервисной архитектуры ДС ИУС;

Были рассмотрены аспекты применения программных средств для тиражирования и синхронизации метаданных, в том числе, были детально описаны клиентские и серверные средства конструирования.

Для процессов тиражирования была выбрана технология Redis Pub/Sub. Был проведен эксперимент для более корректного обоснования выбора. Разработанные алгоритмы были реализованы с применением данной технологии.

Проведена экспериментальная оценка разработанных моделей, алгоритмов и программных средств управления метаданными. В результате проведенного эксперимента и анализа можно сделать вывод, что разработанные средства показали высокую эффективность.

Результаты апробации работы и презентация представлены в приложениях В и Г соответственно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Enterprise-Control System Integration, Part 1: Models and Terminology – Instrument Society of America, 1999. – 150 с.
2. Cassandras, C. G. Introduction to Discrete Event Systems / C. G. Cassandras, S. Lafortune. – Springer, 2008. – 772 с.
3. Unity in diversity, diversity in unity: Retrospective and prospective views on control of discrete event systems / R. Boel, X. R. Cao, G. Cohen, A. Giua, W. M. Wonham, J. H. Schuppen // J. Discrete Event Dynamic Systems: Theory and Application. – 2002. – Vol. 12, № 3. – P. 253–264.
4. Harel D. Statecharts: a Visual Formalism for Complex Systems. Science of Computer Programming. 1987. vol. 8. pp. 231–274.
5. Шибанов, С. В. Сервис потоковой обработки событий и исполнения активных правил / С. В. Шибанов, Я. С. Шлепнев // Математическое моделирование и суперкомпьютерные технологии: Труды XXI Международной конференции, Нижний Новгород, 22–26 ноября 2021 года. – Нижний Новгород: Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского, 2021. – С. 403-407.
6. Adaikkalavan R. Active Authorization Rules for Enforcing Role-Based Access Control and its Extensions / R. Adaikkalavan, S. Chakravarthy // 21st International Conference on Data Engineering Workshops – Tokyo : Изд-во IEEE, 2005. – С. 220 – 222.
7. Шибанов, С. В. Мета модель сервиса потоковой обработки событий / С. В. Шибанов, Я. С. Шлепнев, А. С. Гусаров // Методы, средства и технологии получения и обработки измерительной информации («Шляндинские чтения - 2022») : материалы XIV Международной научно-технической конференции с элементами научной школы и конкурсом научно-исследовательских работ для обучающихся и молодых ученых, Пенза, 24–26 октября 2022 года. – Пенза: Пензенский государственный университет, 2022. – С. 240-243.
8. David C. Luckham Event Processing and the Survival of the Modern Enterprise / David C. Luckham – New York, Изд-во Wiley, 2011. – 288 с.

9. Big Data. Principles and Paradigms / Rajkumar Buyya, Amir Vahid Dastjerdi, Rodrigo N. Calheiros – Melbourne, Изд-во Morgan Kaufmann Publishers, 2016. – 494 с.
10. Cassandra, C. G.; Lafortune, S. Introduction to Discrete Event Systems / Cassandra, C. G.; Lafortune, S. – New York, Изд-во Springer, 2008. – 770 с.
11. Psaltis Streaming Data. Understanding the real-time pipeline / Andrew G. Psaltis – New York, Изд-во Manning, 2017. – 216 с.
12. Paschke A., Kozlenkov A. Rule-Based Event Processing and Reaction Rules / Adrian Paschke, Alexander Kozlenkov // Rule Interchange and Applications – Las Vegas : Изд-во Springer, 2009. – С. 53 – 66.
13. Шибанов С. В., Скоробогатько А. А., Э. В. Лысенко. Интегрированная модель активных правил / С. В. Шибанов, А. А. Скоробогатько, Э. В. Лысенко // Математическое и программное обеспечение систем в промышленной и социальной сферах: междунар. сб. науч. трудов – Магнитогорск : Изд-во Магнитогорск. гос. техн. ун-та им. Г.И. Носова, 2011. – С. 41 – 46.
14. Федотов А.М., Леонова Ю.В. Требования к прототипу системы управления информационными ресурсами в распределенных информационных системах поддержки научных исследований // Вычислительные технологии – Новосибирск, Изд-во Институт вычислительных технологий СО РАН, 2018. – Т. 23, №5. С. 82 – 109.
15. Лядова Л. Мета моделирование и многоуровневые метаданные как основа технологии создания адаптируемых информационных систем / Людмила Лядова // Advanced Studies in Software and Knowledge Engineering – Varna, Изд-во Institute of Information Theories and Applications FOI ITHEA, 2008. – С. 125 – 132.
16. Richardson C. Microservices Patterns / Chris Richardson – New York, Изд-во Manning, 2018. – 520 с.

17. Pattern: Database per service // Microservice Architecture [Электронный ресурс]. URL: <https://microservices.io/patterns/data/database-per-service.html> (дата обращения 05.03.2023)

18. Ньюмен С. От монолита к микросервисам. – СПб.: БХВ-Петербург, 2021. – 272 с.

19. What is Strict Consistency vs Eventual Consistency? // Cohesity [Электронный ресурс]. URL: <https://www.cohesity.com/blogs/strict-vs-eventual-consistency> (дата обращения 12.12.2023)

20. Picking an Active-Active Geo Distribution Strategy: Comparing Merge Replication and CRDT // InfoQ [Электронный ресурс]. URL: <https://www.infoq.com/articles/database-merge-replication-crdt> (дата обращения 20.11.2023)

21. Concurrency and Automatic Conflict Resolution // Dev.to [Электронный ресурс]. URL: <https://dev.to/frosnerd/concurrency-and-automatic-conflict-resolution-4i9o> (дата обращения 20.11.2023)

22. Top 5 Ways to Implement Real-Time Rich Text Editor [Электронный ресурс]. URL: <https://exaspark.medium.com/top-5-ways-to-implement-real-time-rich-text-editor-ranked-by-complexity-3bc26e3c777f> (дата обращения 20.11.2023)

23. Гусаров А.С. Серверные программные средства управления метаданными для сервиса потоковой обработки событий : выпускная квалификационная работа бакалавра : 09.03.04 – Пенза, 2022. – 122 с.

24. PostgreSQL [Электронный ресурс]. URL: <https://www.postgresql.org> (дата обращения 01.04.2024).

25. Redis [Электронный ресурс]. URL: <https://redis.io> (дата обращения 01.04.2024).

26. Sequelize ORM [Электронный ресурс]. URL: <https://sequelize.org> (дата обращения 01.04.2024).

27. React [Электронный ресурс]. URL: <https://react.dev> (дата обращения 01.04.2024).

28. Redux Toolkit [Электронный ресурс]. URL: <https://redux-toolkit.js.org> (дата обращения 01.04.2024).

29. Гусаров, А.С. Использование No-Code подхода при управлении метаданными сервиса потоковой обработки событий и исполнения активных правил / А. С. Гусаров, С. В. Шибанов, Я. С. Шлепнев // Информационные технологии в науке и образовании. Проблемы и перспективы : материалы X Ежегодной всероссийской межвузовской научно-практической конференции – Пенза : Изд-во ПГУ, 2023. – С. 368 – 374.

30. Шибанов, С. В. Портал управления метаданными сервиса потоковой обработки событий и исполнения активных правил / С. В. Шибанов, Я. С. Шлепнев, А. С. Гусаров // Информационные технологии в моделировании и управлении: подходы, методы, решения : материалы VI Всероссийской научной конференции с международным участием – Тольятти : Изд-во ТГУ, 2023. – С. 577 – 587.

31. Брендан Бернс: Распределенные системы. Паттерны проектирования – СПб.: Питер, 2019. – 224 с.

32. Prometheus [Электронный ресурс]. URL: <https://prometheus.io> (дата обращения 01.04.2024).

33. Grafana Labs [Электронный ресурс]. URL: <https://grafana.com> (дата обращения 01.04.2024).

ПРИЛОЖЕНИЕ А

(справочное)

Алгоритмы и программные средства управления метаданными в
дискретно-событийных динамических информационно управляющих
системах

Глоссарий

Дискретно-событийная информационно управляющая система (ДС ИУС): система, которая характеризуется дискретностью пространства состояний и изменением состояний во времени при помощи асинхронных дискретных событий.

Event-Condition-Action (ECA): нотация описания активных правил, в соответствии с которой правилом является тройка объектов – событие предметной области, условие, предназначенное для проверки параметров события, и действие, описывающее операции, совершаемые в случае истинности условия. Событие и условие в контексте активных правил иногда называют Ситуацией.

Активное правило: объект специального вида, задающий реакцию на событие.

Обработка сложных событий (Complex Event Processing, CEP): это подход к анализу событий в реальном времени, с выявлением их взаимосвязей, причин и следствий.

Сложное событие: событие, полученное из одного или нескольких других событий.

Событие: любой факт или действие, которое происходит с наблюдаемым объектом.

Event Stream Processing (ESP): технология потоковой обработки событий, которые поступают в систему.

ACID (атомарность, согласованность, изоляция, долговечность): набор требований к транзакционной системе, обеспечивающий наиболее надёжную и предсказуемую её работу.

ПРИЛОЖЕНИЕ Б

(рекомендуемое)

Алгоритмы и программные средства управления метаданными в дискретно-событийных динамических информационно управляющих системах

Программный код для проведения эксперимента

Листинг Б.1 – Конфигурационный модуль главного компонента ИУС:

```
@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: process.env.POSTGRES_HOST,
      port: parseInt(process.env.POSTGRES_PORT),
      username: process.env.POSTGRES_USER,
      password: process.env.POSTGRES_PASSWORD,
      database: process.env.POSTGRES_DB,
      entities: ENTITIES,
      synchronize: process.env.TYPEORM_SYNC === 'true',
    }),
    CacheModule.register({
      isGlobal: true,
    }),
    HttpModule.register({
      timeout: 5000,
      retries: 5,
    }),
    ClientsModule.register([
      {
        name: 'REDIS_PUB_SUB_SERVICE',
        transport: Transport.REDIS,
        options: {
          host: process.env.REDIS_PUB_SUB_HOST,
          port: parseInt(process.env.REDIS_PUB_SUB_PORT),
        },
      },
      {
        name: 'GRPC_SERVICE',
        transport: Transport.GRPC,
        options: {
          package: 'metadata',
          protoPath: join(__dirname, 'assets/metadata.proto'),
          url:
            '${process.env.GRPC_GATEWAY_HOST}:${process.env.GRPC_GATEWAY_PORT}',
        },
      },
    ]),
    ScheduleModule.forRoot(),
    MetricsModule,
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

Листинг Б.2 – Контроллер главного компонента ИУС

```
@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}
```

```

private readonly logger = new Logger(AppController.name);

@Cron(CronExpression.EVERY_10_SECONDS, { name:
'UpdateMetadata' })
async handleCron() {
    // Инициация процесса обновления метаданных
    // В данном случае, обновление версии метаданных для
эксперимента
    this.logger.debug('Updating metadata version');
    const currentVersion = await
this.appService.updateMetadataVersion();

    // В данном случае, отправляется лишь обновлённая версия
    this.logger.debug('Sending metadata (v: ${currentVersion})
to microservices');
    await
this.appService.sendMetadataToMicroservices(currentVersion);
}

@EventPattern('metadata:update:confirmed')
async handleDataEventRedis(@Payload() data: any, @Ctx()
context: RedisContext) {
    await
this.appService.handleMetadataUpdateConfirmation(data);
    return;
}
}

```

Листинг Б.3 – Основной сервис главного компонента ИУС

```

@Injectable()
export class AppService {
    constructor(
        @Inject(CACHE_MANAGER) private cacheManager: Cache,
        @Inject('REDIS_PUB_SUB_SERVICE') private clientRedis:
ClientProxy,
        @Inject('GRPC_SERVICE') private clientGRPC: ClientGrpc,
        private readonly metricService: MetricsService,
    ) {}

    private gMetadataService: MetadataServiceType;

    private readonly logger = new Logger(AppService.name);

    private readonly microservices = ['processor1', 'processor2'];
    private readonly acknowledgements: Record<
        string,
        Record<string, { status: 'pending' | 'success' | 'failed';
time: number }>
    > = {};

    onModuleInit() {

```

```

    this.gMetadataService =
this.clientGRPC.getService<MetadataServiceType>('MetadataService
');
}

/**
 * Обновление текущей версии метаданных
 */
async updateMetadataVersion(): Promise<string> {
    const currentVersion = +(await
this.cacheManager.get('metadata:version')) || 0;
    const newVersion = currentVersion + 1;
    await this.cacheManager.set('metadata:version', newVersion,
0);
    return newVersion.toString();
}

/**
 * Отправка метаданных в микросервисы
 */
async sendMetadataToMicroservices(currentVersion: string) {
    const metadataEntity: MetadataUpdateType = {
        target: 'main',
        version: currentVersion,
        startTime: Date.now(),
    };

    // Инициализация статуса для всех микросервисов как
'pending'
    this.acknowledgements[currentVersion] = {};
    for (const microservice of this.microservices) {
        this.acknowledgements[currentVersion][microservice] = {
status: 'pending', time: Date.now() };
    }

    // Отправка обновлений
    this.clientRedis.emit('metadata:update', metadataEntity);
}

/**
 * Отслеживание статуса обновления метаданных
 */
async handleMetadataUpdateConfirmation(data: {
    version: string;
    microservice: string;
    status: 'success' | 'failed';
}) {
    // Время получения подтверждения о доставке
    const confirmationTime = Date.now() -
this.acknowledgements[data.version][data.microservice].time;
    const { version, microservice, status } = data;
    this.logger.log(

```

```

        'Received metadata update confirmation from
        ${microservice} for version ${version}. ${confirmationTime} ms',
    );
    this.acknowledgements[version][microservice] = {
        status,
        time: confirmationTime,
    };

    // Проверка статуса обновления
    if (Object.values(this.acknowledgements[version]).every((el)
=> el.status === 'success')) {
        this.logger.debug('Metadata version ${version} has been
successfully updated');
        const confirmationTimeMax =
Math.max(...Object.values(this.acknowledgements[version]).map((e
l) => +el.time));
        this.logger.debug('Max confirmation time:
${confirmationTimeMax} ms');

        // Отправка корректной новой версии в Gateway
        this.logger.debug('Sending metadata version ${version} to
gateway');
        const { confirmationSeconds } = await
this.gMetadataService
            .updateMetadataVersion({ version, startTime: Date.now()
})
            .toPromise();
        this.logger.debug(
            'Metadata version ${version} has been successfully
processed on Gateway in ${confirmationSeconds} ms',
        );

        const allTimeMetadataUpdateMs = +confirmationSeconds +
confirmationTimeMax;
        this.logger.debug('Total time for metadata update:
${allTimeMetadataUpdateMs} ms');
        // Grafana

this.metricService.recordMetadataUpdateCompletion(allTimeMetadat
aUpdateMs);
    }
}
}

```

Листинг Б.4 – Контроллер шлюза

```

@Controller()
export class AppController {
    constructor(
        private readonly appService: AppService,
        private readonly metricService: MetricsService,
    ) {}

    private readonly logger = new Logger(AppController.name);

```

```

    @MessagePattern('gateway:data')
    handleDataEvent2(@Payload() data: any, @Ctx() context:
MqttContext) {
        this.logger.log('Received data: ${JSON.stringify(data)} on
topic: ${context.getTopic()}');
        // Передача данных для обработки
        // В эксперименте используется последовательная передача
        // Gateway -> Processor1 -> Processor2

        // Передача данных в Processor1
        const startTime = Date.now();
        this.appService.sendDataToProcessor1(data);
        this.metricService.recordGatewayEvent(Date.now() -
startTime);
    }

    @GrpcMethod('MetadataService', 'UpdateMetadataVersion')
    async updateMetadataVersion(data: { version: string;
startTime: number }) {
        // Обновление метаданных
        this.logger.log('Received metadata update:
${JSON.stringify(data)}');
        await this.appService.updateMetadataVersion(data.version);
        const processingTimeMs = Date.now() - data.startTime;
        return { confirmationSeconds: processingTimeMs };
    }
}

```

Листинг Б.5 – Сервис шлюза

```

@Injectable()
export class AppService {
    constructor(
        @Inject('MQTT_CLIENT') private readonly clientMqtt:
ClientMqtt,
        @Inject(CACHE_MANAGER) private cacheManager: Cache,
    ) {}

    private readonly logger = new Logger(AppService.name);

    /**
     * Отправка данных на обработку в Processor1
     */
    async sendDataToProcessor1(data: any) {
        try {
            this.logger.log('Sending data to Processor1:
${JSON.stringify(data)}');
            // Указание версии с которой нужно обработать событие
            (обогащение события)
            data['metadataVersion'] = (await
this.cacheManager.get('metadata:version')) || 0;
            // Публикация данных на специфическом топике, который
слушает Processor1

```

```

        this.clientMqtt.emit('processor1:data', data);
    } catch (error) {
        this.logger.error('Failed to send data to Processor1',
error);
    }
}

async updateMetadataVersion(version: string) {
    // Обновление метаданных
    await this.cacheManager.set('metadata:version', version, 0);
}
}

```

Листинг Б.6 – Конфигурационный модуль первого микросервиса

```

@Module({
  imports: [
    ClientsModule.register([
      {
        name: 'MQTT_CLIENT',
        transport: Transport.MQTT,
        options: {
          url: 'mqtt://mqtt-broker:1883',
        },
      },
    ]),
    CacheModule.register<RedisOptions>({
      isGlobal: true,
      store: redisStore,
      host: process.env.REDIS_LOCAL_HOST,
      port: parseInt(process.env.REDIS_LOCAL_PORT),
    }),
    MetricsModule,
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}

```

Листинг Б.7 – Контроллер первого микросервиса

```

@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  private readonly logger = new Logger(AppController.name);

  @MessagePattern('processor1:data')
  handleDataEvent2(@Payload() data: any, @Ctx() context:
MqttContext) {
    this.logger.log('Received data: ${JSON.stringify(data)} on
topic: ${context.getTopic()}');
    // Обработка данных
    this.appService.processEvent(data);
  }
}

```

```
}
```

Листинг Б.8 – Сервис первого микросервиса

```
@Injectable()
export class AppService {
  constructor(
    @Inject(CACHE_MANAGER) private cacheManager: Cache,
    @Inject('MQTT_CLIENT') private readonly clientMqtt:
ClientMqtt,
    private readonly metricsService: MetricsService,
  ) {}

  private readonly logger = new Logger(AppService.name);

  /**
   * Обработка данных в Processor1
   */
  async processEvent(eventData: any) {
    this.metricsService.recordEventStart();
    const startTime = Date.now();
    try {
      // Имитация обработки события в течение 500 мс
      await new Promise((resolve) => setTimeout(resolve, 500));
      const processingTime = (Date.now() - startTime) / 1000;

      // Получение версии метаданных с которой нужно обработать
событие
      const metadata = await
this.cacheManager.get('metadata:version:${eventData['metadataVer
sion']}');
      if (!metadata) {
        throw new Error('Metadata not found');
      }

      // Запись метрик
      this.metricsService.recordEventCompletion(processingTime);
      this.logger.log('Processed event:
${JSON.stringify(eventData)}');

      // Mock: после обработки данных в Processor1, данные
решено передать в Processor2
      this.sendDataToProcessor2(eventData);
    } catch (error) {
      this.logger.error('Error processing event', error);
    }
  }

  /**
   * Отправка данных на обработку в Processor1
   */
  sendDataToProcessor2(data: any) {
    try {
```

```

        this.logger.log('Sending data to Processor2:
${JSON.stringify(data)}');
        // Публикация данных на специфическом топике, который
        слушает Processor1
        this.clientMqtt.emit('processor2:data', data);
    } catch (error) {
        this.logger.error('Failed to send data to Processor2',
error);
    }
}
}
}

```

Листинг Б.9 – Контроллер Sidecar-контейнера

```

@Controller()
export class AppController {
    constructor(private readonly appService: AppService) {}

    private readonly logger = new Logger(AppController.name);

    @EventPattern('metadata:update')
    handleDataEventRedis(@Payload() data: any, @Ctx() context:
RedisContext) {
        this.logger.log('Received data: ${JSON.stringify(data)} on
topic: ${context.getChannel()}');
        // Обновление метаданных
        this.appService.updateMetadata(data);
        return;
    }
}

```

Листинг Б.10 – Сервис Sidecar-контейнера

```

@Injectable()
export class AppService {
    constructor(
        @Inject(CACHE_MANAGER) private cacheManager: Cache,
        @Inject('REDIS_PUB_SUB_SERVICE') private clientRedis:
ClientProxy,

        private readonly metricsService: MetricsService,
    ) {}

    onModuleInit() {
        this.cacheManager.set('metadata:version:0', {}, 0);
    }

    async updateMetadata(data: any) {
        // Обновление метаданных
        this.metricsService.recordReceivedMetadataUpdate();
        await
this.cacheManager.set('metadata:version:${data.version}', {},
0);

        // Обновление метаданных завершено
    }
}

```

```

    this.metricsService.recordProcessedMetadataUpdate();
    this.emitConfirmation(data.version, 'success');

    // Обновление метаданных завершилось ошибкой
    // this.metricsService.recordFailedMetadataUpdate();
  }

  async emitConfirmation(version: string, status: 'success' |
'failed') {
    this.clientRedis.emit('metadata:update:confirmed', {
      version,
      microservice: process.env.MICROSERVICE_NAME,
      status,
    });
  }
}

```

Листинг Б.11 – Код генератора событий

```

import mqtt from 'mqtt';

const clientId = `mqtt_${Math.random().toString(16).slice(3)}`;

const client = mqtt.connect('mqtt://mqtt-broker:1883', {
  clientId,
  clean: true,
  connectTimeout: 4000,
  reconnectPeriod: 1000,
});

const topic = 'gateway:data'; // Тема, куда будут отправляться
события
const message = JSON.stringify({ temperature: 22.4, status: 'ok'
}); // Пример сообщения

client.on('connect', () => {
  console.log('Connected');
  setInterval(() => {
    client.publish(topic, message, { qos: 0, retain: false },
(error) => {
      if (error) {
        console.error('Publish error: ${error}');
      }
    });
  }, 1000); // Отправка сообщения каждую секунду
});

client.on('error', (err) => {
  console.error('Connection error: ', err);
  client.end();
});

```

ПРИЛОЖЕНИЕ В

(рекомендуемое)

Алгоритмы и программные средства управления метаданными в дискретно-событийных динамических информационно управляющих системах

Результаты апробации работы

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
МИНИСТЕРСТВО ОБРАЗОВАНИЯ ПЕНЗЕНСКОЙ ОБЛАСТИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ



НЕДЕЛЯ
НАУКИ 2023

ДИПЛОМ

НАГРАЖДАЕТСЯ

ГУСАРОВ АЛЕКСАНДР СЕРГЕЕВИЧ
РУКОВОДИТЕЛЬ: ШИБАНОВ СЕРГЕЙ ВЛАДИМИРОВИЧ

ПОБЕДИТЕЛЬ СЕКЦИИ «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ
АВТОМАТИЗИРОВАННЫХ СИСТЕМ И ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ»

X ВСЕРОССИЙСКОЙ
НАУЧНО-ПРАКТИЧЕСКОЙ КОНФЕРЕНЦИИ
«ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ
В НАУКЕ И ОБРАЗОВАНИИ.
ПРОБЛЕМЫ И ПЕРСПЕКТИВЫ»

РЕКТОР ПГУ



23 г.

А. Д. Гуляков

Рисунок В.1 – Диплом победителя I степени в конкурсе научно-исследовательских докладов в рамках X ежегодной всероссийской межвузовской научно-практической конференция «Информационные технологии в науке и образовании. Проблемы и перспективы.»



Рисунок В.2 – Диплом победителя I степени в конкурсе научно-исследовательских докладов в рамках XIV международной научно-технической конференции с элементами научной школы и конкурсом научно-исследовательских работ «Шляндинские чтения – 2022»



Рисунок В.3 – Сертификат участника в конкурсе научно-исследовательских докладов в рамках IX международной научно-практической конференции молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук»



Рисунок В.4 – Сертификат участника в конкурсе научно-исследовательских докладов в рамках всероссийской научно-практической конференции молодых ученых «Наука, инновации и технологии: от идей к внедрению»



Рисунок В.5 – Диплом победителя I степени в конкурсе научно-исследовательских докладов в рамках XX международной научно-технической конференции «Новые информационные технологии и системы»



Рисунок В.6 – Сертификат участника в региональном конкурсе научно-исследовательских работ обучающихся «Научный прорыв»



Рисунок В.7 – Сертификат участника в конкурсе научно-исследовательских докладов в рамках XXXV всероссийской с международным участием научно-технической конференции, посвященной 80-летию Пензенского государственного университета «Проблемы автоматизации и управления в технических системах»



ДИПЛОМ | DIPLOMA

НАГРАЖДАЕТСЯ | AWARDED TO

Тусаров Александр Сергеевич

Магистратура, 2 курс, 09.04.01 Информатика и вычислительная техника /
Прикладной искусственный интеллект, кафедра "Вычислительная техника",
факультет вычислительной техники, Пензенский государственный университет

лауреат 3 степени конкурса

"30.12.2023 — Международный конкурс «Студент года — 2023»", проводимого 30
декабря 2023 года по инициативе проекта Intercllover.ru, г. Нижний Новгород

Номинация: Молодой исследователь

Научный руководитель: Шибанов Сергей Владимирович, Кандидат
технических наук, доцент кафедры «МО и ПЭВМ», Пензенский государственный
университет

Красн
Краснова Н.А. | Krasnova N.
руководитель НОО
"Профессиональная наука",
председатель конкурсного жюри |
Head of the "Professional Science"
Chairman of the Competition Jury

Канаева
Канаева Ю.О. | Kanaeva Yu.
зам. руководителя НОО
"Профессиональная наука",
секретарь конкурсного жюри |
Deputy Head of the "Professional
Science", Secretary of the Jury

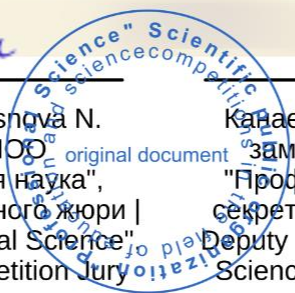


Рисунок В.8 – Диплом лауреата III степени в международном конкурсе
«Студент года – 2023», проводимого по инициативе проекта Intercllover



Рисунок В.9 – Диплом победителя II степени в XXV международном конкурсе научно-исследовательских работ «Путь в науку – 2022»

ПРИЛОЖЕНИЕ Г

(рекомендуемое)

Алгоритмы и программные средства управления метаданными в дискретно-событийных динамических информационно управляющих системах

Презентация

АЛГОРИТМЫ И ПРОГРАММНЫЕ СРЕДСТВА УПРАВЛЕНИЯ МЕТАДААННЫМИ В ДИСКРЕТНО-СОБЫТИЙНЫХ ДИНАМИЧЕСКИХ ИНФОРМАЦИОННО УПРАВЛЯЮЩИХ СИСТЕМАХ

Докладчик:

студент гр. 22ВВИм2 Гусаров А.С.

Научный руководитель:

к.т.н., доцент кафедры
«МОиПЭВМ» Шибанов С.В.

Рисунок Г.1 – Слайд презентации 1

Реагирующие дискретно-событийные динамические информационно управляющие системы

Характеризуются дискретностью пространства состояний и изменение состояний во времени производится асинхронными дискретными событиями

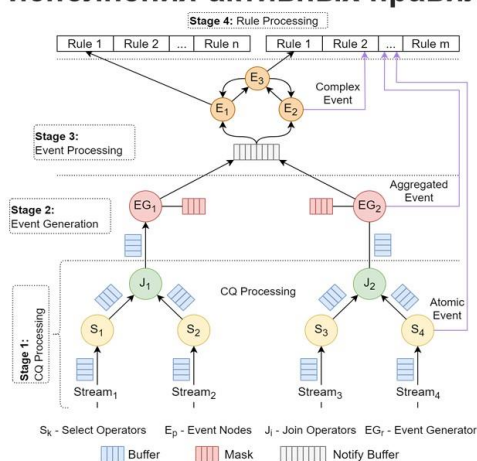
Требования:

- 1 Механизмы управления и обработки информации должны работать **в реальном масштабе времени**
- 2 **Нулевое** время простоя
- 3 Большая часть управляющих воздействий должна вырабатываться **без участия пользователя**
- 4 Умение обрабатывать **нестандартные** и **аварийные** ситуации

2/43

Рисунок Г.2 – Слайд презентации 2

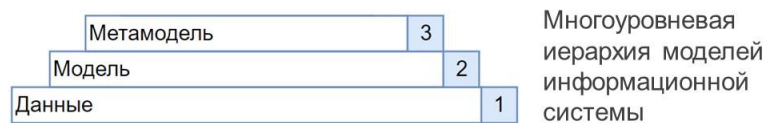
ДС ИУС на основе потоковой обработки событий и исполнения активных правил



3/43

Рисунок Г.3 – Слайд презентации 3

Применение метамоделирования для проектирования и реализации систем потоковой обработки событий с применением активных правил



4/43

Рисунок Г.4 – Слайд презентации 4

Актуальность



5/43

Рисунок Г.5 – Слайд презентации 5

Объект и предмет исследования

- 1 **Объектом исследования** являются процессы управления метаданными в системах потоковой обработки событий и исполнения активных правил на основе обобщенной метамодели.
- 2 **Предметом исследования** является обобщенная метамодель систем потоковой обработки событий, алгоритмы и программные средства управления и анализа метаданных

6/43

Рисунок Г.6 – Слайд презентации 6

Цель и задачи

Цель

Исследование и разработка моделей, алгоритмов и программных средств управления метаданными в ДС ИУС, построенных на базе микросервисной программной платформы потоковой обработки и исполнения активных правил

1. Разработка обобщенной метамодели ДС ИУС, функционирующей на базе микросервисной архитектуры с применением технологии потоковой обработки событий и механизма активных правил
2. Разработка единой архитектуры распределенного репозитория метаданных для ДС ИУС, функционирующей на базе микросервисной архитектуры с применением технологии потоковой обработки событий и механизма активных правил
3. Разработка алгоритмов управления метаданными ДС ИУС на этапах жизненного цикла, включая конструирование, анализ, тиражирование и синхронизацию метаданных
4. Разработка программных средств управления метаданными с применением low-code и no-code подходов
5. Экспериментальная оценка и анализ предлагаемых решений

7/43

Рисунок Г.7 – Слайд презентации 7

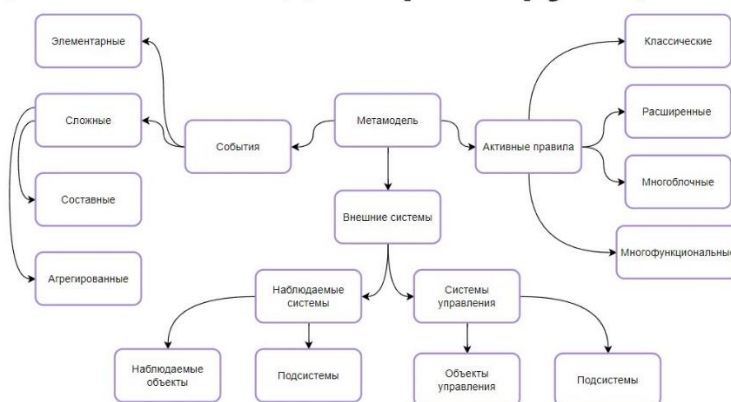
Научная новизна, оригинальность идей

- 1 Предложена **обобщенная метамодель** ДС ИУС на основе теоретико-множественного подхода, описывающая расширенный набор классов событий и активных правил.
- 2 Разработана **единая архитектура распределенного репозитория метаданных**, ориентированная на микросервисную архитектуру ДС ИУС, применение потоковой обработки событий и активных правил
- 3 Разработаны **алгоритмы тиражирования и синхронизации метаданных** в локальных репозиториях ДС ИУС, соответствующие требованиям скорости, надежности и восстановления состояния метаданных в случаях сбоев

8/43

Рисунок Г.8 – Слайд презентации 8

Единая метамодель реагирующей ИУС



9/43

Рисунок Г.9 – Слайд презентации 9

Метамоделль целевых систем

Множество целевых систем:

$$S = \{S_1, S_2, \dots, S_n\}.$$

Каждая система S_i может состоять из множества подсистем $Ssub_i$:

$$Ssub_i = \{Ssub_i^1, Ssub_i^2, \dots, Ssub_i^p\}.$$

Каждая система S_i и ее подсистемы $Ssub_i$ могут содержать множество объектов мониторинга и управления:

$$Om_i = \{Om_i^1, Om_i^2, \dots, Om_i^k\},$$

$$Oc_i = \{Oc_i^1, Oc_i^2, \dots, Oc_i^l\}.$$

Объект мониторинга одновременно может являться объектом управления:

$$Om \cap Oc \neq \emptyset.$$

10/43

Рисунок Г.10 – Слайд презентации 10

Метамоделль событий

Элементарное событие:
 $e_primitive = \langle PEid, Context, S, T, t \rangle$

Агрегированное событие:
 $e_aggregated = \langle AEid, Context, Q, T, tb, te \rangle$

Составное событие:
 $e_composite = \langle CEid, Context, C, T, tb, te \rangle$



11/43

Рисунок Г.11 – Слайд презентации 11

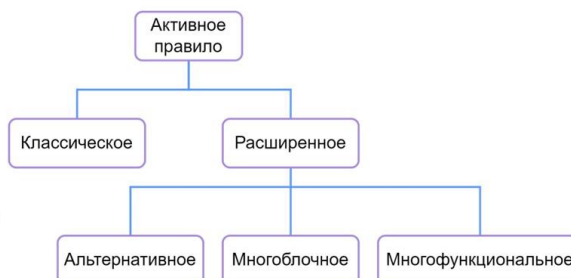
Метамоделль активных правил

Классическое АП:
 $r = \langle e, c, a \rangle$

Альтернативное АП:
 $r = \langle e, c, \{a_{base}, a_{alt}\} \rangle$

Многоблочное АП:
 $r = \langle \{e_1, c_1, a_1\}, \dots, \{e_n, c_n, a_n\} \rangle$

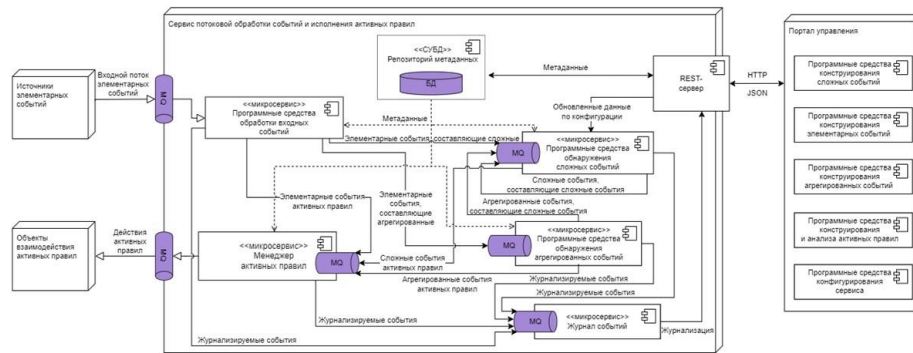
Многофункциональное АП:
 $r = \langle e, c, \{a_1, \dots, a_k\} \rangle$



12/43

Рисунок Г.12 – Слайд презентации 12

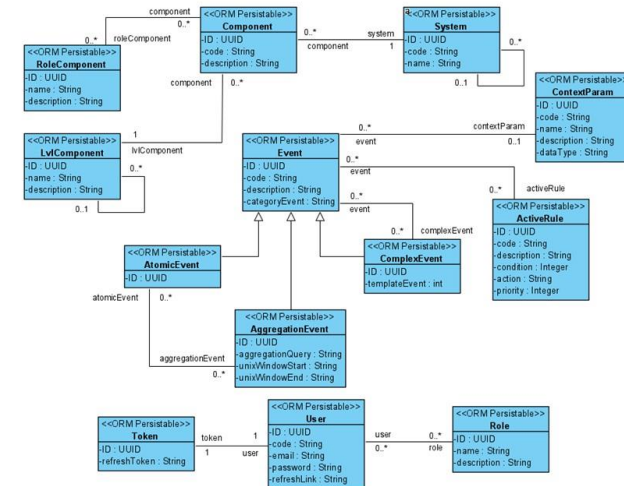
Архитектура сервиса



13/43

Рисунок Г.13 – Слайд презентации 13

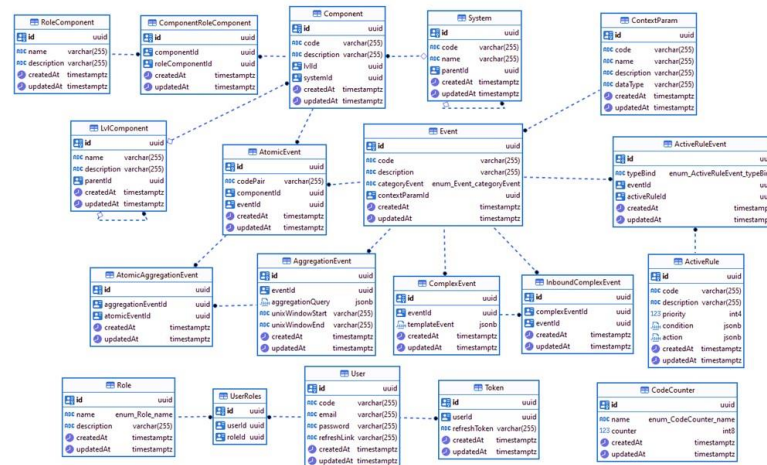
Объектная модель репозитория метаданных



14/43

Рисунок Г.14 – Слайд презентации 14

Реляционная схема репозитория метаданных



15/43

Рисунок Г.15 – Слайд презентации 15

Основные задачи управления метаданными

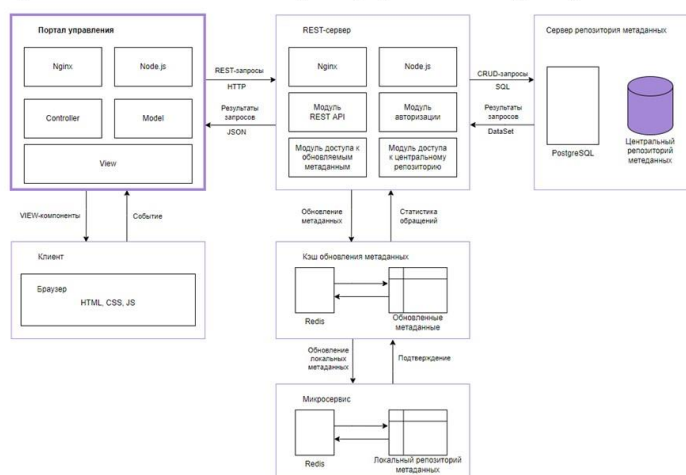
- 1 Создание и подготовка метаданных конфигурирования и управления
- 2 Конструирование метаданных событий и активных правил
- 3 Оперативное обновление метаданных в локальных репозиториях
- 4 Установка ассоциаций между объектами метаданных
- 5 Анализ метаданных



16/43

Рисунок Г.16 – Слайд презентации 16

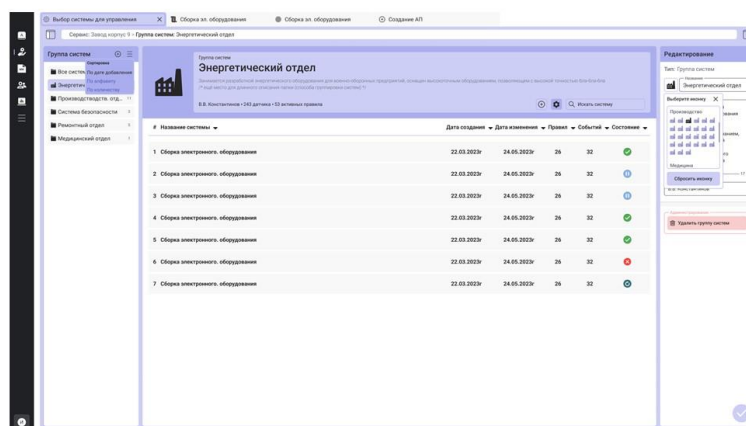
Программные средства управления



17/43

Рисунок Г.17 – Слайд презентации 17

Портал управления метаданными



18/43

Рисунок Г.18 – Слайд презентации 18

Применение No-Code в портале управления

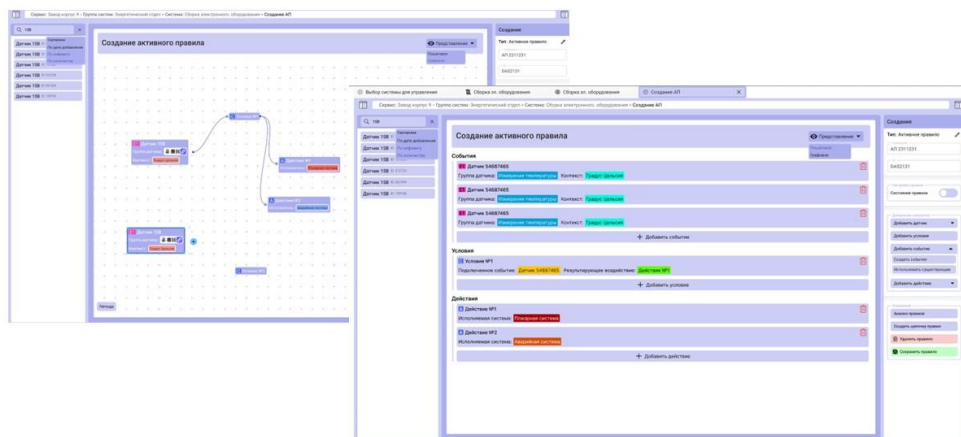
- 1 Работа в рамках единой метамодели
- 2 Применение «программ-мастеров» и технологии «Drag-and-drop» при создании/редактировании метаданных
- 3 Интеграция табличного, графового и языкового представлений метаданных
- 4 Автоматическая генерация языкового представления метаданных
- 5 Использование готовых шаблонов



19/43

Рисунок Г.19 – Слайд презентации 19

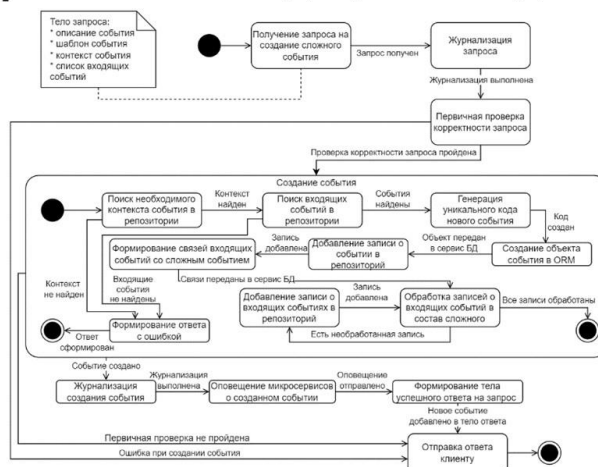
Конструирование метаданных



20/43

Рисунок Г.20 – Слайд презентации 20

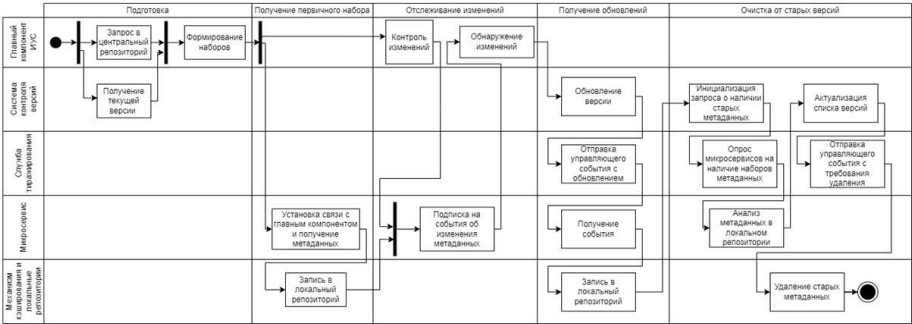
Первичная валидация метаданных



21/43

Рисунок Г.21 – Слайд презентации 21

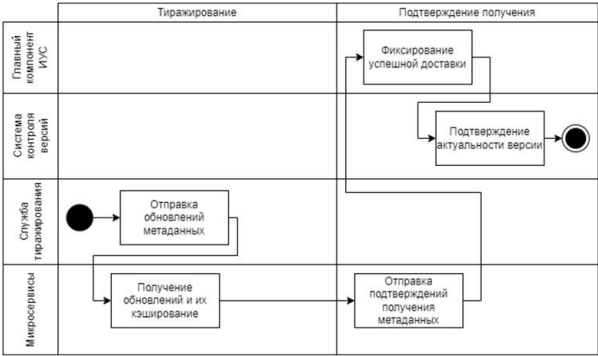
Тиражирование метаданных



22/43

Рисунок Г.22 – Слайд презентации 22

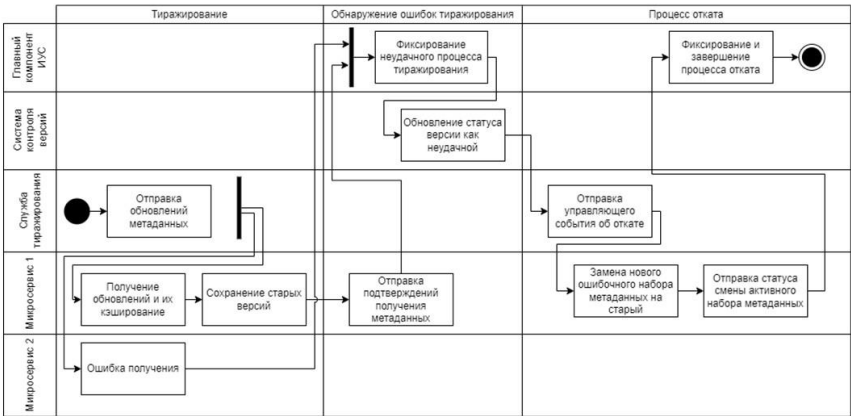
Процесс синхронизации при успешном тиражировании



23/43

Рисунок Г.23 – Слайд презентации 23

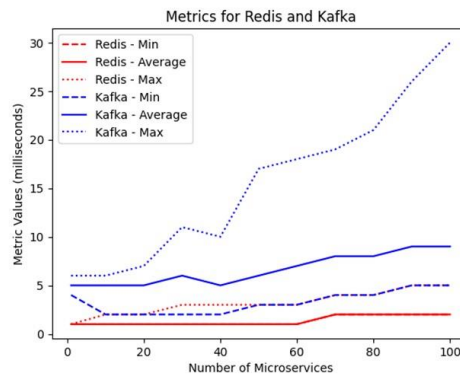
Процесс отката до предыдущего состояния метаданных



24/43

Рисунок Г.24 – Слайд презентации 24

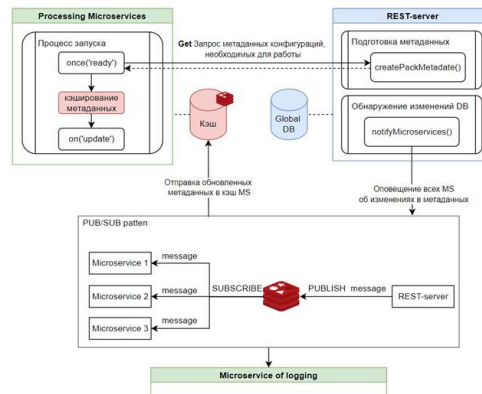
Выбор средств тиражирования и синхронизации



25/43

Рисунок Г.25 – Слайд презентации 25

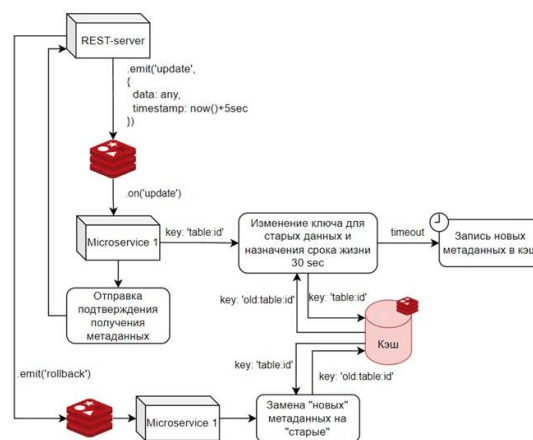
Архитектура программных средств в процессе тиражирования и синхронизации



26/43

Рисунок Г.26 – Слайд презентации 26

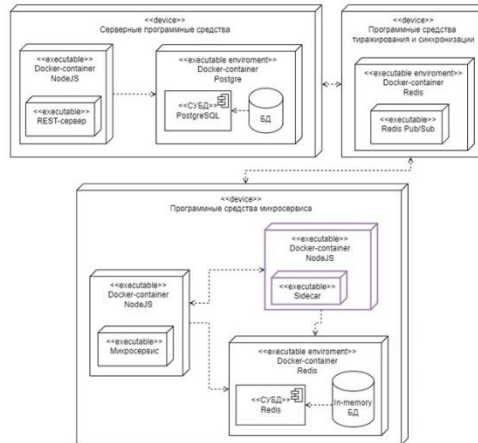
Rollback-процесс



27/43

Рисунок Г.27 – Слайд презентации 27

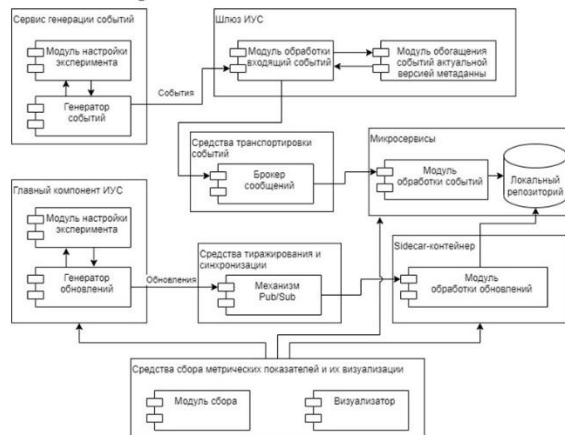
Использование Sidecar-контейнера



28/43

Рисунок Г.28 – Слайд презентации 28

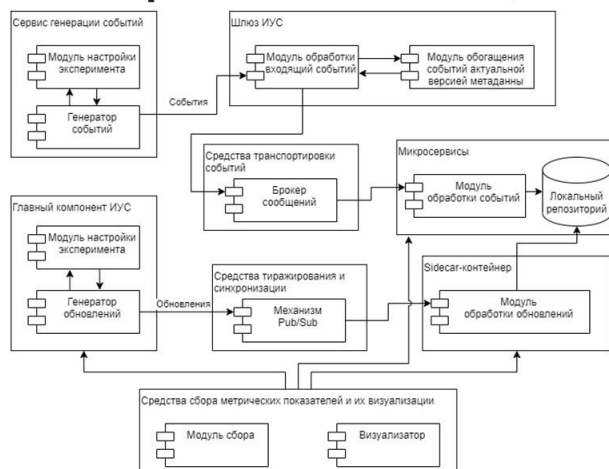
Экспериментальная оценка



29/43

Рисунок Г.29 – Слайд презентации 29

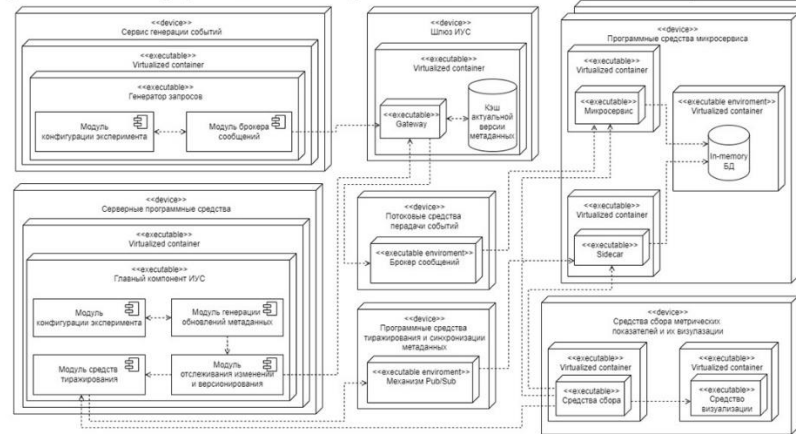
Экспериментальная оценка



30/43

Рисунок Г.30 – Слайд презентации 30

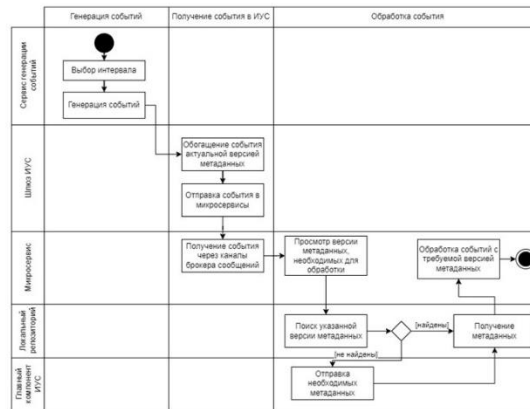
Архитектура для проведения эксперимента



31/43

Рисунок Г.31 – Слайд презентации 31

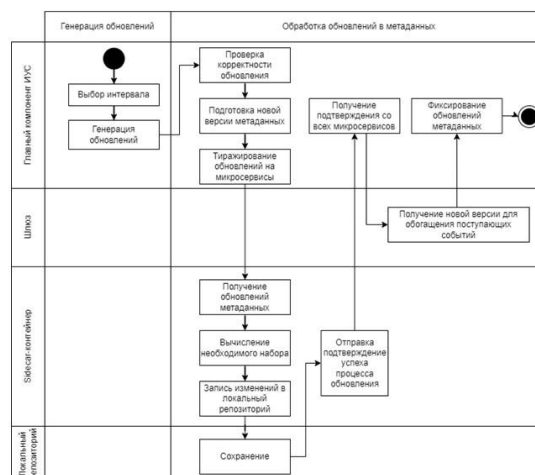
Первый сценарий эксперимента



32/43

Рисунок Г.32 – Слайд презентации 32

Второй сценарий эксперимента



33/43

Рисунок Г.33 – Слайд презентации 33

Стандартные показатели

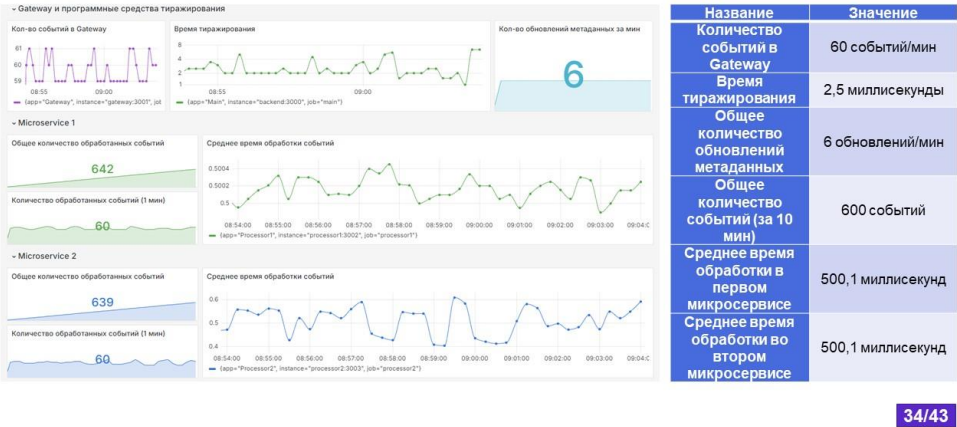


Рисунок Г.34 – Слайд презентации 34

Показатели при увеличении числа генерируемых событий

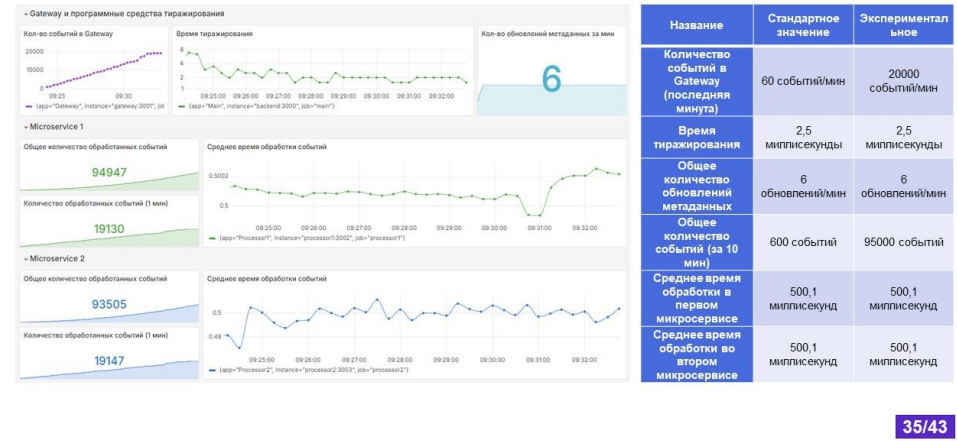


Рисунок Г.35 – Слайд презентации 35

Показатели при увеличении числа обновлений метаданных

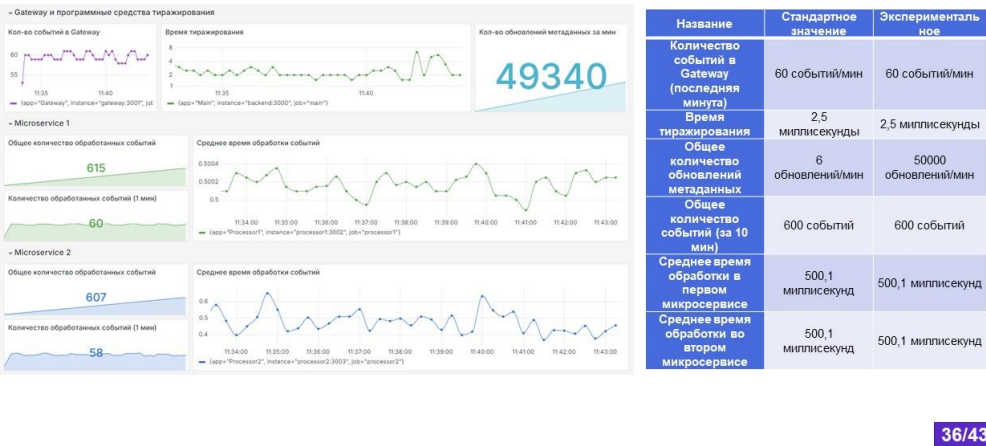
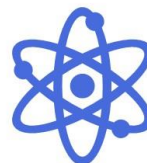


Рисунок Г.36 – Слайд презентации 36

Основные результаты работы

- 1 Исследованы и разработаны модели, алгоритмы и программные средства управления метаданными в дискретно-событийных информационно управляющих системах, построенных на базе микросервисной программной платформы потоковой обработки и исполнения активных правил
- 2 Разработана единая метамодель, которая включает в себя метамодель внешних систем, событий и активных правил
- 3 Разработаны алгоритмы анализа, тиражирования и синхронизации метаданных в компонентах микросервисной архитектуры
- 4 Проведена экспериментальная оценка разработанных моделей, алгоритмов и программных средств управления метаданными. В результате можно сделать вывод, что разработанные средства показали высокую эффективность



37/43

Рисунок Г.37 – Слайд презентации 37

Список опубликованных научных трудов

- 1 «Метамодель сервиса потоковой обработки событий»
Методы, средства и технологии получения и обработки измерительной информации («Шляндские чтения - 2022») : материалы XIV Международной научно-технической конференции с элементами научной школы и конкурсом научно-исследовательских работ для обучающихся и молодых ученых, Пенза, 24–26 октября 2022 года. – Пенза: Пензенский государственный университет, 2022. – С. 240-243.
- 2 «Использование No-Code подхода при управлении метаданными сервиса потоковой обработки событий и исполнения активных правил»
Информационные технологии в науке и образовании. Проблемы и перспективы : сборник статей по материалам X Всероссийской научно-практической конференции, Пенза, 15 марта 2023 года. – Пенза: Пензенский государственный университет, 2023. – С. 212-215
- 3 «Микросервис исполнения активных правил»
Инженеринг и технологии. – 2023. – Т. 8, № 1. – С. 23-27.
- 4 «Портал управления метаданными сервиса потоковой обработки событий и исполнения активных правил»
Сборник трудов Всероссийской научной конференции с международным участием «Информационные технологии в моделировании и управлении: подходы, методы, решения», Тольятти, 18–20 апреля 2023 года.
- 5 «Алгоритм выявления и устранения коллизии заикливания активных правил с применением сетей Петри»
Сборник материалов IX Международной научно-практической конференции (школы-семинара) молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук», Тольятти, 18–20 апреля 2023 года.

38/43

Рисунок Г.38 – Слайд презентации 38

Список опубликованных научных трудов

- 6 «Архитектура метаданных сервиса потоковой обработки событий и исполнения активных правил»
Научное издание «Вестник Пензенского государственного университета». – 2023. – № 3(43).
- 7 «Применение NoCode-подхода к управлению метаданными сервиса потоковой обработки событий и исполнения активных правил»
Наука, инновации и технологии: от идеи к внедрению : Материалы Всероссийской научно-практической конференции молодых ученых, Комсомольск-на-Амуре, 16–17 ноября 2023 года. – Комсомольск-на-Амуре: Комсомольский-на-Амуре государственный университет, 2023. – С. 445-447.
- 8 «Тиражирование и синхронизация метаданных в микросервисных приложениях»
Новые информационные технологии и системы (НИТИС-2023). Сборник научных статей по материалам XX Международной научно-технической конференции, посвященной 80-летию Пензенского государственного университета, Пенза, 16–17 ноября 2023 года. – Пенза: ПГУ, 2023. – С. 226-231.
- 9 «Программные средства выработки управляющих воздействий на основе активных правил в информационно-управляющих системах микросервисной архитектуры»
Проблемы автоматизации и управления в технических системах : Сборник научных статей XXXV Всероссийской с международным участием научно-технической конференции, посвященной 80-летию Пензенского государственного университета. в 2 т., Пенза, 17–19 октября 2023 года. – Пенза: Пензенский государственный университет, 2023. – С. 314-320.

39/43

Рисунок Г.39 – Слайд презентации 39

Апробация работы

- 1 XXV Международный конкурс научно-исследовательских работ «Путь в науку – 2022»
Диплом победителя II степени
- 2 Открытая XXXII университетская научно-практическая конференция «Актуальные проблемы науки и образования» (АПНО-2022, 11-15 апреля 2022 г.)
Диплом победителя I степени
- 3 Всероссийская научная конференция с международным участием «Информационные технологии в моделировании и управлении: подходы, методы, решения»
- 4 Всероссийская научно-практическая конференция молодых ученых «Наука, инновации и технологии: от идей к внедрению»
- 5 XXXV Всероссийская с международным участием научно-техническая конференция «Проблемы автоматизации и управления в технических системах»

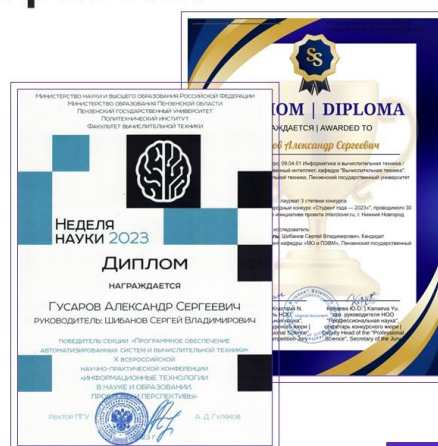


40/43

Рисунок Г.40 – Слайд презентации 40

Апробация работы

- 6 XIV Международная научно-техническая конференция с элементами научной школы и конкурсом научно-исследовательских работ «Шляндские чтения – 2022»
Диплом победителя I степени
- 7 X Ежегодная всероссийская межвузовская научно-практическая конференция «Информационные технологии в науке и образовании. Проблемы и перспективы.»
Диплом победителя I степени
- 8 Международная научно-практическая конференция молодых ученых «Прикладная математика и информатика: современные исследования в области естественных и технических наук»
- 9 XX Международная научно-техническая конференция «Новые информационные технологии и системы»

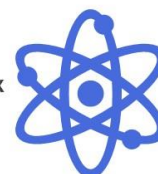


41/43

Рисунок Г.41 – Слайд презентации 41

Направления дальнейших исследований и разработок

- 1 Спецификация и разработка средств подключения внешних систем управления и наблюдения с целью сбора первичных метаданных
- 2 Спецификация и разработка эффективных алгоритмов и программных средств взаимодействия элементов микросервисной архитектуры в контексте обработки событий
- 3 Алгоритмы и программная реализация статического и динамического анализов для выявления аномалий в объектах метаданных
- 4 Улучшение средств журнализации для расширенного анализа метаданных



42/43

Рисунок Г.42 – Слайд презентации 42



**Спасибо за
ВНИМАНИЕ**

Рисунок Г.43 – Слайд презентации 43