

**Министерство образования и науки Российской Федерации**  
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  
**“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**  
**УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,**  
**МЕХАНИКИ И ОПТИКИ”**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**АВТОМАТИЗАЦИЯ КОНТРОЛЯ СОСТОЯНИЯ ПРОЦЕССОВ И**  
**МОНИТОРИНГ ИХ СЕТЕВОЙ АКТИВНОСТИ В СОСТАВЕ**  
**РАСПРЕДЕЛЕННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ**

Автор Крихели Артём Мерабович \_\_\_\_\_  
(Фамилия, Имя, Отчество) (Подпись)

Направление подготовки (специальность) \_\_\_\_\_

09.04.04 – Программная инженерия  
(код, наименование)

Квалификация магистр  
(бакалавр, магистр)

Руководитель Радченко И. А., к. т. н. \_\_\_\_\_  
(Фамилия, И., О., ученое звание, степень) (Подпись)

**К защите допустить**

Зав. кафедрой Муромцев Д. И., к. т. н., доцент \_\_\_\_\_  
(Фамилия, И., О., ученое звание, степень) (Подпись)

“ ” 20 \_\_\_\_ г.

Санкт-Петербург, 20 \_\_\_\_ г.

Студент Крихели А. М. Группа Р4217 Кафедра ИПМ Факультет ПИ и КТ  
(Фамилия, И, О.)

Направленность (профиль), специализация \_\_\_\_\_

09.04.04 – Разработка программно-информационных систем

Консультант (ы):

а) Чистяков Александр Анатольевич \_\_\_\_\_ (Подпись)  
(Фамилия, И., О., ученое звание, степень)

б) \_\_\_\_\_ (Подпись)  
(Фамилия, И., О., ученое звание, степень)

ВКР принята “ \_\_\_\_\_ ” \_\_\_\_\_ 20 \_\_\_\_\_ г.

Оригинальность ВКР \_\_\_\_\_ %

ВКР выполнена с оценкой \_\_\_\_\_

Дата защиты “ \_\_\_\_\_ ” \_\_\_\_\_ 20 \_\_\_\_\_ г.

Секретарь ГЭК \_\_\_\_\_ (Подпись)  
(Фамилия, И., О.)

Листов хранения \_\_\_\_\_

Демонстрационных материалов / чертежей хранения \_\_\_\_\_

**Министерство образования и науки Российской Федерации**  
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  
**“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**  
**УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,**  
**МЕХАНИКИ И ОПТИКИ”**

**УТВЕРЖДАЮ**

Зав. кафедрой \_\_\_\_\_

(ФИО)

(подпись)

« \_\_\_\_\_ » « \_\_\_\_\_ » 20 \_\_\_\_ г.

**ЗАДАНИЕ**  
**НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ**

Студенту Крихели А.М. Группа Р4217 Кафедра ИПМ Факультет ПИиКТ **Руководитель**  
Радченко И.А., к.т.н., доцент

(ФИО, ученое звание, степень, место работы, должность)

**1 Наименование темы:** Автоматизация контроля состояния процессов и мониторинг их сетевой активности в составе распределенной информационной системы

**Направление подготовки (специальность)** 09.04.04 Программная инженерия

**Направленность (профиль)** Разработка программно-информационных систем

**Квалификация** магистр

**2 Срок сдачи студентом законченной работы** « \_\_\_\_\_ » « \_\_\_\_\_ » 20 \_\_\_\_ г.

**3 Техническое задание и исходные данные к работе**

Необходимо выполнить анализ различных подходов, применяемых при проектировании распределенных информационных систем уровня предприятия. Сформировать перечень требований к программному решению для управления такими системами; проанализировать существующие программные продукты, выявить их преимущества и недостатки. Разработать собственный независимый от платформы инструмент, соответствующий сформулированному ранее перечню требований.

**4 Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов)**

4.1 Анализ предметной области.

4.2 Анализ программных решений для управления распределенной информационной системой.

4.3 Анализ и выбор средств разработки.

4.4 Проектирование программного инструмента.

4.5 Тестирование разработанного приложения.

## **5 Перечень графического материала (с указанием обязательного материала)**

---

---

---

---

---

---

---

## **6 Исходные материалы и пособия**

---

---

---

---

---

---

---

**7 Дата выдачи задания** « \_\_\_\_ » « \_\_\_\_\_ » 20 \_\_\_\_ г.

Руководитель ВКР \_\_\_\_\_  
(подпись)

Задание принял к исполнению \_\_\_\_\_ « \_\_\_\_ » « \_\_\_\_\_ » 20 \_\_\_\_ г.  
(подпись)

**Министерство образования и науки Российской Федерации**  
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  
**“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**  
**УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,**  
**МЕХАНИКИ И ОПТИКИ”**

## **АННОТАЦИЯ**

### **ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ**

Студент Крихели Артём Мерабович

(ФИО)

**Наименование темы ВКР:** Автоматизация контроля состояния процессов и мониторинг их сетевой активности в составе распределенной информационной системы

**Наименование организации, где выполнена ВКР:** Университет ИТМО

### **ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ**

1 Цель исследования: решение проблемы автоматизации контроля и мониторинга состояния процессов в корпоративных распределенных информационных системах.

2 Задачи, решаемые в ВКР: анализ различных подходов, применяемых при проектировании распределенных информационных систем уровня предприятия. Формирование перечня требований к программному решению для управления такими системами; анализ существующих программных продуктов. Разработка собственного независимого от платформы инструмента, соответствующего сформулированному ранее перечню требований.

3 Число источников, использованных при составлении обзора: 14

4 Полное число источников, использованных в работе: 23

5 В том числе источников по годам

Отечественных			Иностранных		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет
13	2	1	7	0	0

6 Использование информационных ресурсов Internet: Да, 9

(Да, нет, число ссылок в списке литературы)

7 Использование современных пакетов компьютерных программ и технологий (Указать, какие именно, и в каком разделе работы)

Пакеты компьютерных программ и технологий	Раздел работы
Java SE 8, IDE NetBeans, Maven, Google Guice, Jetty, Jersey	3-7
ReactJS	4
Hazelcast	5
JUnit 4	6

8 Краткая характеристика полученных результатов: В рамках выполнения данной работы был разработан кроссплатформенный программный инструмент для автоматизации контроля и мониторинга состояния процессов в распределенных корпоративных системах.

9 Полученные гранты, при выполнении работы: Нет  
(Название гранта)

10 Наличие публикаций и выступлений на конференциях по теме выпускной работы: Да  
(Да, нет)

- а) 1 Крихели А.М., Цопа Е.А. Автоматизация контроля состояния процессов и мониторинг их сетевой активности // Альманах научных работ молодых ученых Университета ИТМО - 2016. - Т. 3. - С. 14-16.  
2 Крихели А.М., Радченко И.А. Автоматизация контроля и мониторинг состояния процессов в составе распределенных корпоративных информационных систем // Сборник тезисов докладов конгресса молодых ученых. Электронное издание [Электронный ресурс]. - Режим доступа:  
[http://kmu.ifmo.ru/collections\\_article/7689/avtomatizaciya\\_kontrolya\\_i\\_monitoring\\_sostoyaniya\\_processov\\_v\\_sostave\\_raspredeleennyh\\_korporativnyh\\_informacionnyh\\_sistem.htm](http://kmu.ifmo.ru/collections_article/7689/avtomatizaciya_kontrolya_i_monitoring_sostoyaniya_processov_v_sostave_raspredeleennyh_korporativnyh_informacionnyh_sistem.htm), свободный.
- б) 1 Выступление на XLVI научной и учебно-методической конференции Университета ИТМО, 2017 год.  
2 Доклад на тему «Автоматизация контроля и мониторинг состояния процессов в составе распределенных корпоративных информационных систем» на VII Конгрессе молодых ученых, 2018 год.

Студент: Крихели Артём Мерабович  
(ФИО) (подпись)

Руководитель: Радченко Ирина Алексеевна  
(ФИО) (подпись)

“ \_\_\_\_\_ ” \_\_\_\_\_ 20\_\_ г.

## Оглавление

ВВЕДЕНИЕ.....	6
Глава 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ .....	8
1.1 Распределенные информационные системы.....	8
1.1 Микросервисная архитектура .....	10
Глава 2. АНАЛИЗ ПРОГРАММНЫХ РЕШЕНИЙ ДЛЯ УПРАВЛЕНИЯ РАСПРЕДЕЛЕННОЙ ИНФОРМАЦИОННОЙ СИСТЕМОЙ .....	14
2.1 Формирование и анализ требований.....	14
2.2 Анализ существующих программных инструментов .....	17
2.3 Результаты анализа .....	21
Глава 3. АНАЛИЗ И ВЫБОР СРЕДСТВ РАЗРАБОТКИ.....	23
Глава 4. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ИНСТРУМЕНТА.....	31
4.1 Архитектура приложения.....	31
4.2 Машина состояний процесса .....	35
4.3 Машина состояний экземпляра менеджера.....	38
4.4 Проектирование программного интерфейса для мониторинга микросервисов .....	41
Глава 5. СИНХРОНИЗАЦИЯ ДАННЫХ В РАСПРЕДЕЛЕННОМ ХРАНИЛИЩЕ .....	43
5.1 Структурирование данных в Hazelcast.....	43
5.2 Алгоритм синхронизации данных.....	47
5.3 Завершение работы экземпляра системного менеджера .....	49
Глава 6. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ .....	52
Глава 7. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	54
7.1 Конфигурационный файл .....	54
7.2 Пользовательский веб-интерфейс .....	56

ЗАКЛЮЧЕНИЕ .....	61
СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ.....	62
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....	63



## **ВВЕДЕНИЕ**

На сегодняшний день информационные системы являются неотъемлемой частью инфраструктуры множества промышленных предприятий, научных комплексов, государственных учреждений, учебных заведений и даже небольших индивидуальных организаций. Электронный документооборот и его автоматизация становятся потребностью, и необходимы в различных областях социального взаимодействия. Индустрия программного обеспечения динамично прогрессирует и развивается, не оставляя без влияния сегмент мировой экономики.

Профессиональное проектирование программного продукта, в первую очередь, подразумевает глубокий анализ предметной области для корректного последующего выбора технических решений и подходов. От поставленных функциональных и нефункциональных требований напрямую зависят не только архитектура системы, но и методология разработки. Необходимо учитывать множество косвенных и прямых факторов для обеспечения процессов конструирования информационной системы, ее тестирования и последующего внедрения.

В настоящее время наблюдается тенденция перехода от крупных монолитных централизованных систем к совокупности распределенных микросервисов [1], образующих единую корпоративную информационную систему. В качестве основных преимуществ использования микросервисной архитектуры можно выделить модульность, независимость выбора технологических решений и масштабируемость. Делегирование зон ответственности работы системы различным компонентам гарантирует гибкость разработки и упрощает расширяемость программного продукта. Поддержка и расширение набора функциональных возможностей информационных систем, в основе которых лежит монолитное ядро, по мере разрастания становится гораздо сложнее.

Основная сложность при работе с системами, использующими микросервисную архитектуру, заключается в более сложном проектировании таких систем и их развертывании. Любой из сервисов может непредвиденно завершить свою работу, и, как следствие, бизнес-логика системы, реализуемая данным сервисом, окажется недоступной. В связи с этим появляется необходимость технического мониторинга приложений [2], автоматизации их работы, сбора различных метрик, логирования и обеспечения отказоустойчивости. С помощью семантического мониторинга можно получить актуальную логическую информацию, касающуюся конкретных бизнес-процессов, а также статистические данные о работе всей информационной системы.

Целью данной работы является исследование различных методов управления распределенными корпоративными информационными системами, модернизация существующих методик и повышение их эффективности в рамках заданных критериев. В ходе анализа предметной области была выявлена необходимость внедрения специализированных программных решений, позволяющих автоматизировать контроль состояния процессов внутри распределенных систем, а также составлен перечень требований к таким программным инструментам. Были выявлены преимущества и недостатки готовых приложений, удовлетворяемых сформированному списку требований в полном объеме или частично.

В процессе выполнения исследования была выявлена необходимость реализации собственных алгоритмов и универсальной машины состояний процессов для последующего внедрения в новый программный продукт. Помимо разработки собственного кроссплатформенного приложения, был реализован определенный набор интерфейсов, гарантирующий модульность, модернизацию и расширяемость программного инструмента. Разработанное решение имеет открытый исходный код, что, несомненно, положительно сказывается на практической и теоретической значимости работы.

## **Глава 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

### **1.1 Распределенные информационные системы**

В различных научных литературных источниках можно найти множество определений для термина «распределенная информационная система». В данной работе за основу будет взята формулировка, которую предоставляет Эндрю Таненбаум. Он определяет распределенную систему в качестве набора независимых друг от друга вычислительных машин, представляющихся пользователю единой системой [3]. К характеристикам любой распределенной информационной системы можно отнести расширяемость и масштабирование. Для конечного пользователя децентрализация системы всегда остается скрытой; пользователь взаимодействует с системой как с обычным классическим приложением.

На сегодняшний день клиент-серверная архитектура [4] лежит в основе большинства распределенных систем. Множество физических вычислительных машин объединяется в специализированные группы, что способствует повышению производительности. В качестве классического примера корпоративной информационной распределенной системы можно рассмотреть банковскую сеть. Филиалы такой сети могут находиться в разных географических средах, но это не должно препятствовать их взаимодействию друг с другом. Каждый из филиалов банка должен иметь прикладной доступ к хранилищу данных сети для ведения учета, регистрации новых клиентов, совершению межбанковских транзакций и других операций. Упрощенная классическая схема взаимодействия с такой системой выглядит следующим образом: сотрудник банка через специальную программу-клиент выполняет запрос на один из банковских серверов, например, о совершении операции банковского перевода. Задачей сервера является в первую очередь проверка корректности данных запрашиваемой операции. Необходимо убедиться в том, что сумма, на которую

запрошен перевод, доступна на счете клиента. Соответственно, информационная система должна гарантировать согласованность данных и их доступность в любой момент времени, вне зависимости от того, к какому физическому серверу был сделан запрос.

Если рассматривается централизованное хранилище, то основную ответственность за согласованность несет база данных [5]. Однако в некоторых случаях объемы данных могут быть очень велики, что влечет за собой необходимость децентрализации и внедрения распределенных транзакций. Сбой одного или нескольких серверов не должен вызывать критических нарушений в работе системы. Более того, система должна оставаться в согласованном состоянии, а все запросы, которые не были обработаны из-за отказа вычислительных машин, попасть в очередь на обработку доступным серверам. Несмотря на то, что при аварийном завершении работы нескольких серверов общая производительность всей банковской сети будет снижена, система все равно продолжит функционирование. Данное свойство носит название «отказоустойчивость» и оно является характерным для распределенных информационных систем.

Группа вычислительных машин, взаимодействующих друг с другом, которая рассматривается, как единая система, называется кластером. Физический компьютер внутри кластера принято называть узлом. Использование кластерных решений позволяет абстрагироваться от физической топологии размещения вычислительных машин. В рассматриваемом примере с банковской сетью сервера образуют кластеры, что обеспечивает прозрачность местоположения и прозрачность отказов. В случае высокой нагрузки, распределенные системы могут значительно повысить производительность путем горизонтального масштабирования.

Таким образом, распределенные информационные системы предоставляют ряд преимуществ, необходимых для стабильной работы корпоративных приложений.

### 1.1 Микросервисная архитектура

Микросервисная архитектура получила широкое распространение и используется в большом количестве промышленных информационных систем. Основа концепции проектирования микросервисов заключается в разделении бизнес-процессов не только на логические уровни одного приложения, но и на физически разделенные компоненты (сервисы) [6]. Для выявления преимуществ и недостатков использования микросервисов рассмотрим архитектуру классического монолитного приложения, представленную на рисунке 1.



Рисунок 1 – Монолитная архитектура

К основным недостаткам подхода монолитной архитектуры можно отнести следующие:

1. В процессе разработки возрастает сложность системы, что может негативно сказаться на ее поддержке и эксплуатации в дальнейшем. Обучение новых разработчиков занимает большее количество времени и ресурсов.
2. Проблематичное написание модульных тестов и автоматизированного тестирования в целом. Возникает необходимость ручного тестирования через пользовательский интерфейс.
3. Привязка к конкретным технологиям и инструментам, которые могут потерять поддержку в ходе эксплуатации и дальнейшего развития системы.
4. Возрастание сложности разделения разработки.

Учитывая перечень недостатков монолитного дизайна, можно сделать вывод, что данный вид архитектуры хорошо применим для небольших прикладных пользовательских приложений, но непригоден для использования в крупных корпоративных информационных системах.

Исходная концепция микросервисов задает перечень требований и характеристик для данного архитектурного шаблона. В рамках одного микросервиса всегда решается ограниченный набор задач (очень часто – единственная задача), который может быть спроецирован на конкретный бизнес-процесс предприятия. Микросервисы взаимодействуют друг с другом с помощью предоставляемых интерфейсов, сохраняя при этом слабую связанность, что подразумевает их повторное использование. Данный подход упрощает процедуру тестирования в виду небольших размеров микросервисов. Разработка становится более продуктивной, а отладка более простой и явной за счет независимого развертывания микросервисов друг от друга. Схема такой распределенной системы представлена на рисунке 2.

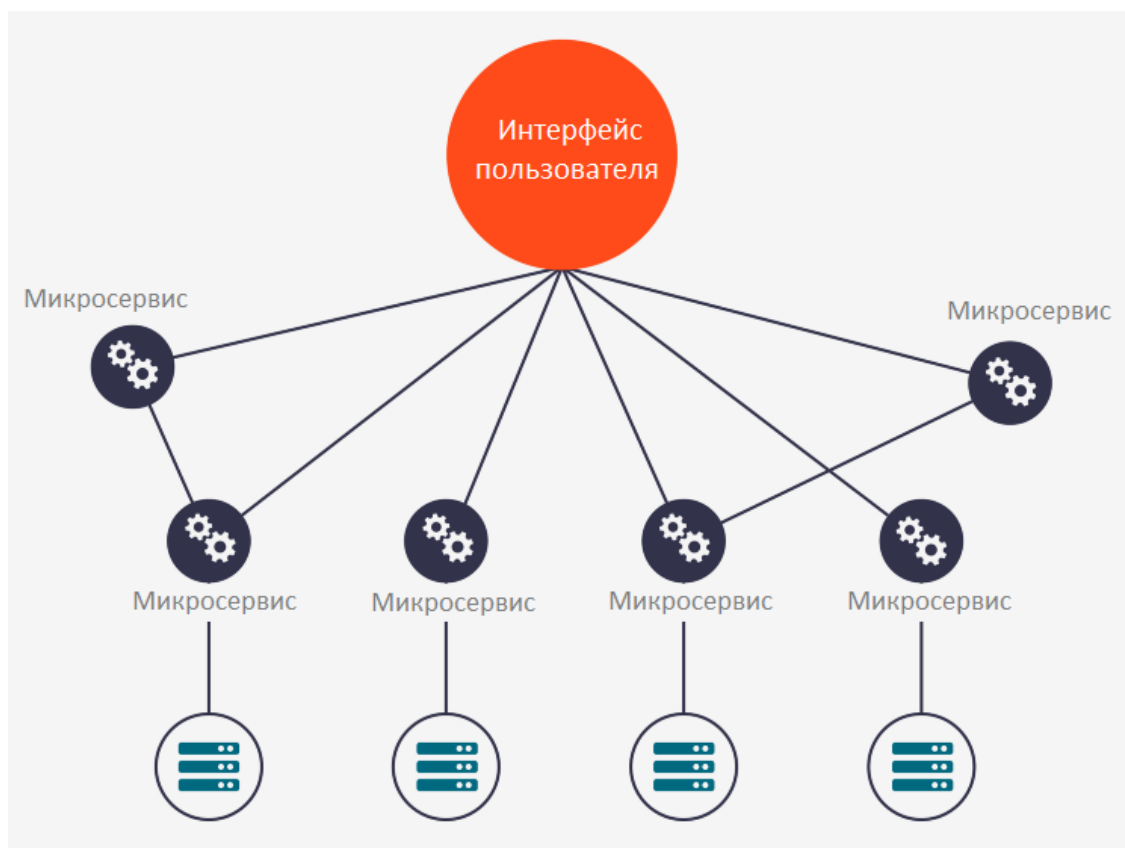


Рисунок 2 – Микросервисная архитектура

В виду разделения логики информационной системы на различные слабо связанные компоненты, инкапсулирующие в себе логику бизнес-процессов предприятия, можно говорить об отказоустойчивости такой системы. В случае непредвиденной аварийной остановки вычислительной машины, на которой запущена часть микросервисов, информационная система продолжит свою работу, утратив лишь определенный набор функциональных возможностей. Если же в системе предусмотрена репликация микросервисов, то она продолжит свою работу в штатном режиме.

Немаловажной особенностью микросервисной архитектуры является гетерогенность. При проектировании таких систем нет жесткой привязки к конкретным техническим инструментам для разработки и языкам программирования. Таким образом, в случае поддержки кроссплатформенности

всеми компонентами, отдельные группы сервисов могут работать под управлением разных операционных систем.

Основными недостатками данной архитектуры программного обеспечения являются:

1. Обмен данными между микросервисами происходит посредством сети, которая не является надежным каналом связи. Это всегда необходимо учитывать при проектировании и разработке системы. В зарубежной литературе данный подход имеет название «design for failure».
2. Несмотря на слабую связанность, изменения интерфейсов взаимодействия микросервисов могут повлечь за собой необходимость глобальных доработок во всей системе. Проектирование программных интерфейсов должно быть продумано заранее с учетом множества факторов.
3. Необходим непрерывный мониторинг поведения системы в целом и отдельных ее частей; должна быть продумана централизованная система логирования. Внедрение инструментов для получения различных метрик, характеризующих состояние микросервисов, является потребностью.
4. Необходим инструмент для автоматизированного развертывания системы, контролирующей ее состояние.

В некоторых случаях управление распределенными системами представляет собой нетривиальную задачу. Актуальность проблемы исследования методов универсальных решений администрирования таких систем обусловлена широким использованием описанных подходов в современном мире информационных технологий.



## **Глава 2. АНАЛИЗ ПРОГРАММНЫХ РЕШЕНИЙ ДЛЯ УПРАВЛЕНИЯ РАСПРЕДЕЛЕННОЙ ИНФОРМАЦИОННОЙ СИСТЕМОЙ**

### **2.1 Формирование и анализ требований**

Детальный анализ предметной области показал, что существует необходимость в некотором программном решении, позволяющем упростить процесс мониторинга корпоративной информационной распределенной системы и автоматизировать контроль состояния микросервисов в ней. На основе результатов анализа можно выдвинуть список основных требований для такого программного инструмента.

В первую очередь, так как работа ведется с распределенными системами, необходимо обеспечить децентрализацию программного менеджера. Соответственно, он должен обладать такими характеристиками, как отказоустойчивость и консистентность данных. При обращении к любому из экземпляров менеджера пользователь должен иметь возможность получить состояние всей системы, а не конкретного физического узла, к которому привязан данный экземпляр. В случае критического сбоя одного или нескольких экземпляров системного менеджера, микросервисы, находящиеся под контролем этих экземпляров, должны продолжить свою работу.

Следующим важным требованием, предъявляемым к программному обеспечению, является модульность [7]. Каждая система уникальна и имеет ряд специфических особенностей, поэтому найти универсальный способ, обеспечивающий автоматизацию работы любой распределенной системы, не представляется возможным. Соответственно, помимо базовой реализации функциональных возможностей, программное решение должно предоставлять ряд интерфейсов для разработчиков и возможность задания детальной конфигурации. Необходимо учесть, что в рамках одной системы группы микросервисов могут взаимодействовать с помощью различных сетевых протоколов, обмениваясь при

этом данными разных форматов. Для выполнения технического мониторинга сервисов с помощью программного менеджера нужно обеспечить поддержку подключения сторонних настраиваемых модулей, реализующих логику наблюдения за сервисами.

Немаловажным является обеспечение независимости от конкретной платформы, так как в ходе анализа предметной области было выявлено, что группы микросервисов могут быть запущены на разных физических компьютерах с различными операционными системами. Конфигурирование программного менеджера также должно быть прозрачным для администратора системы: привязки к конкретной платформе должны быть сокращены до минимума. Несмотря на то, что большинство серверов работают на базе операционных систем семейства UNIX, менеджер должен предоставлять максимальную переносимость с одной платформы на другую.

Журналирование всех событий, связанных с работой микросервисов, является обязательным требованием. Детализированные лог-файлы помогают быстрее обнаружить неполадки в работе программного обеспечения, выявить их характер и основные причины, а значит, соответственно, дают возможность их оперативного устранения, что минимизирует затраты ресурсов. Помимо журналирования, программный инструмент должен предоставлять наличие пользовательского интерфейса для визуализации данных о текущем состоянии всей системы.

Возможность базовой автоматизации развертывания всей системы или отдельных ее частей должна быть реализована в системном менеджере. Помимо запуска системы или групп микросервисов, необходимо иметь также механизм их корректной остановки. Кроме того, программный инструмент должен обеспечить контроль состояний каждого из сервисов в реальном времени и его автоматизацию. Поведение менеджера при сбое конкретного сервиса должно

описываться в конфигурационном файле для обеспечения гибкости системы и ее расширяемости.

Данное программное решение должно предоставлять ряд базовых возможностей для управления микросервисами и их группами на одном или нескольких физических узлах кластера. Основываясь на том, что каждый микросервис в контексте операционной системы является отдельно запущенным процессом, можно выделить ряд следующих операций, реализацию которых должен предоставлять менеджер администратору системы:

1. Запуск сервиса.
2. Остановка сервиса.
3. Запуск группы сервисов.
4. Остановка группы сервисов.
5. Загрузка и обновление конфигурации в реальном времени.
6. Выключение экземпляра менеджера.

Управление должно выполняться с помощью пользовательского и программных интерфейсов, предоставляемых системным менеджером. Вне зависимости от того, с каким конкретно экземпляром менеджера работает пользователь, управление группами всех сервисов должно быть доступно для гарантирования прозрачности. Некоторый набор функциональных возможностей по управлению микросервисами должен быть доступен и инструментам, выполняющим мониторинг внутри системного менеджера. Таким образом, при обнаружении пропажи сетевой активности у какого-либо сервиса может быть выполнена его автоматическая остановка и последующий запуск.

Открытый исходный код [8]. Необходимо учитывать, что множеств систем работают с секретной закрытой информацией, доступ к которой не является допустимым для третьих лиц. Внедрение в такую систему программного решения с уязвимостями, которые могут быть даже нарочно запрограммированы

разработчиками в целях хищения информации, недопустимо. С другой стороны, открытый исходный код помогает упростить процесс модернизации программного обеспечения и исправления существующих недоработок. Последующая возможная разработка собственных модулей, дополняющих базовые возможности системного менеджера, подчеркивает потребность в открытых исходных кодах.

## **2.2 Анализ существующих программных инструментов**

Широкое распространение распределенных систем в индустрии информационных технологий повлекло за собой необходимость разработки комплексных программных решений для управления такими системами. Микросервисная архитектура, как шаблон проектирования, была построена на базе философии UNIX [9]. Данный подход призывает к разработке гибких, небольших и расширяемых приложений, которые можно будет легко поддерживать и дополнять другим разработчикам, не являющимися его первоначальными создателями. Следовательно, необходимо обратить внимание на набор стандартных средств, входящих в состав экосистемы современных операционных UNIX-подобных систем.

На сегодняшний день широкое применение получила утилита под названием «systemd» [10] в семействе Linux-систем. Основное предназначение данного инструмента заключается в управлении системными сервисами и в их инициализации. С другой стороны, данное программное решение призвано унифицировать конфигурацию и поведение сервисов во всех дистрибутивах операционных систем Linux. В настоящее время подавляющее большинство дистрибутивов используют «systemd» в качестве системы инициализации сервисов по умолчанию. Рассмотрим более детально возможности данного программного продукта:

1. Запуск сервисов, включающий в себя учет и разрешение зависимостей. Если существует необходимость старта одного сервиса, которому требуется активное состояние другого сервиса, `systemd` автоматически разрешит все зависимости, гарантируя правильный порядок запуска.
2. Разрешение конфликтов при управлении сервисами. Например, если двум сервисам потребуются одни и те же ресурсы, то `systemd` не позволит активировать оба сервиса одновременно.
3. Отслеживание состояния сервисов и автоматический перезапуск в случае их непредвиденной остановки.
4. Управление сервисами по определенным событиям устройств или событиям времени.
5. Поддержка пользовательских скриптов и программ, выполняющихся при изменении состояния сервисов.

Несмотря на вышеизложенные возможности системной утилиты `systemd`, она не соответствует перечню требований, предъявленных к программному решению. В первую очередь, приложение `systemd` не предназначено для работы в распределенном окружении, так как его основная направленность связана с контролем сервисов в рамках одной операционной системы. Также данный программный продукт является зависимым от платформы и привязан к операционным системам, основанным на ядре Linux. Следовательно, можно сделать вывод, что использование приложения `systemd` является целесообразным для управления микросервисами в нераспределенных небольших информационных системах, которые нацелены на работу с конкретной платформой.

В качестве инструмента управления кластером широкое применение обретает программный продукт под названием «Apache Mesos» [11]. Данный инструмент отличается гибкостью и тонкостью настройки за счет подключения дополнительных фреймворков, что является неоспоримым преимуществом, так

как модульность - одно из основных и главных требований к программному решению. Соответственно, сам «Apache Mesos» представляет из себя некоторое ядро, предоставляющее среду выполнения. На данный момент разработано множество библиотек и фреймворков для взаимодействия с описываемым программным обеспечением. В какой-то степени можно даже проводить аналогию между описанным ранее systemd и Apache Mesos. Mesos является прикладным решением для управления задачами (процессами) внутри распределенной системы. Основные компоненты рассматриваемого программного продукта представлены на рисунке 3.

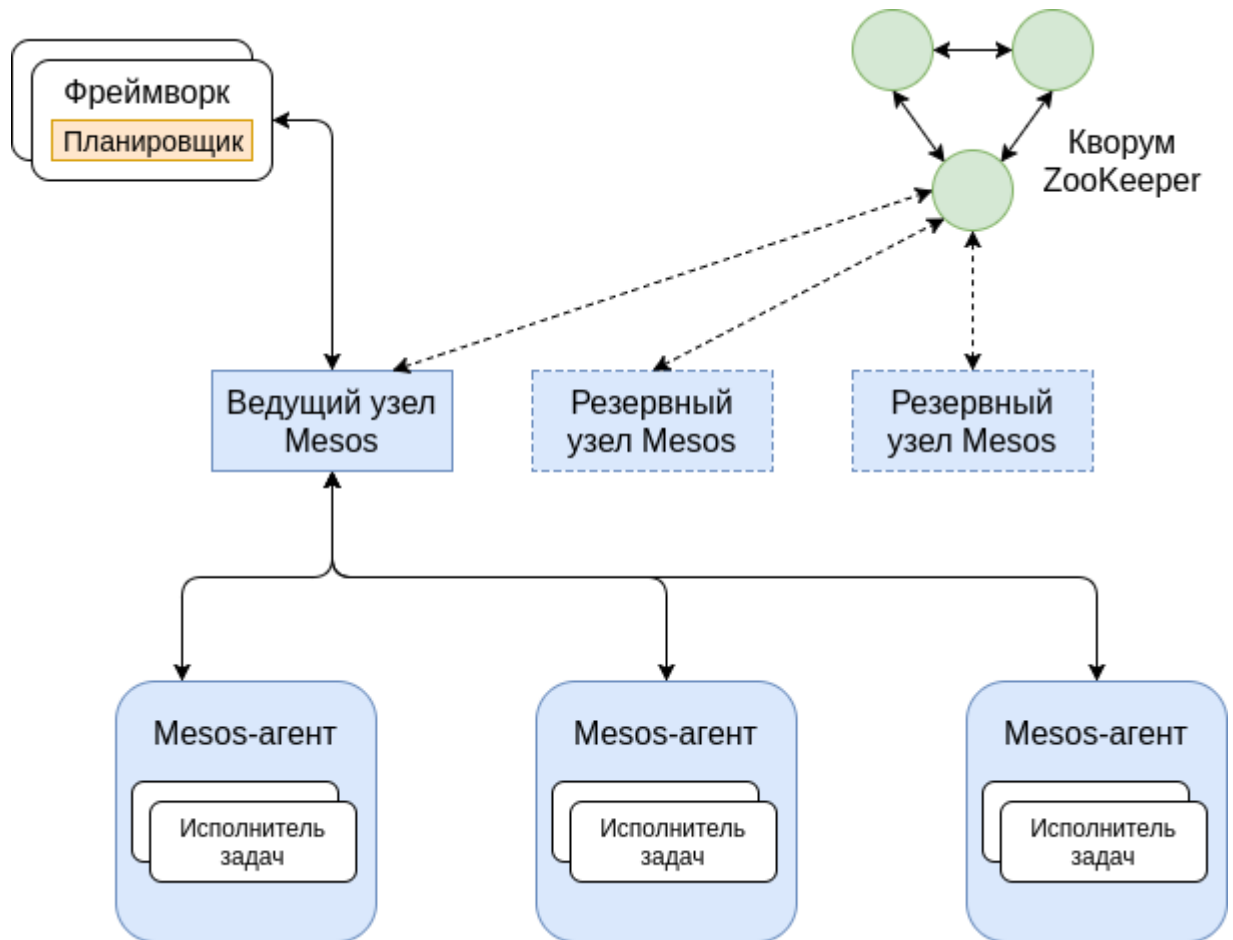


Рисунок 3 – Архитектура Apache Mesos

К основным особенностям Apache Mesos можно отнести поддержку управления контейнерами. Контейнеризация программного обеспечения

становится широко распространенной. Данное направление является перспективным, однако применение технологии контейнеров не представляется возможным для всех информационных систем. Помимо этого, Mesos является кластерным планировщиком.

Как видно из схемы, представленной на рисунке 3, Mesos является централизованным инструментом. Отказоустойчивость достигается за счет введения дополнительных резервных узлов mesos. Ведущий узел mesos является лидером, с которым осуществляют коммуникацию все mesos-агенты. Координация всех mesos-узлов выполняется с помощью программного инструмента под названием «Apache ZooKeeper» [12]. ZooKeeper также обеспечивает выбор лидера при условии наличия кворума.

Программное решение Apache Mesos помогает распределить задачи и эффективно использовать ресурсы внутри кластера. Множество крупных корпораций, например, таких, как Twitter и eBay, активно использует Apache Mesos в инфраструктуре своих информационных систем.

К основным недостаткам рассмотренного программного продукта можно отнести следующее:

1. Сложность эксплуатации. Apache Mesos требует установку и настройку дополнительных компонентов, необходимых для работы приложения. Одним из таких компонентов является ZooKeeper.
2. Для организации кворума необходимо минимум три физических узла. При этом для повышения доступности и отказоустойчивости разработчики рекомендуют использовать пять физических машин. Существуют распределенные системы, в которых использование большого количество узлов является нецелесообразным или избыточным. Соответственно, нет необходимости в затратах ресурсов для поддержки дополнительного оборудования.

3. Разработчики фреймворка «Marathon», который используется в Mesos в качестве исполнителя долгосрочных задач, предоставляют возможность проверки состояния запущенных сервисов посредством множества сетевых протоколов. Данная возможность в контексте Apache Mesos называется «Health Check». В качестве протокола очень часто используется HyperText Transfer Protocol (далее HTTP), а конфигурация задается в формате JavaScript Object Notation (далее JSON). Однако платформа для детального семантического мониторинга сервиса не предусмотрена.
4. Привязка к конкретной платформе. В официальной документации Apache Mesos в разделе «Known Limitations» указано, что ведущие узлы не совместимы с операционными системами семейства Windows.

### **2.3 Результаты анализа**

В ходе выполнения анализа существующих продуктов было выявлено, что на данный момент не существует легковесного кроссплатформенного решения, эксплуатация которого не требовала бы привлечения дополнительных специалистов, для управления распределенными системами и микросервисами в них. Основываясь на результатах детального анализа предметной области, можно сделать вывод, что за счет специфичности и нестандартности таких систем применение определенных готовых продуктов с открытым исходным кодом в качестве универсального единственного решения является крайне сложным. Существует практика разработки и внедрения дополнительных сервисов для управления конкретной информационной системой, однако такие решения являются строго специализированными и, как правило, производятся в рамках одной компании, следовательно, являются коммерческими продуктами.

Текущие результаты исследования показывают, что существует необходимость разработки собственного программного продукта, соответствующему перечню требований, описанному в части 2.1 данной работы.



Особое внимание должно быть уделено следующим свойствам, присущим распределенным системам:

1. **Согласованность данных.** В данном случае под данными подразумевается состояние групп микросервисов, работающих на разных физических узлах. Необходимо позаботиться об актуальности при визуализации данных.
2. **Согласованность операций.** Управление микросервисами, работающими на одном физическом узле, может производиться через другой узел кластера. Данным образом обеспечивается прозрачность управления для пользователя.
3. **Отказоустойчивость.** В случае отказа одного или нескольких физических узлов экземпляры менеджера, запущенные на других физических узлах должны продолжить свою работу при отсутствии иных конфигурационных правил поведения менеджера.

### Глава 3. АНАЛИЗ И ВЫБОР СРЕДСТВ РАЗРАБОТКИ

Решение о разработке программного обеспечения, предназначенного для контроля микросервисов в распределенных системах, было принято на основе полученных результатов анализа предметной области и анализа имеющихся инструментов с открытым исходным кодом. За основу решения прикладной задачи управления процессами были взяты результаты исследований, описанных в работе Крихели А.М. «Автоматизация контроля состояния процессов и мониторинг их сетевой активности». Данная работа предполагала последующую модернизацию системного менеджера для поддержки работы в распределенных системах, что было описано в главе 4 [13]. Помимо этого, все исходные коды, которые являлись приложением к исследовательской работе, находятся в открытом доступе. Соответственно, необходимо проанализировать средства разработки, которые использовались в вышеупомянутой работе, а также их переносимость и совместимость с текущими требованиями, сформулированными для нового программного решения.

Для решения проблемы с зависимостью от платформы и получения множества готовых реализаций набора возможностей для управления процессами в различных операционных системах было принято решение использования языка программирования Java [14]. Все исходные коды представлены именно на этом языке, в качестве инструмента для автоматизации сборки используется программное обеспечение «Apache Maven». Следует упомянуть, что в настоящее время язык Java широко используется для разработки непосредственно самих микросервисов и их групп, соответственно, для разработчика специфических модулей мониторинга упрощается процесс их интеграции и реализации: например, для получения какой-либо логической информации менеджером от микросервисов может использоваться технология Java Management Extensions. Помимо прочего, для языка Java существует множество библиотек,

предназначенных для ведения логов, наличие которых необходимо при работе с распределенными системами.

Согласно требованиям, необходима реализация визуализации состояний микросервисов в контексте кластера. Также существует потребность в удаленном управлении системой с помощью менеджера. Следовательно, необходима реализация серверной части. В качестве протокола передачи данных было принято решение использовать протокол прикладного уровня HTTP; архитектура взаимодействия с приложением построена на принципах «Representational state transfer» (далее – REST). В качестве реализации спецификации «Java API for RESTful Web Services» был выбран фреймворк под названием «Jersey». Библиотека, предоставляющая HTTP-сервер и контейнер сервлетов – «Jetty». Следует отметить, что при использовании данных библиотек нет необходимости в установке дополнительных серверов приложений, например, таких, как «Tomcat» или «Glassfish», что упрощает использование программного инструмента.

Для обеспечения гибкости, модульности и последующей модернизации продукта было принято решение о внедрении фреймворка, предоставляющего контейнер зависимостей. Исходя из ранее сформулированных требований, помимо реализации основных функциональных возможностей приложения, другим разработчикам должен предоставляться Application Programming Interface (далее – API), позволяющий реализовать новый или переопределить существующий набор функциональных возможностей. Использование контейнера зависимостей позволяет легко подменять реализацию существующих интерфейсов без повторной компиляции исходных кодов. Библиотека под названием «Google Guice» [15] является легковесным контейнером, обеспечивающим внедрение зависимостей, и легко интегрируется с описанными ранее используемыми инструментами.

Ведущим инструментом, необходимым для обеспечения поддержки сохранения и получения согласованного состояния микросервисов, запущенных

на разных физических узлах, является распределенное хранилище данных. Одним из главных требований к такому хранилищу является возможность его интеграции в конечное java-приложение. Таким образом, развертывание программного инструмента и его эксплуатация окажется достаточно простым (отсутствует необходимость в настройке и запуске дополнительных процессов, пропадает потребность в использовании баз данных). Широкой известностью обладает продукт под названием «Oracle Coherence», являющийся мощным инструментом для распределенных вычислений. Однако данное решение является платным, поэтому был рассмотрен его аналог с открытым исходным кодом «Hazelcast» [16]. В ходе выполнения анализа данного программного обеспечения были выявлены следующие особенности и достоинства:

1. Hazelcast является библиотекой с открытым исходным кодом, поставляющейся в качестве одного JAR-файла. Установка дополнительного программного обеспечения не требуется.
2. Отсутствует явное определение ведущего мастер-узла (лидера), соответственно, нет единой точки отказа.
3. Каждый физический узел обладает одинаковым набором функциональных возможностей.
4. Каждый физический узел может получить информацию обо всех участниках кластера.
5. Hazelcast предоставляет набор базовых распределенных структур данных (списки, словари, очереди и т.п.). Помимо этого, данная библиотека предоставляет распределенные блокировки и возможность проведения транзакций. Структуры данных могут добавляться и конфигурироваться динамически.
6. Hazelcast предоставляет распределенную реализацию шаблона проектирования «Издатель-подписчик» с помощью стандартных средств, входящих в состав библиотеки.

7. Распределение резервных копий данных по кластеру.
8. Поддержка динамического добавления новых участников (физических узлов) в кластер. Возможность использования различных механизмов обнаружения участников кластера.
9. Возможность получения метрик с помощью JMX.

Помимо этого, для упрощения этапов разработки и тестирования, Hazelcast предоставляет специальный центр управления («Hazelcast Management Center»), включающий в себя встроенный веб-интерфейс, для получения данных о работе кластера и его мониторинге. Также Hazelcast предоставляет возможность детальной настройки множества параметров посредством единственного конфигурационного файла.

Hazelcast может быть развернут в приложении двумя способами: встроенная интеграция и клиент-серверная. В рамках данной работы будет использоваться встроенная интеграция, подразумевающая то, что каждый участник кластера является экземпляром менеджера и имеет доступ к данным и сервисам распределенного хранилища (см. рисунок 4).

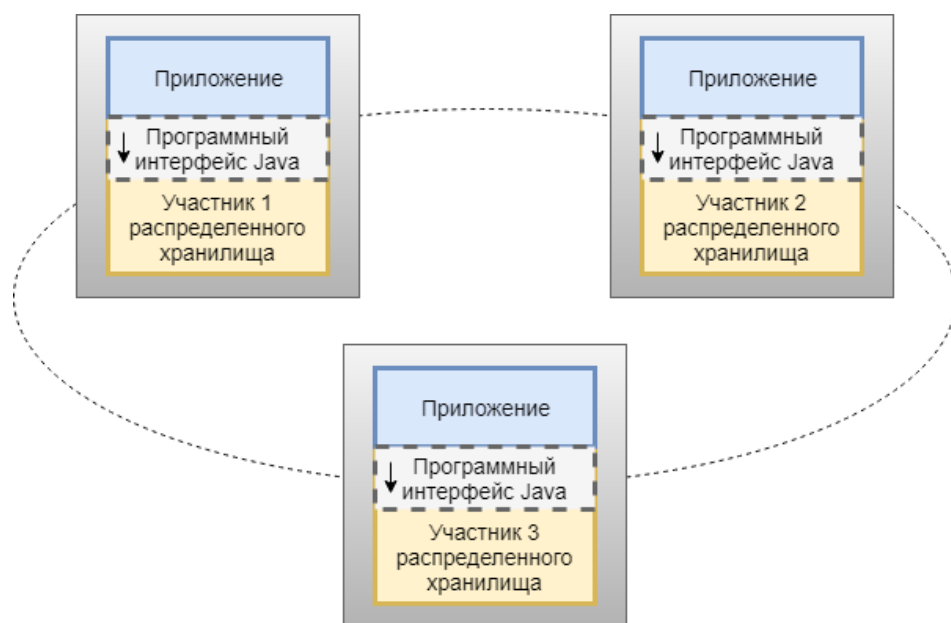


Рисунок 4 – Встроенная интеграция Hazelcast

Для хранения данных внутри кластера данная библиотека использует разделы (partitions). Каждый раздел является прикладным сегментом памяти, который может содержать в себе множество блоков данных. Разделы равномерно распределяются по кластеру, при этом каждый из разделов содержит некоторое количество копий (replica), хранимых на других физических узлах. Физический узел, на котором располагается непосредственно сам раздел, называется его владельцем (partition owner). Таким образом, при любых манипуляциях с данными происходит прозрачное взаимодействие с физическим узлом-владельцем нужного раздела.

По умолчанию Hazelcast предоставляет 271 раздел. После запуска кластера, состоящего из единственного участника, этот участник становится владельцем всех разделов (см. рисунок 5).

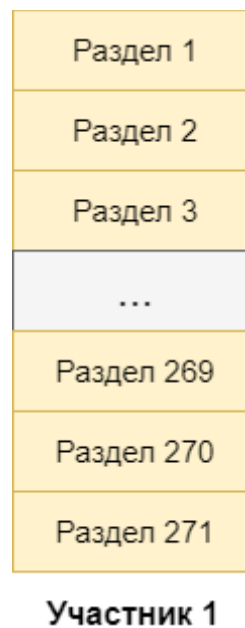


Рисунок 5 – Разбиение на разделы при работе с одним участником

После присоединения к кластеру второго участника разделы будут распределяться так, как продемонстрировано на рисунке 6, при условии, что резервное копирование не отключено в конфигурации Hazelcast.

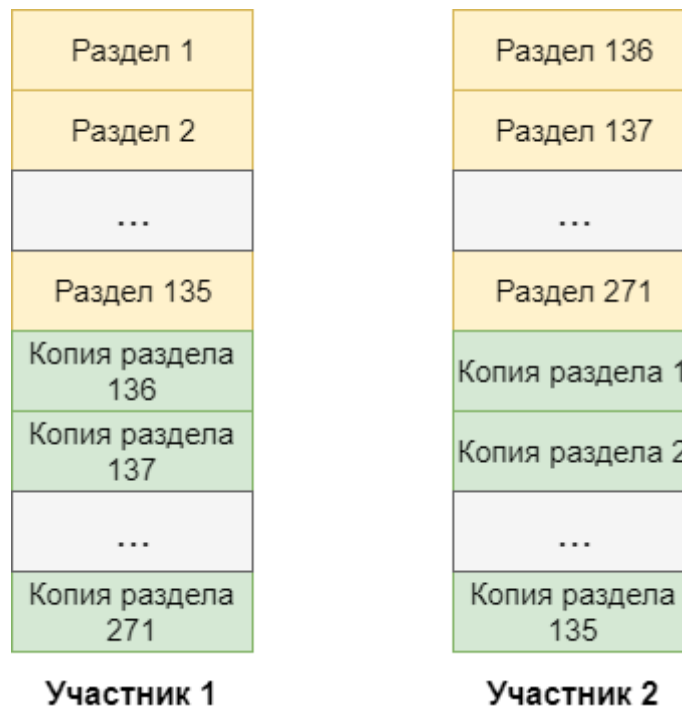


Рисунок 6 – Разбиение на разделы при работе с двумя участниками

По мере добавления новых участников и корректного завершения работы старых, происходит процесс миграции разделов и их резервных копий между текущими участниками. Информация о владельцах разделов содержится в специальной таблице, которая содержит в себе связку в виде уникального идентификатора раздела и уникального идентификатора участника, который является непосредственно владельцем. Каждый участник кластера рассылает локальную таблицу разделов остальным по определенным событиям и заданному интервалу. Таким образом гарантируется актуальность метаданных внутри кластера, и каждый участник имеет возможность получения доступа к данным, физически располагающимся на другом физическом узле.

Следует отметить, что на текущий момент данное программное обеспечение имеет подробную официальную техническую документацию, и продукт активно развивается. Разработчики предоставляют множество вспомогательных инструментов для упрощения работы с библиотекой.

Детальный анализ продукта «Hazelcast» показал, что данное программное обеспечение подходит для использования в разрабатываемом программном обеспечении в качестве распределенного хранилища данных.

Для визуализации состояния микросервисов было принято решение предоставить пользователю веб-интерфейс [17]. Помимо отображения состояния, управление микросервисами, физически располагающихся на разных узлах, может также производиться посредством веб-интерфейса, доступ к которому предоставляет один из экземпляров или несколько экземпляров системного менеджера. В качестве фреймворка для реализации интерфейса пользователя был выбран «ReactJS» [18]. Схема взаимодействия приложения пользовательского интерфейса с системным менеджером представлена на рисунке 7.

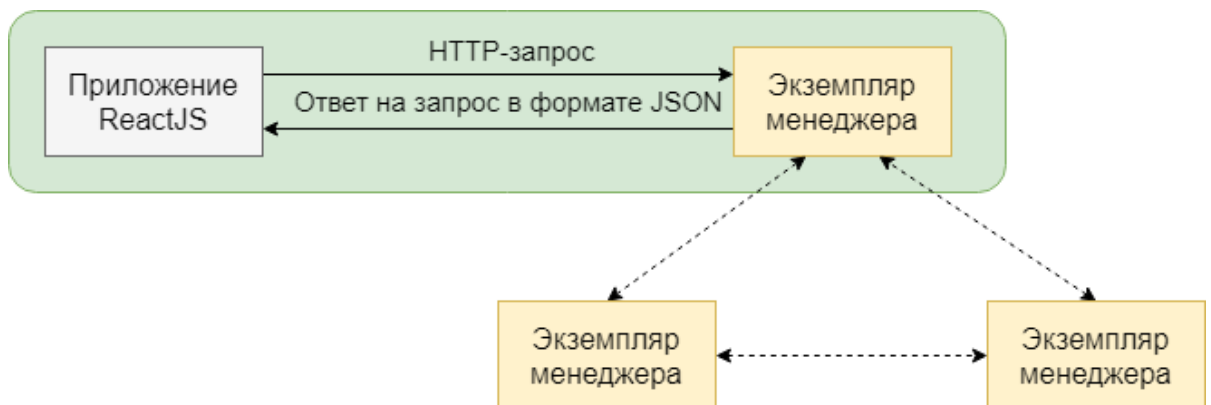


Рисунок 7 – Взаимодействие клиента с экземплярами системного менеджера

Приложение ReactJS может быть интегрировано в качестве статического ресурса, обрабатываемого HTTP-сервером Jetty. Таким образом, необходимость в отдельной установке дополнительного веб-приложения, предоставляющего пользовательский интерфейс, исчезает.

Использование Apache Maven [19] в качестве инструмента, позволяющего автоматизировать сборку приложения, является целесообразным. Во-первых, данное программное обеспечение позволяет с легкостью подключать дополнительные библиотеки для использования в проекте, с автоматическим разрешением зависимостей и конфликтов. Во-вторых, процесс сборки конечного



приложения или отдельных его частей с помощью Apache Maven является очень гибким за счет подключения и использования дополнительных конфигурируемых плагинов. На этапе разработки не всегда имеет смысл сборки проекта целиком – достаточно компиляции отдельных его частей. Однако конечному пользователю целесообразно предоставлять один JAR-файл для удобства использования конечного продукта. С помощью Apache Maven можно настроить различные варианты сборки в зависимости от конкретных целей и окружения.

Основываясь на результатах анализа различных средств разработки, описанных в данной главе, можно сделать следующие выводы:

1. Использование представленных средств и библиотек позволит организовать качественный процесс разработки и отладки приложения.
2. Все вышеописанные библиотеки имеют открытый исходный код, соответственно, конечный продукт будет полностью открытым, что является немаловажным требованием.
3. Кроссплатформенность приложения гарантируется самими средствами разработки.
4. Легковесность приложения и простота эксплуатации обеспечивается за счет отсутствия необходимости установки и настройки сторонних приложений и выделения дополнительных ресурсов.

## **Глава 4. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ИНСТРУМЕНТА**

### **4.1 Архитектура приложения**

Основываясь на перечне требований, выдвинутых к программному инструменту, было принято решение о применении следующих подходов при проектировании системного менеджера:

1. Экземпляр менеджера является отдельным процессом, который должен быть установлен на каждый физический узел кластера.
2. В контексте операционной системы каждый экземпляр является родителем всех процессов (группы микросервисов), наблюдение за которой должно осуществляться.
3. Все экземпляры являются членами кластера Hazelcast.
4. Дополнительные сервисы для осуществления мониторинга и уведомлений встраиваются в системный менеджер в качестве плагинов. HTTP-сервер и веб-интерфейс пользователя входят в состав экземпляра. Запуск дополнительных процессов не требуется.
5. Процесс взаимодействия пользователя со всеми экземплярами системного менеджера происходит посредством веб-интерфейса, предоставляемым одним или несколькими экземплярами. Коммуникация между экземплярами происходит через специальные прокси-объекты и стандартные средства распределенного хранилища Hazelcast.
6. Конфигурация экземпляра хранится в одном файле в формате XML. Путь к конфигурационному файлу передается в качестве параметра при запуске приложения.
7. Программный инструмент предоставляется конечному пользователю в виде одного JAR-файла, что способствует доступной эксплуатации инструмента

без настройки и установки дополнительных сервисов, баз данных и прочих программных продуктов.

Основные составляющие разработанного системного менеджера представлены на рисунке 8.



Рисунок 8 – Составляющие системного менеджера

Рассмотрим реализацию и архитектуру каждой составляющей программного менеджера более детально.

Менеджер конфигурации необходим для работы с файлом настроек. С его помощью осуществляется поиск конфигурационного файла в случае, если он не

указан явно, проверка корректности содержимого файла и его последующая загрузка, необходимая для инициализации системного менеджера. Результатом операции загрузки конфигурационного файла является модель данных в виде java-объекта.

На основе полученного объекта инициализируется менеджер процессов. Для каждого микросервиса (процесса) создается специальный конфигурируемый объект, который хранится в менеджере процессов. В состав данного объекта входят:

1. Контроллер процесса, с помощью которого осуществляется управление микросервисом. Каждый контроллер ассоциируется с реальным процессом операционной системы, что позволяет автоматизировать, например, процесс перезапуска сервиса в случае необходимости (микросервис аварийно завершил работу).
2. Сервис мониторинга процессов. Данный сервис предоставляет набор фабрик, с помощью которых создаются конкретные реализации, позволяющие осуществлять мониторинг микросервисов.
3. Информация о текущей сессии микросервиса и его состоянии.
4. Набор методов, необходимых для реализации операций управления, выполняемых по отношению к микросервисам.

Локальная синхронизация состояний осуществляется посредством очереди событий. Контроллер регистрирует изменение состояния, создает специальный объект события в зависимости от типа изменения, наполняет его нужными данными и добавляет в очередь. Обработчик событий получает этот объект, распознает тип события и регистрирует его в контексте менеджера процессов.

Любое локальное изменение состояния в контексте менеджера должно быть отражено в распределенном хранилище. Взаимодействие с ним осуществляется с помощью менеджера кластера. После обработки события об изменении состояния

микросервиса формируется объект, который передается в менеджер кластера для синхронизации данных. Помимо синхронизации состояний менеджер кластера предоставляет набор специальных прокси-объектов для управления любым экземпляром системного менеджера и, соответственно, любым микросервисом, входящим в состав распределенной системы. Hazelcast предоставляет стандартные возможности реализации шаблона «издатель-подписчик», что способствует обработке событий, поступающих с одних узлов кластера на другие узлы. Схема реализации обмена событиями между участниками кластера представлена на рисунке 9.



Рисунок 9 – Обмен событиями между экземплярами

В данном случае событие не отражает изменение состояния микросервисов или экземпляра, а является командой, следовательно, оно относится к другому типу событий. Соответственно, обработчик событий делегирует управление исполнителю команд. В зависимости от имени поступившего события (команды), исполнитель предоставляет нужную фабрику, которая в очередь позволит создать нужную реализацию для выполнения. Все команды выполняются в специальном контексте, который ассоциируется с текущим экземпляром системного менеджера на время выполнения. Таким образом, разработчики получают возможность реализации собственных команд и их внедрения в программный продукт в

качестве модулей. В базовой версии программного продукта реализован следующий набор команд:

1. Запуск группы микросервисов.
2. Остановка группы микросервисов.
3. Перезагрузка конфигурационного файла экземпляра менеджера.
4. Запуск отдельного микросервиса.
5. Остановка отдельного микросервиса.
6. Выключение экземпляра менеджера.

Выключение экземпляра влечет за собой остановку всех микросервисов, контроль которых осуществлялся данным экземпляром. Перезагрузка конфигурационного файла может быть полезна, например, в случае изменения настроек отдельных микросервисов или же добавления новых микросервисов. При этом те микросервисы, настройки которых не были модифицированы, продолжают свою работу в обычном режиме.

НТТР-сервер и веб-интерфейс встроены в конечное приложение. Установка дополнительных баз данных, платформ, серверов или фреймворков не требуется. Так как системный менеджер является децентрализованным инструментом, нет необходимости включать НТТР-сервер в настройках всех экземпляров. Однако для достижения максимальной отказоустойчивости и предоставления одного множества ролей всем экземплярам системного менеджера, НТТР-сервер может быть активирован на каждом физическом узле.

## **4.2 Машина состояний процесса**

Контроллеры процессов осуществляют управление микросервисами и регистрируют изменения состояний. Реализация методов для запуска процесса, его остановки и проверки состояния (запущен или нет) предусмотрена в стандартных библиотеках языка программирования Java. Однако для

возможности более детального контроля микросервисов появилась необходимость разработки собственного набора состояний процесса, а также потокобезопасного алгоритма перехода из состояния в состояние. Машина состояний процесса представлена на рисунке 10.

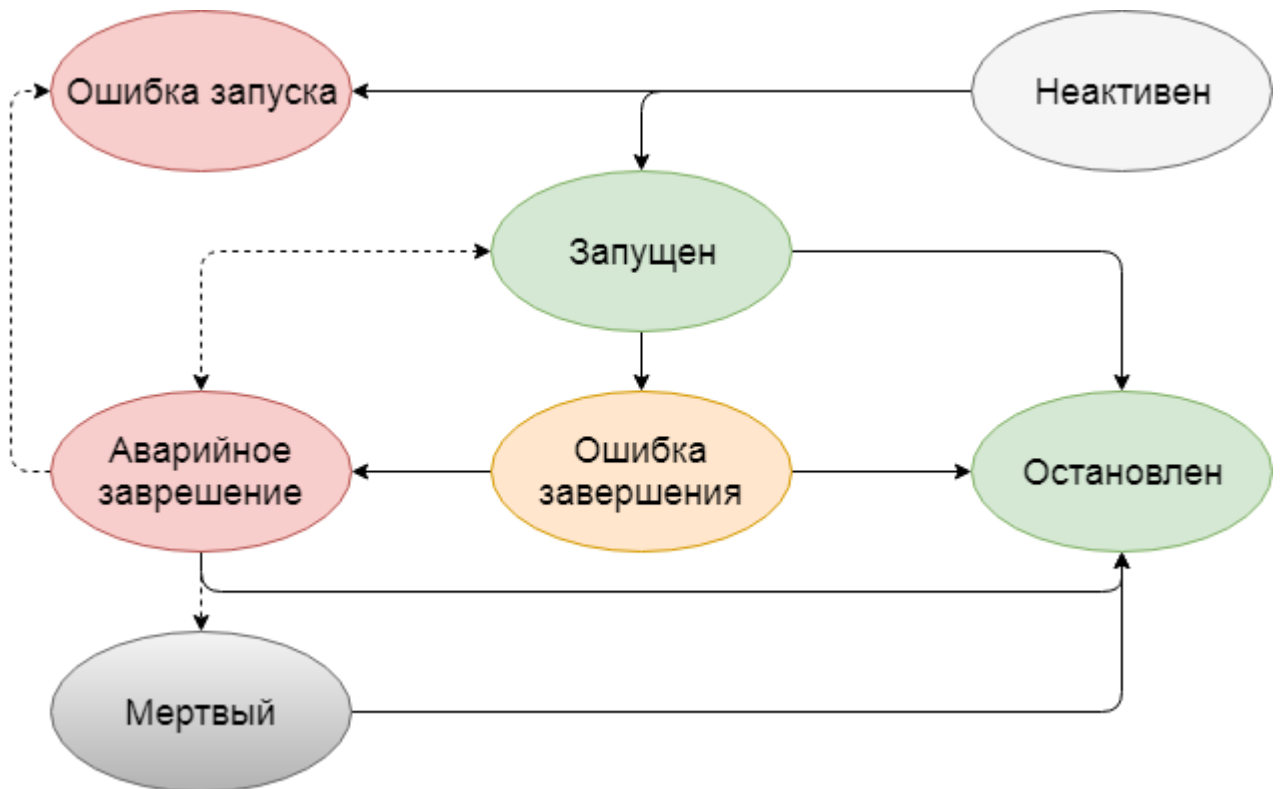


Рисунок 10 – Машина состояний процесса

В момент инициализации экземпляра на этапе загрузки конфигурационного файла происходит создание контроллеров для каждого микросервиса посредством специального провайдера. В контексте созданного контроллера микросервис находится в состоянии «Неактивен». Затем происходит процесс конфигурирования контроллера, в ходе которого секция настроек текущего микросервиса применяется к объекту ассоциированного контроллера. После успешной инициализации приложения пользователю становится доступен набор функциональных возможностей для запуска микросервисов, управление которыми осуществляет экземпляр менеджера.

Так как все настройки, необходимые для корректной работы контроллера, считываются с конфигурационного файла, а пользователь может, например, допустить ошибку в команде запуска того или иного микросервиса, контроллер не может предоставить гарантии, что процесс будет запущен. Следовательно, появляется необходимость в состоянии «Ошибка запуска». При переходе процесса в данное состояние контроллер завершает свою работу.

В случае успешного запуска процесса контроллер изменяет состояние на «Запущен». Любое незапланированное завершение работы микросервиса инициирует переход в состояние «Аварийное завершение». Поведение контроллера при переходе в данное состояние зависит от конфигурации. Таким образом, процесс может перейти в одно из четырех состояний:

1. «Ошибка запуска». Контроллер пытается запустить процесс вновь, однако в силу некоторых факторов выполнение команды запуска заканчивается ошибкой.
2. «Запущен». Контроллер пытается запустить процесс вновь, попытка запуска удачная.
3. «Остановлен». В момент регистрации контроллером аварийного завершения микросервиса пользователь выполнил команду остановки этого микросервиса посредством экземпляра системного менеджера. В таком случае пропадает необходимость попытки перезапуска процесса; процесс считается остановленным.
4. «Мертвый». Контроллер не пытается перезапустить процесс и завершает свою работу.

Помимо пользователя и внешних факторов, влияющих на работу процессов, возможность завершения работы процесса с последующим перезапуском предоставляется сервисам мониторинга. В конфигурационном файле для каждого микросервиса задается политика остановки. Таким образом, для микросервисов,



входящих в состав одной группы, можно задавать различные реализации остановки. Однако гарантировать завершение работы микросервиса по команде пользователя во всех случаях контроллер не может. В связи с необходимостью отражения ошибок остановки в контексте экземпляра менеджера, было введено состояние «Ошибка завершения». Впоследствии процесс может перейти в одно из двух состояний:

1. «Аварийное завершение». Процесс перейдет в данное состояние в случае, если изначальная остановка процесса была вызвана с целью последующего запуска сервисом мониторинга.
2. «Остановлен». Процесс перейдет в данное состояние в случае изначальной преднамеренной остановки процесса пользователем посредством экземпляра менеджера.

В случае корректной остановки микросервиса, контроллер завершает свою работу, изменяя состояние на «Остановлен».

### **4.3 Машина состояний экземпляра менеджера**

Помимо машины состояний процессов (микросервисов) необходима реализация машины состояний экземпляра менеджера и внутренняя синхронизация исполнителя команд. В один момент времени экземпляр системного менеджера может выполнять только одну команду, при этом доступность выполнения команды напрямую зависит от состояния экземпляра. В некоторых случаях необходимо учитывать также состояние микросервиса (например, при выполнении команды остановки микросервиса, необходимо проверить, не был ли он остановлен ранее). В большинстве случаев состояние экземпляра подвергается изменению в процессе выполнения команд, заданных пользователем. Однако при переходе одного или нескольких микросервисов в состояние «Ошибка завершения» экземпляр переходит в состояние «Сбой»

автоматически. Машина состояний экземпляра менеджера представлена на рисунке 11.

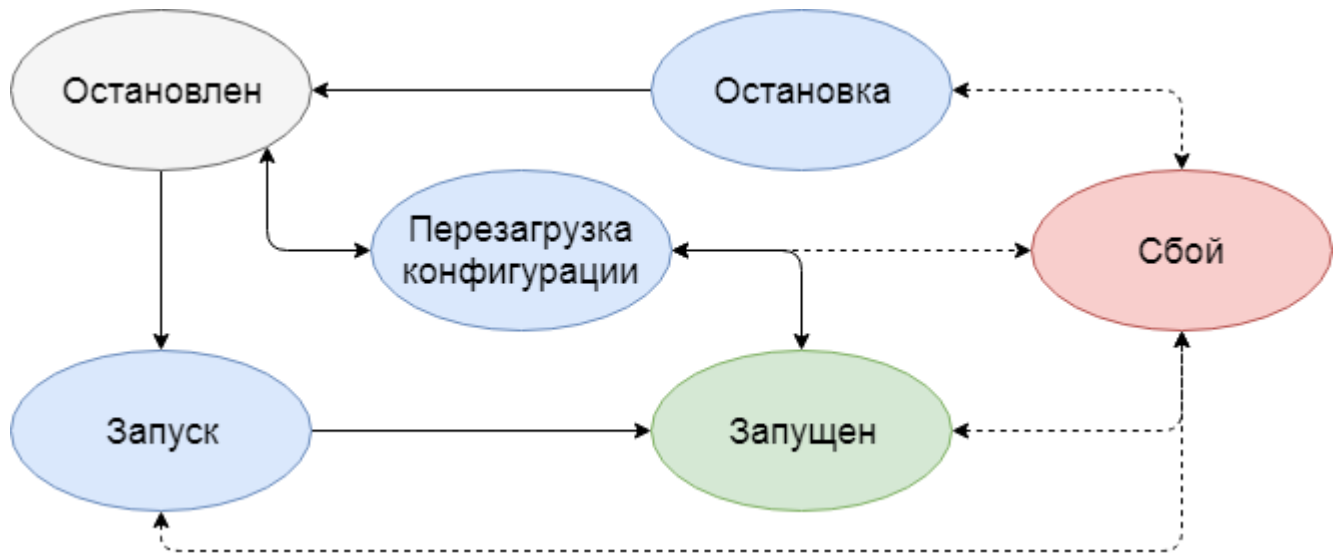


Рисунок 11 – Машина состояний экземпляра менеджера

Состояния «Запуск», «Остановка» и «Перезагрузка конфигурации» являются промежуточными состояниями. В данные состояния экземпляр переходит в процессе выполнения команд «Запуск группы микросервисов», «Остановка группы микросервисов» и «Перезагрузка конфигурационного файла экземпляра менеджера». В случае пребывания менеджера в одном из трех указанных состояний исполнение других команд автоматически блокируется до момента завершения текущей команды. События, которые не связаны с изменением состояния процессов, записываются в контекст выполнения команды, и их обработка так же откладывается до завершения.

Переход экземпляра менеджера в состояние «Сбой» не регулируется пользователем. Пока хотя бы один микросервис пребывает в состоянии «Ошибка завершения», работа экземпляра считается некорректной, и, следовательно, выполнение любых команд блокируется на программном уровне. Как только процесс перейдет в состояние «Аварийное завершение» или «Остановлен»,

экземпляр автоматически изменит свое состояние согласно алгоритму в зависимости от предыдущего состояния до регистрации сбоя. Переходу из состояния «Сбой», например, в состояние «Запущен» может способствовать принудительное завершение работы процесса, который инициировал ошибку, со стороны пользователя.

Рассмотрим следующую ситуацию: появилась необходимость замены одного микросервиса другим, при которой остановка работы всех микросервисов, контроль которых осуществляется на данном физическом узле, не требуется. В таком случае пользователю необходимо модифицировать содержимое конфигурационного файла (удалить секцию с описанием микросервиса, который необходимо остановить и добавить новую секцию с описанием микросервиса, который должен быть запущен). После чего пользователь может выполнить команду перезагрузки конфигурационного файла, не прибегая к остановке экземпляра менеджера. Исполнитель команды считает измененный файл с конфигурацией и попытается остановить старый микросервис, так как его описание отсутствует в новом файле настроек. В случае ошибки завершения работы микросервиса, менеджер зарегистрирует его как «исключенный процесс» и продолжит наблюдение за ним после перехода в состояние «Сбой». Как только микросервис остановится, контроллер, ассоциированный с ним, завершит свою работу, а менеджер процессов прекратит выполнять контроль состояния исключенного микросервиса. Экземпляр менеджера продолжит свою работу в штатном режиме.

Визуализация для пользователя вышеописанной ситуации подразумевает отображение микросервиса в списке процессов экземпляра менеджера с отметкой, что данный процесс исключен, посредством веб-интерфейса.

В момент выполнения команд все события, связанные с изменениями состояний процессов регистрируются и обрабатываются экземпляром точно так же, как и в случае режима обычного наблюдения.

#### **4.4 Проектирование программного интерфейса для мониторинга микросервисов**

Потребность предоставления другим разработчикам программных интерфейсов для реализации собственных алгоритмов мониторинга обусловлена наличием ряда специфических особенностей различных корпоративных информационных систем. Интерфейсы и протоколы взаимодействия микросервисов друг с другом могут отличаться, микросервисы могут предоставлять разные наборы данных и метрик, которые представляют интерес для администратора системы. В связи с этим, разработка единой универсальной программной реализации, позволяющей осуществлять семантический мониторинг любого процесса, не представляется возможной. Соответственно, целесообразным решением является поддержка API для разработчиков.

Для реализации собственных подходов к мониторингу процессов необходимо расширить базовый абстрактный класс под названием «AbstractSessionMonitor», который размещается в пакете «solvo.jwatch.process.services.impl». В базовом классе реализован ряд методов, позволяющих получить настройки инструмента мониторинга, которые указаны в исходном конфигурационном файле в виде словаря «ключ-значение», данные о текущей сессии процесса, опубликовать событие, которое может быть сформировано по результатам выполнения мониторинга, и завершить работу процесса с последующим запуском. Перезапуск процесса может быть полезен, например, в случае, если процесс длительное время не отвечает на сетевые запросы.

Экземпляр осуществляет мониторинг процесса следующим образом. В момент регистрации события о том, что процесс запущен, происходит обращение к сервису мониторинга для получения нужных фабрик. Таким образом, если в конфигурационном файле для микросервиса «cryptoman» указан инструмент мониторинга под названием «CryptoMonitor» с настройками «interval: 60, onFail:

interrupt», реализация такого инструмента будет создана с помощью фабрики, которая ассоциирована с именем «CryptoMonitor». Настройки будут переданы в качестве java-объекта, реализующего интерфейс «Map», где «interval» и «onFail» будут являться ключами, а «60» и «interrupt» значениями соответственно. После создания объекта будет вызван метод «start», который определяется в базовом классе как абстрактный. Соответственно, разработчику необходимо реализовать данный метод. В случае завершения работы процесса гарантируется вызов метода под названием «complete», реализация которого так же определяется разработчиком.

Все события, издателем которых являются объекты для осуществления мониторинга, регистрируются в экземпляре системного менеджера для последующей визуализации данных, полученных в процессе мониторинга, посредством веб-интерфейса. Следует отметить, что для одного микросервиса может быть активировано множество различных реализаций мониторинга, которые не зависят друг от друга.

Добавление собственных сервисов мониторинга не требует вмешательства в исходный код продукта. Разработчику необходимо позаботиться лишь об уникальности имени фабрики, которая создает объект с нужной реализацией. Инструменты мониторинга подключаются в качестве модулей, что способствует гибкости разработанного продукта.

## **Глава 5. СИНХРОНИЗАЦИЯ ДАННЫХ В РАСПРЕДЕЛЕННОМ ХРАНИЛИЩЕ**

### **5.1 Структурирование данных в Hazelcast**

Все локальные изменения состояний в контексте экземпляра системного менеджера должны быть отражены в распределенном хранилище Hazelcast для последующей визуализации данных с помощью веб-интерфейса. Целостность данных является очень важной составляющей в реализации распределенных систем, соответственно, необходимо выработать оптимальный механизм структурирования данных в хранилище и разработать алгоритм синхронизации состояний. Операции записи и чтения при работе с распределенным хранилищем Hazelcast являются сетевыми. Соответственно, необходимо учитывать возможные перебои в работе сети, результатом которых будет являться ошибка выполнения таких операций.

Hazelcast предоставляет возможность хранения данных в памяти в двух форматах:

1. Двоичный формат. Все ключи и значения хранятся в сериализованном двоичном виде.
2. Объектный формат. Все ключи хранятся в двоичном формате, значения – в объектном формате. При использовании данного формата операция чтения значения возвращает копию объекта.

Соответственно, при использовании объектного формата для выполнения чтения данных необходим процесс сериализации объекта на физическом узле, являющимся владельцем раздела, а затем десериализация объекта на физическом узле, который непосредственно вызывал операцию чтения. При использовании двоичного формата требуется только процесс десериализации. По аналогии с операцией чтения, операция записи происходит быстрее при использовании

двоичного формата, так как необходимость в десериализации объекта отсутствует. Для увеличения производительности при работе с распределенным хранилищем был выбран двоичный формат данных.

В определение «глобального состояния» экземпляра системного менеджера входят следующие составляющие:

1. Объект, представляющий состояние экземпляра.
2. Объекты, представляющие состояния всех микросервисов, включая данные их мониторинга.
3. Объект, представляющий журнал выполнения команд на данном экземпляре менеджера.
4. Объект, представляющий текущую конфигурацию, используемую менеджером.

Для организации целостности данных изменение значения любого из свойств описанного множества объектов должно быть зафиксировано в распределенном хранилище. Соответственно, появляется необходимость проектирования структуры хранилища. Рассмотрим основные возможные варианты структурирования данных [20].

В случае записи глобального состояния экземпляра при любом локальном изменении гарантируется целостность данных в распределенном хранилище. Однако данный подход имеет ряд серьезных недостатков:

1. Появляется избыточность данных. Изменение состояния одного микросервиса влечет за собой перезапись состояний других микросервисов, которые оставались неизменными.
2. Следствием избыточности данных является прирост сетевого трафика, соответственно операции записи данных в распределенное хранилище будут происходить медленнее.

3. Время, необходимое на создание «снимка» глобального состояния напрямую зависит от количества микросервисов, контроль за которыми должен осуществляться.

При работе с частичным обновлением свойств распределенного объекта, который инкапсулирует в себе глобальное состояние экземпляра, пропадает необходимость в создании «снимка» всего состояния на стороне менеджера. Однако необходимо учитывать внутреннюю реализацию операций чтения и записи в распределенном хранилище Hazelcast: операция чтения всегда возвращает копию объекта и модификация данной копии никак не отражается на объекте, который находится в хранилище. Следовательно, после изменения копии необходимо совершить операцию записи, для которой, в свою очередь, требуется сериализация объекта. Соответственно, избыточность данных при записи сохраняется. Таким образом, можно сделать вывод, что целесообразным решением является логическое разбиение данных на небольшие структуры в распределенном хранилище для их последующей частичной модификации.

Разработанный механизм структуризации распределенных данных в хранилище Hazelcast представлен на рисунке 12. Рассмотрим используемые структуры данных более детально.

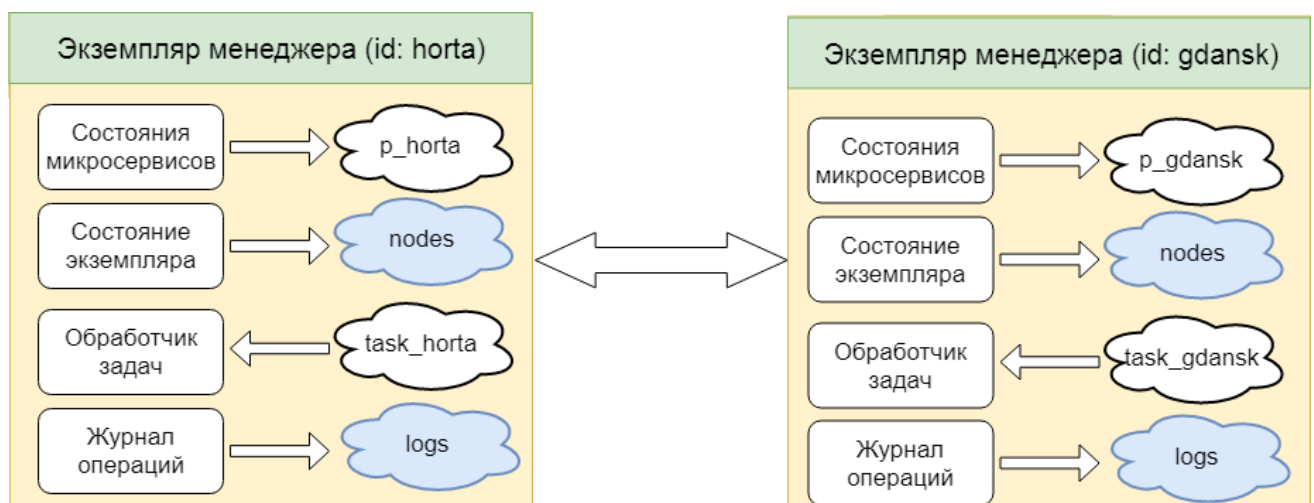


Рисунок 12 – Структурирование распределенных данных



Для хранения состояния микросервисов динамически создается распределенный словарь, имя которого генерируется с помощью конкатенации префикса «process» и уникального идентификатора экземпляра менеджера. Ключом является имя процесса, описанное в конфигурационном файле, а значением – объект, который инкапсулирует в себе копию состояния микросервиса в контексте менеджера и его конфигурацию. Соответственно, при изменении состояния микросервиса экземпляр менеджера должен сгенерировать объект, который включает в себя текущее состояние данного микросервиса и текущее состояние экземпляра (переход микросервиса в состояние «Ошибка завершения» влечет за собой переход экземпляра в состояние «Сбой»). После чего, сгенерированный объект должен быть передан менеджеру кластера для последующей перезаписи в распределенное хранилище. Состояния экземпляров, в свою очередь, сохраняются в распределенном словаре под названием «nodes». Ключом является уникальный идентификатор экземпляра, в качестве значения используется неизменяемый объект, отражающий текущее состояние. Следовательно, необходимо применение транзакций, так как изменение состояния внутри экземпляра менеджера может повлечь за собой перезапись данных в разных распределенных структурах.

Для реализации шаблона «издатель-подписчик» используются стандартные распределенные объекты, которые предоставляет Hazelcast. Данные объекты реализуют интерфейс под названием «ITopic». Для каждого экземпляра менеджера динамически создается распределенный объект, реализующий данный интерфейс. Имя этого объекта формируется по аналогии с именем словаря, хранящим состояния процессов: префикс «task» конкатенируется с уникальным идентификатором экземпляра. В момент инициализация менеджера кластера происходит подписка на прослушивание событий, поступающих в данный распределенный объект. В качестве обработчика выступает специальный класс,

который реализует интерфейс «MessageListener», входящий в состав библиотеки Hazelcast.

Процессы выполнения всех команд регистрируются в распределенном журнале операций в словаре, где ключом является уникальный идентификатор экземпляра, а значением – список объектов, представляющий результаты выполнения команд в контексте данного экземпляра.

## **5.2 Алгоритм синхронизации данных**

Рассмотрим следующую ситуацию: микросервис «CryptoMan» переходит в состояние «Ошибка завершение» из состояния «Запущен» и продолжает свою работу. Следовательно, возникает событие, которое будет обработано экземпляром системного менеджера. Сам экземпляр перейдет непосредственно в состояние «Сбой». Данные изменения должны быть отражены в распределенном хранилище, однако запись является сетевой операцией, при выполнении которой может возникнуть ошибка, и целостность данных будет нарушена. Далее, микросервис «BillingMan» может перейти в состояние «Аварийное завершение» вследствие сбоя в работе. Согласно алгоритму, будет подготовлен объект, инкапсулирующий в себе состояния экземпляра («Сбой») и сервиса «BillingMan» («Аварийное завершение»), и передан менеджеру кластера для записи. В случае если на этот раз операция записи завершится успешно, состояние микросервиса «CryptoMan» в распределенном хранилище будет оставаться неактуальным («Запущен»), несмотря на то, что последнее обновление данных в кластере было удачным. Соответственно, появляется необходимость в разработке алгоритма для выполнения синхронизации состояний между экземплярами и распределенным хранилищем [21]. Данный алгоритм в упрощенном виде представлен на рисунке 13.

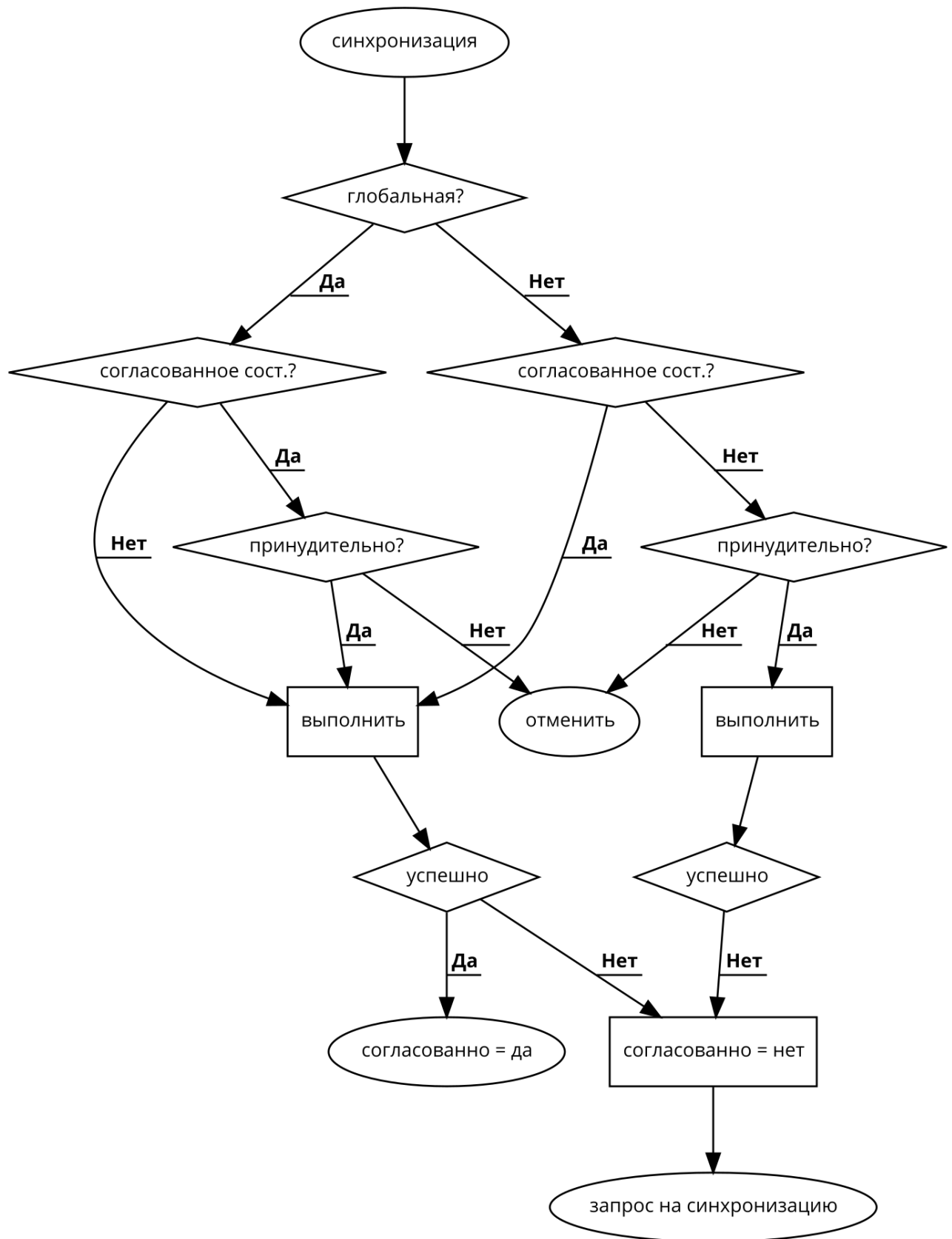


Рисунок 13 – Алгоритм синхронизации состояний

Все задачи на запись, поступающие в менеджер кластера, делятся на два типа: частичное обновление и глобальное обновление. Помимо этого, задача может быть принудительной. В случае ошибки исполнения задачи частного обновления менеджер кластера переходит в состояние «несогласованный» и отправляет запрос экземпляру на глобальную синхронизацию состояний. Экземпляр подготавливает объект, представляющий из себя «снимок» глобального состояния и на его основе формирует новую задачу на запись, которая передается в очередь задач менеджеру кластера. Если менеджер кластера находится в состоянии «несогласованный», то все непринудительные задачи частичной модификации хранилища отвергаются до успешной глобальной синхронизации. Сразу после выполнения успешной глобальной синхронизации менеджер кластера перейдет в состояние «согласованный» и продолжит работу в обычном режиме.

Следует отметить, что помимо выполнения глобальной синхронизации в случае зафиксированных ошибок в ходе частичного обновления, целесообразно выполнять данный процесс по таймеру с заданным интервалом. Для этого в менеджер кластера был добавлен специальный объект-планировщик, основной задачей которого является запрос объекта-задания глобальной синхронизации через внутренние механизмы экземпляра системного менеджера.

### **5.3 Завершение работы экземпляра системного менеджера**

Помимо попытки остановки всех микросервисов, контроль за которыми осуществляет экземпляр менеджера, корректное завершение работы должно быть зафиксировано в распределенном хранилище. Hazelcast предоставляет набор стандартных функциональных возможностей для контроля участников кластера. Одним из способов наблюдения за участниками, которые отключились от кластера, является расширение класса «MembershipAdapter», который находится в пакете под названием «com.hazelcast.core». Необходимо переопределить метод

«memberRemoved», который принимает единственный параметр типа «MembershipEvent». С помощью данного параметра можно получить доступ непосредственно к объекту типа «Member», в котором хранится основная информация об отключившемся участнике.

При получении события о команде «Выключение экземпляра менеджера», вызывается ряд методов необходимых для остановки менеджера кластера:

1. Выполняется остановка планировщика синхронизации.
2. Завершается работа объекта, отвечающего за выполнение задач записи данных в распределенное хранилище.
3. Формируется задача для удаления всех структур данных, которые хранят в себе состояние выключаемого экземпляра. Создается транзакция для обеспечения сохранности согласованности данных, в контексте которой выполняется задача.
4. После успешного удаления объекту-участнику устанавливается атрибут «задача очистки завершена».
5. Происходит остановка экземпляра Hazelcast, ассоциированного с данным экземпляром менеджера. Корректная остановка Hazelcast подразумевает запуск процесса миграции всех разделов, владельцем которых является выключаемый экземпляр. Таким образом, в случае успешной остановки согласованность данных не будет нарушена.

Рассмотрим ситуацию аварийного завершения работы физической машины, на которой запущена группа микросервисов и экземпляр системного менеджера. Необходимо учитывать механизмы хранения внутри программного инструмента Hazelcast, а именно, распределение данных по всем участникам кластера. Следовательно, экземпляр «HORTA» может являться владельцем разделов, в которых хранятся данные о состоянии экземпляра «GDANSK». При отсутствии резервных копий состояния экземпляра «GDANSK» на других физических узлах кластера данные будут потеряны в случае аварийного завершения работы

«HORTA». Это еще раз подчеркивает необходимость контроля участников кластера в момент их отключения. Для предотвращения нарушения целостности данных внутри распределенного хранилища, по событию отключения участника от кластера каждый экземпляр создает внеплановую задачу глобальной синхронизации состояний. Данная задача является «принудительной» и вынуждает менеджер кластера выполнить ее вне зависимости от того, находится ли он в состоянии «согласованный» или нет.

## Глава 6. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

В процессе разработки отдельных модулей и сервисов, входящих в состав конечного приложения, появилась необходимость прибегнуть к автоматизированному единичному тестированию. Для проверки работоспособности менеджера кластера был разработан ряд модульных тестов. Hazelcast предоставляет возможность создания более чем одного участника кластера в рамках одной виртуальной машины Java. Каждый участник имеет собственную конфигурацию и собственный набор объектов, необходимый для работы, соответственно, их можно рассматривать в качестве независимых и разделенных друг от друга. Данная особенность позволяет писать и запускать кластерные модульные тесты в одном Java-приложении. Модульное тестирование было проведено с помощью фреймворка под названием «JUnit» [22].

Работоспособность конечного продукта была проверена в тестовом окружении. Экземпляры системного менеджера запускались на двух и трёх физических узлах кластера. Осуществлялся контроль различного количества микросервисов, которые были запущены на разных физических компьютерах. В виду особенностей разработанного программного продукта помимо автоматизированного модульного тестирования появилась необходимость прибегнуть к ручному тестированию приложения. В ходе ручного тестирования особое внимание было уделено следующим составляющим:

1. Проверка корректности реализации машины состояний процесса и экземпляра системного менеджера. Автоматическая попытка перезапуска микросервиса средствами экземпляра менеджера согласно правилам, прописанным в конфигурации, является одним из ключевых критических моментов в работе системного менеджера.
2. Проверка корректности исполнения команд, поступающих от пользователя, для управления микросервисами.

3. Проверка работоспособности реализаций обнаружения экземплярами друг друга в контексте кластера.
4. Контроль поведения экземпляров менеджера при имитации разрыва сетевого соединения и аварийном завершении работы одного или нескольких физических компьютеров, входящих в состав распределенной системы.
5. Проверка алгоритма синхронизации данных в распределенном хранилище Hazelcast, который был представлен в главе 5. Основным предназначением разработанного алгоритма является сохранение согласованности данных внутри распределенного хранилища.
6. Проверка корректности завершения работы экземпляров менеджера, а именно отключение микросервисов, контроль которых осуществляется, и правильное выключение менеджера кластера.
7. Проверка подключения дополнительных модулей, необходимых для мониторинга сервисов.
8. Проверка веб-интерфейса и разработанных встроенных REST-сервисов, необходимых для взаимодействия пользователя с экземпляром системного менеджера.

Программный продукт успешно прошел все стадии различных видов тестирования. На текущий момент он используется в тестовом окружении компании «СОЛВО»; происходит подготовка к внедрению приложения на сервера заказчиков.



## Глава 7. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

### 7.1 Конфигурационный файл

Конфигурация экземпляра системного менеджера хранится в формате XML [23]. В случае если путь к конфигурационному файлу не указывается при запуске менеджера в качестве аргумента командной строки, экземпляр попытается автоматически найти файл с именем «watchrc.xml» в домашней директории пользователя. Упрощенная структура конфигурационного файла представлена на рисунке 14.

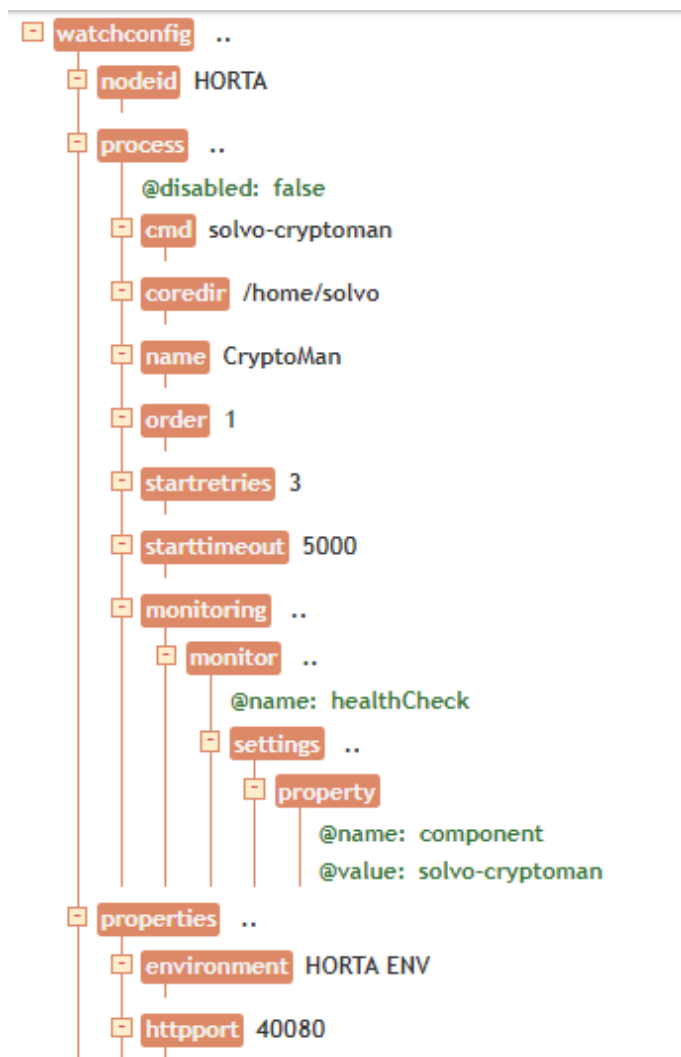


Рисунок 14 – Структура конфигурационного файла XML

Корневым элементом является тэг с именем «watchConfig». Уникальный идентификатор экземпляра помещается в тэг «nodeId». Далее описываются секции настройки микросервисов, которые входят в состав элемента «process». Количество элементов «process» должно быть равно количеству микросервисов, контроль которых необходимо выполнять. Элемент «process» поддерживает атрибут «disabled», принимающий значение «true» в случае, если микросервис не должен быть запущен. Рассмотрим подробнее множество элементов, которые являются дочерними для элемента «process»:

1. Элемент «cmd» – команда запуска микросервиса.
2. Элемент «coreDir» – директория, в которой выполняется команда запуска.
3. Элемент «name» – уникальное имя микросервиса в контексте экземпляра системного менеджера.
4. Элемент «order» – уникальный номер микросервиса, задающий порядок запуска.
5. Элемент «startRetries» – количество попыток, отведенных микросервису для последующего перезапуска в случае непредвиденного завершения работы.
6. Элемент «startTimeout» – задержка в миллисекундах перед первоначальным запуском процесса.

Данные элементы являются обязательными для каждой настройки каждого микросервиса. Помимо этого, пользователь может выбрать политику остановки, имя которой должно быть прописано в элементе «destroyer».

Секция «monitoring» является дочерним элементом тэга «process». В данном примере используется только один инструмент мониторинга с именем «healthCheck». Настройки инструмента описываются с помощью элементов «property», которым могут быть заданы атрибуты «name» (имя настройки) и «value» (значение).

Ряд дополнительных настроек, таких как реализация инструментов уведомления пользователя и конфигурация встроенного HTTP-сервера


прописываются в секции «properties». В случае перезагрузки конфигурационного файла с помощью экземпляра менеджера будут учитываться изменения связанные с элементами «process», так как ряд настроек не может быть применен без повторного запуска экземпляра.



## 7.2 Пользовательский веб-интерфейс

Веб-интерфейс пользователя входит в состав конечного приложения и активируется при включении HTTP-сервера в конфигурационном файле. Основной экран для работы пользователя с запущенными в распределенной системе экземплярами менеджера представлен на рисунке 15.

Jwatch Cluster

Task Log



ID	Node	Host address	State	Manage
0	HORTA	192.168.1.161	Stopped	
10000	GDANSK	192.168.1.162	Stopped	

Stopped








#	Process	PID	State	Retries	Enabled	Manage
1	CryptoMan	-	Inactive	0/1		
2	BillingMan	-	Inactive	0/3		

Рисунок 15 – Веб-интерфейс пользователя

В данном примере экземпляры менеджера запущены на физических компьютерах с адресами «192.168.1.161» и «192.168.1.162», которые имеют уникальные названия «HORTA» и «GDANSK» соответственно. Каждый экземпляр имеет уникальный числовой идентификатор в контексте распределенного хранилища Hazelcast, который генерируется автоматически при инициализации менеджера кластера. Для управления экземплярами пользователю

предоставляется контекстное выпадающее меню в виде шестеренки в колонке «Manage». Возможные пользовательские операции представлены на рисунке 16.

Jwatch Cluster Task Log 

ID	Node	Host address	State	Manage
0	HORTA	192.168.1.161	Running	
10000	GDANSK	192.168.1.162	Stopped	

Stopped

#	Process	PID	State	Retries	Enabled
1	NotifierMan	-	Inactive	0/1	✓
2	VesselMan	-	Inactive	0/3	✓

Start node

Stop node

Reload node

Shut down

Рисунок 16 – Меню управления экземпляром системного менеджера

В виду того, что на данный момент экземпляр «GDANSK» находится в состоянии «Остановлен» (Stopped), операция остановки экземпляра является недоступной. После успешного запуска экземпляра «GDANSK», все микросервисы, контроль за которыми осуществляется в рамках этого экземпляра перейдут в состояние «Запущен». Согласно разработанной машине состояний процесса, описанной в главе 4, изначально каждый микросервис пребывает в неактивном состоянии («Inactive»).

Переключение между экземплярами менеджера для просмотра микросервисов, контроль за которыми осуществляется, и информации о них, происходит посредством одиночного клика левой кнопки мыши на идентификатор нужного экземпляра. Текущий экземпляр менеджера, с которым работает пользователь, выделяется в таблице серым цветом.

Для просмотра подробной информации о конкретном микросервисе необходимо выполнить одиночный клик левой кнопкой мыши по имени

микросервиса (колонка «Process»). На экране пользователя появится модальное окно, визуализирующее текущее состояние выбранного микросервиса, включая информацию, полученную от подключенных инструментов мониторинга, в подробном виде. Пример такого модального окна представлен на рисунке 17.

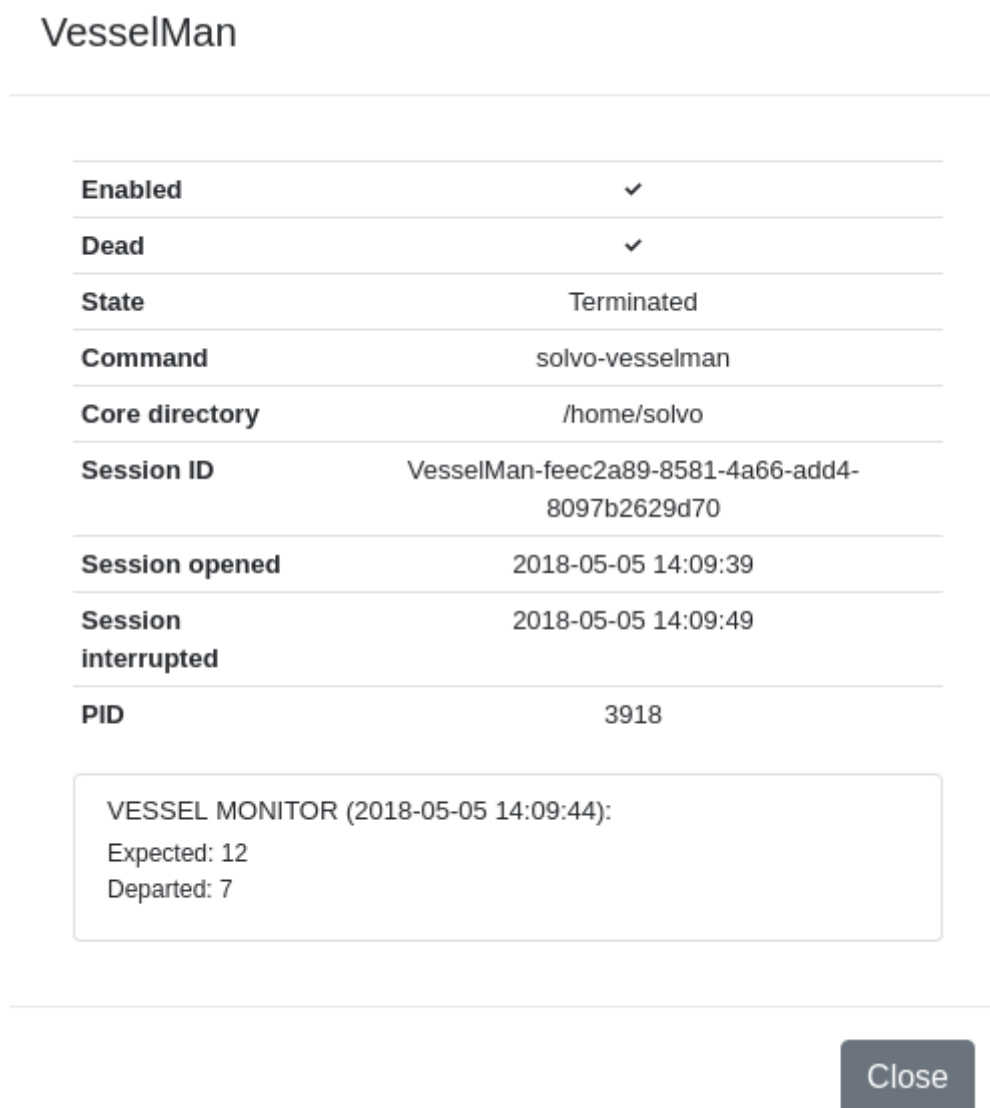





Рисунок 17 – Информация о работе микросервиса «VesselMan»

В данном случае мониторинг происходит с помощью модуля под названием «Vessel Monitor», который предоставляет текущую информацию о количестве ожидаемых и убывших судах.

Управление отдельными микросервисами происходит по аналогии с управлением экземплярами менеджера. Контекстное меню с командами для остановки и запуска микросервиса вызывается с помощью одиночного клика левой кнопкой мыши по значку шестеренки в столбце «Manage» в списке микросервисов.

Если микросервис был удален из конфигурационного файла, но его остановка не была выполнена при перезагрузке файла настроек, в веб-интерфейсе пользователя его имя будет зачеркнуто. Визуализация состояний микросервисов в таком случае представлена на рисунке 18.

Jwatch Cluster Task Log 						
ID	Node	Host address	State		Manage	
0	<del>HORTA</del>	192.168.1.161	Halting			
10000	GDANSK	192.168.1.162	Running			




Halting						
#	Process	PID	State	Retries	Enabled	Manage
1	<del>CryptoMan</del>	4361	Halting	1/1	✓	
1	ServiceMan	4410	Running	1/1	✓	
2	BillingMan	4416	Running	1/3	✓	

Рисунок 18 – Ошибка остановки процесса, удаленного из конфигурации

Пользователю доступна визуализация журнала выполненных команд на всех экземплярах системного менеджера, входящих в состав кластера. Для перехода в режим просмотра необходимо выполнить одиночный клик левой кнопки мыши на ссылку «Task Log», которая находится в верхней панели навигации. Возврат к главному экрану осуществляется посредством клика на ссылку «Jwatch Cluster». Пример журнала операций для экземпляров «HORTA» и «GDANSK» представлен на рисунке 19.


Jwatch Cluster Task Log 				
Node	Date	Description	Result	Error
10000-GDANSK	2018-05-05 14:18:46	Received: #2 SHUT DOWN NODE		
0-HORTA	2018-05-05 14:17:36	Completed: #4 RELOAD NODE	Completed	
0-HORTA	2018-05-05 14:17:21	Received: #4 RELOAD NODE		
0-HORTA	2018-05-05 14:16:52	Completed: #3 START NODE	Completed	
0-HORTA	2018-05-05 14:16:42	Received: #3 START NODE		
10000-GDANSK	2018-05-05 14:16:18	Completed: #1 STOP NODE	Completed	
10000-GDANSK	2018-05-05 14:16:13	Received: #1 STOP NODE		
0-HORTA	2018-05-05 14:10:36	Received: #2 STOP NODE		
0-HORTA	2018-05-05 14:10:36	Completed: #2 STOP NODE	Completed	
10000-GDANSK	2018-05-05 14:09:19	Completed: #0 START NODE	Completed	
10000-GDANSK	2018-05-05 14:09:09	Received: #0 START NODE		
0-HORTA	2018-05-05 14:05:20	Completed: #1 STOP NODE	Completed	
0-HORTA	2018-05-05 14:05:15	Received: #1 STOP NODE		
0-HORTA	2018-05-05 14:02:55	Completed: #0 START NODE	Completed	
0-HORTA	2018-05-05 14:02:44	Received: #0 START NODE		

Рисунок 19 – Журнал операций, выполненных в системе

Для принудительного обновления информации о глобальных состояниях всех экземпляров системного менеджера без принудительной перезагрузки веб-страницы в навигационную панель была добавлена кнопка «обновить», представленная в виде иконки.

Все исходные файлы веб-интерфейса поставляются вместе с исходными кодами конечного приложения, что предполагает их возможную дальнейшую модернизацию другими разработчиками в зависимости от потребностей или специфичности реализации конкретной корпоративной распределенной информационной системы.

## ЗАКЛЮЧЕНИЕ

Распределенные информационные системы, использующие микросервисную архитектуру, широко распространены в настоящее время. Необходимость в автоматизированном управлении и мониторинге сервисов, входящих в состав таких систем, обусловлена спецификой проектирования и реализации. В ходе анализа предметной области было выявлено, что существует потребность внедрения дополнительного программного обеспечения, позволяющего осуществлять контроль распределенной системы, а также, представлен перечень требований к такому инструменту.

Результат детального анализа существующих программных решений показал, что готовые продукты имеют ряд своих достоинств и недостатков, основными из которых являются сложность в эксплуатации и потребность в установке и настройке дополнительного программного обеспечения для работы приложения. В связи с этим было принято решение о проектировании и разработке собственного инструмента, предоставляющего набор функциональных возможностей для автоматизированного контроля и мониторинга микросервисов в распределенных системах.

В ходе выполнения практической части исследования были разработаны машины состояний процесса и экземпляра системного менеджера, а также алгоритмы для синхронизации данных в используемом распределенном хранилище. Для дальнейшей модернизации приложения был представлен набор программных интерфейсов. Разработанный программный продукт имеет открытый исходный код и распространяется под лицензией MIT. Тестирование данного инструмента производилось на серверах, предоставленных компанией «СОЛВО»; на данный момент продукт находится на этапе внедрения в систему «Solvo.TOS», что подчеркивает его практическую значимость в сфере информационных технологий.



## **СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

API – Application Programming Interface.

HTTP – HyperText Transfer Protocol.

JSON – JavaScript Object Notation.

REST – Representational state transfer.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Хорсдал К. Микросервисы на платформе .NET – СПб.: Питер, 2018 – 352 с.
2. Ужинский А.В. Методы и средства мониторинга сервисов передачи данных – LAP Lambert Academic Publishing, 2011 – 124 с.
3. Andrew S. Tanenbaum, Maarten van Steen. Distributed Systems: Principles and Paradigms – CreateSpace Independent Publishing Platform, 2016. – 716 с.
4. Комер Д., Стивенс Д. Сети TCP/IP. Том 3. Разработка приложений типа клиент/сервер для Linux/POSIX – М.: Вильямс, 2002. – 592 с.
5. Дэйт К. Введение в системы баз данных – М.: Вильямс, 2017 – 1328 с.
6. Ньюмен С. Создание микросервисов – СПб.: Питер, 2016 – 304 с.
7. Васильев А.Н. Java. Объектно-ориентированное программирование. Учебное пособие. Стандарт третьего поколения – СПб.: Питер, 2012 – 400 с.
8. Радченко И.А. Использование открытых данных в научных исследованиях // Информационное общество. 2013. № 1-2. С. 93-101
9. Стивенс Р., Раго С. UNIX. Профессиональное программирование – СПб.: Питер, 2018 – 944 с.
10. Petersen, R. L. Fedora Linux Servers with Systemd – М.: Книга по Требованию, 2016 – 508 с.
11. Apache Mesos Documentation [Электронный ресурс]. — Режим доступа: <http://mesos.apache.org/documentation/latest/>, свободный. Яз. англ. (дата обращения 12.04.2018).
12. Apache ZooKeeper Documentation [Электронный ресурс]. — Режим доступа: <https://zookeeper.apache.org/documentation.html>, свободный. Яз. англ. (дата обращения 12.04.2018).
13. Крихели А.М. Автоматизация контроля состояния процессов и мониторинг их сетевой активности. Выпускная квалификационная

- работа бакалавра. – Университет ИТМО, 2016 [Электронный ресурс]. – Режим доступа: [https://isu.ifmo.ru/pls/apex/f?p=2143:0:112183750813617::DWNLD\\_F:NO::FILE:A847C8FE02838B858CAC43E902DD4C0E](https://isu.ifmo.ru/pls/apex/f?p=2143:0:112183750813617::DWNLD_F:NO::FILE:A847C8FE02838B858CAC43E902DD4C0E), ограниченный. Яз. рус. (дата обращения 20.04.2018).
14. Шилдт Г. Java 8. Полное руководство – М.: Вильямс, 2015 – 1376 с.
15. Google Guice Wiki [Электронный ресурс]. — Режим доступа: <https://github.com/google/guice/wiki>, свободный. Яз. англ. (дата обращения 01.05.2018).
16. Hazelcast Documentation [Электронный ресурс]. — Режим доступа: <http://docs.hazelcast.org/docs/3.9/manual/html-single/index.html>, свободный. Яз. англ. (дата обращения 01.05.2018).
17. Скотт Б., Нейл Т. Проектирование веб-интерфейсов – СПб.: Символ-Плюс, 2010 – 352 с.
18. ReactJS Documentation [Электронный ресурс]. — Режим доступа: <https://reactjs.org/docs>, свободный. Яз. англ. (дата обращения 01.05.2018).
19. Apache Maven Documentation [Электронный ресурс]. — Режим доступа: <http://maven.apache.org/guides/>, свободный. Яз. англ. (дата обращения 01.05.2018).
20. Лафоре Р. Структуры данных и алгоритмы в Java – СПб.: Питер, 2017 – 704 с.
21. Фоккинк У. Распределенные алгоритмы. Интуитивный подход – СПб.: Питер, 2017 – 272 с.
22. JUnit Project Documentation [Электронный ресурс]. — Режим доступа: <https://junit.org/junit4/index.html>, свободный. Яз. англ. (дата обращения 01.05.2018).
23. JAXB Architecture [Электронный ресурс]. — Режим доступа: <https://docs.oracle.com/javase/tutorial/jaxb/intro/arch.html>, свободный. Яз. англ. (дата обращения 01.05.2018).