Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство)»
(ФГБОУ ВО «РГУ им. А.Н. Косыгина»)

На правах рукописи

Фрасын Павел Геннадьевич

РАЗРАБОТКА МЕТОДОВ УПРАВЛЕНИЯ ПРОГРАММНОЙ СРЕДОЙ АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ ТЕХНОЛОГИЧЕСКИМИ ПРОЦЕССАМИ

Специальность 2.3.3 – Автоматизация и управление технологическими процессами и производствами

ДИССЕРТАЦИЯ
на соискание ученой степени кандидата технических наук

Научный руководитель:
доктор технических наук,
доцент
Рыжкова Елена Александровна

Москва – 2025

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 6

ГЛАВА 1. АРХИТЕКТУРА ПРОГРАММНОЙ СРЕДЫ ДИСПЕТЧЕРСКОГО УРОВНЯ АСУТП И ОРГАНИЗАЦИЯ ЕЕ СОПРОВОЖДЕНИЯ 14

1.1 Программная среда диспетчерского уровня АСУТП 14

1.2 Эксплуатационное сопровождение программной среды 15

1.3 Влияние архитектуры исполнения SCADA-систем на организацию процедур сопровождения программной среды 17

1.3.1 SCADA как программный комплекс с целостным исполнением ... 17

1.3.2 SCADA как совокупность исполняемых компонентов 19

1.4 Ограничения существующих процедур сопровождения программной среды 21

1.5 Анализ научной проработанности задач сопровождения программной среды 24

1.6 Постановка задачи управления конфигурацией программной среды .. 26

1.7 Концепция и архитектура контура сопровождения 27

1.7.1 Замкнутая архитектура 28

1.7.2 Архитектура с внутренним контуром сопровождения 30

1.7.3 Архитектура с вынесенным контуром сопровождения 31

Выводы по 1 главе 34

ГЛАВА 2. МЕТОДОЛОГИЧЕСКИЕ ОСНОВЫ ОРГАНИЗАЦИИ УПРАВЛЕНИЯ И АВТОМАТИЗАЦИИ ЗАДАЧ СОПРОВОЖДЕНИЯ ПРОГРАММНОЙ СРЕДЫ АСУТП 35

2.1 Формализация конфигурации программной среды АСУТП 35

2.1.1 Модель фактической конфигурации программной среды 35

2.1.2 Критерий полноты формализованного представления фактической конфигурации 37

2.1.3 Критерий достоверности результата воспроизведения фактической конфигурации 38

2.2 Методы управления конфигурацией программной среды АСУТП 39

2.2.1 Императивный метод выполнения операций сопровождения 39

2.2.2 Декларативный метод формирования корректирующих воздействий 42

2.3 Стандартизация среды исполнения компонентов программной среды 46

2.3.1 Изолированное исполнение компонентов и воспроизводимость среды 46

2.3.2 Модель стандартизированной среды исполнения программных компонентов 53

2.4 Автоматизация сопровождения конфигурации программной среды АСУТП 56

2.4.1 Модель системы сопровождения 56

2.4.2 Архитектурные варианты размещения и ограничения применения системы сопровождения 60

2.4.3 Функционирование замкнутого контура сопровождения программной среды 62

Выводы по 2 главе 64

ГЛАВА 3. РАЗРАБОТКА СИСТЕМЫ СОПРОВОЖДЕНИЯ ПРОГРАММНОЙ СРЕДЫ АСУТП ДЛЯ ТЕХНОЛОГИЧЕСКОГО ОБЪЕКТА ВОДОЗАБОРНОГО УЗЛА 66

3.1. Условия экспериментальной верификации и требования к программной среде 66

3.1.1 Экспериментальный объект и среда внедрения 66

3.1.2 Условия проведения экспериментальной верификации методов... 68

3.1.3 Архитектура программной среды АСУТП верификационной площадки..... 68

3.1.4 Критерии успешной экспериментальной верификации 69

3.2. Реализация компонентов программной среды АСУТП..... 71

3.2.1 Опрос полевых устройств и публикация телеметрии..... 71

3.2.2 Долговременное хранение данных телеметрии..... 74

3.2.3 Визуализация параметров в интерфейсе оператора..... 76

3.3. Реализация системы автоматизированного сопровождения конфигурации программной среды АСУТП 79

3.3.1 Параметризация нормативной конфигурации компонентов 79

3.3.2 Согласование параметров взаимодействия компонентов 83

3.3.3 Получение фактической конфигурации программной среды..... 86

3.4 Формирование воспроизводимых сред исполнения 92

3.4.1 Подготовка контейнерных компонентов 92

3.4.2 Контроль исходных данных для формирования образа 93

3.4.3 Проверка структуры и безопасности программных компонентов . 96

3.4.4 Анализ итогового образа..... 98

3.5. Организация процедур сопровождения программной среды АСУТП 101

3.5.1 Этапы сопровождения программной среды и структура программного конвейера..... 101

3.5.2 Механизмы активации процедур и управление параметрами выполнения..... 103

3.5.3 Контроль и безопасность операций сопровождения 104

Выводы по 3 главе 107

ГЛАВА 4. ЭКСПЕРИМЕНТАЛЬНАЯ ВЕРИФИКАЦИЯ И АНАЛИЗ ЭФФЕКТИВНОСТИ СИСТЕМЫ СОПРОВОЖДЕНИЯ ПРОГРАММНОЙ СРЕДЫ АСУТП 108

4.1 Экспериментальная проверка корректности функционирования методов управления конфигурацией программной среды АСУТП 108

4.1.1 Корректность выполнения процедур сопровождения при отсутствии конфигурационных отклонений 109

4.1.2 Экспериментальная проверка автоматизированного выявления конфигурационных расхождений 111

4.1.3 Экспериментальная проверка формирования и применения корректирующих воздействий 113

4.2. Количественная оценка эффективности системы сопровождения программной среды в промышленной эксплуатации 117

4.2.1 Порядок количественной оценки трудоемкости процедур сопровождения 118

4.2.2 Сопоставление трудоемкости сопровождения в различные периоды эксплуатации 120

Выводы по 4 главе 127

ЗАКЛЮЧЕНИЕ 129

СПИСОК ТЕРМИНОВ 131

СПИСОК ЛИТЕРАТУРЫ 132

ПРИЛОЖЕНИЯ 150

ВВЕДЕНИЕ

Актуальность темы

В интегрированных автоматизированных системах управления (ИАСУ) нижний уровень связан с управлением технологическими процессами и формированием оперативной информации о ходе производства. На данном уровне функционирует автоматизированная система управления технологическими процессами (АСУТП), в составе которой диспетчерская подсистема реализуется средствами SCADA-систем. Они обеспечивают мониторинг состояния технологического процесса, визуализацию параметров, регистрацию и архивирование данных, обработку событий и передачу команд на уровень управления оборудованием.

Корректность функционирования SCADA-систем в значительной степени определяется согласованностью средств программной среды диспетчерского уровня АСУТП и характером их взаимодействия. Нарушения в работе программной среды приводят к искажению информационной модели технологического процесса, увеличению времени реагирования на отклонения, снижению эффективности операторского контроля и формированию условий, повышающих вероятность технологических и информационных рисков.

Фактическая конфигурация программной среды определяется текущим составом программных компонентов, их версиями, параметрами настройки и условиями межкомпонентного взаимодействия. Длительное функционирование системы сопряжено с воздействием дестабилизирующих факторов, связанных с аппаратными отказами, эксплуатационными вмешательствами и неявными дефектами программного обеспечения. Под влиянием указанных факторов в процессе эксплуатации возможно отклонение фактических параметров программной среды от нормативного конфигурационного описания при сохранении внешней работоспособности системы.

Эксплуатационное сопровождение программной среды ориентировано преимущественно на обеспечение работоспособности системы и контроль функциональных и эксплуатационных признаков ее функционирования,

наблюдаемых в процессе эксплуатации. Указанные процедуры направлены на подтверждение выполнения системой заданных функций и, как правило, не предполагают регулярного детального сопоставления фактической конфигурации программной среды с ее нормативным описанием. Вследствие этого отдельные конфигурационные расхождения, не проявляющиеся на уровне функциональных признаков, могут сохраняться на ранних этапах эксплуатации и выявляться лишь при дальнейшем развитии нарушения, что сопровождается ростом неопределенности при диагностике и увеличением трудоемкости восстановительных процедур.

Указанные обстоятельства обусловливают актуальность научно-технической задачи по разработке методов формализованной организации и автоматизации сопровождения программной среды обеспечивающих подсистем диспетчерского уровня АСУТП. Предлагаемые решения ориентированы на контроль соответствия фактической конфигурации нормативному описанию в процессе эксплуатации, что позволяет повысить эффективность эксплуатационных процедур и обеспечить снижение трудоемкости контрольно-диагностических операций.

Степень разработанности темы исследования. Теоретические основы системного анализа, математических методов, компьютерного моделирования и оптимизации функционирования сложных технических систем разработаны в трудах академиков АН СССР В.М. Глушкова, В.В. Кафарова и В.А. Трапезникова, академиков РАН В.П. Мешалкина и Д.А. Новикова, а также профессоров А.Ф. Егорова и Ф.Ф. Пащенко.

В отечественных и зарубежных исследованиях представлены результаты, связанные с организацией функционирования программных и вычислительных сред сложных технических систем, включая вопросы архитектурной организации, распределения ресурсов, модернизации программных компонентов и согласованности вычислительных сред. В трудах ряда отечественных и зарубежных исследователей, в том числе А.В. Абдалова, К.И. Воловича, А.В. Котенко, В.В. Сивова, А.А. Спицына и других, а также в работах зарубежных авторов, включая

R.N. Taylor, N. Medvidovic, E. Dashofy, D. Schmidt и L. Baresi, рассмотрены вопросы построения и сопровождения программных систем.

Наличие теоретических и практических работ отечественных и зарубежных исследователей подтверждает актуальность рассматриваемой тематики и характеризует определенную степень ее научной разработанности. В существующих исследованиях рассматриваются вопросы формализованного описания и сопровождения программных систем, однако они, как правило, ориентированы на общие вычислительные и информационные среды и не учитывают эксплуатационную специфику программной среды диспетчерского уровня АСУТП, связанную с регламентированным характером изменений, необходимостью формализованного выполнения процедур сопровождения и контроля конфигурационных параметров в условиях промышленной эксплуатации. Указанные особенности определяют необходимость дальнейшего развития данных положений применительно к задачам сопровождения программной среды АСУТП.

Целью работы является разработка методов автоматизированного сопровождения конфигурации программной среды диспетчерского уровня АСУТП, обеспечивающих ее соответствие нормативному описанию в процессе эксплуатации. Для достижения указанной цели необходимо решить следующие **задачи**:

1. Сформулировать и обосновать постановку задачи управления конфигурацией программной среды диспетчерского уровня АСУТП в условиях промышленной эксплуатации.

2. Разработать модель формализованного представления фактической конфигурации программной среды диспетчерского уровня АСУТП, обеспечивающую получение параметрического представления на основе эксплуатационно доступных данных, пригодного для сопоставления с нормативным конфигурационным описанием.

3. Разработать методы автоматизированного управления конфигурацией программной среды АСУТП, обеспечивающие выявление конфигурационных отклонений, формирование корректирующих воздействий и поддержание

нормативной конфигурации в процессе эксплуатации в рамках регламентированных процедур сопровождения.

4. Разработать систему автоматизированного сопровождения программной среды диспетчерского уровня АСУТП, обеспечивающую практическую реализацию разработанных моделей и методов управления конфигурацией и их интеграцию в процессы промышленной эксплуатации.

5. Провести экспериментальную верификацию разработанных методов и системы сопровождения на промышленной программной среде диспетчерского уровня АСУТП водозаборного узла.

6. Выполнить количественную оценку эффективности системы сопровождения программной среды диспетчерского уровня АСУТП путем сопоставления фактической трудоемкости процедур сопровождения до и после ее внедрения.

Объектом исследования является архитектура и организация программной среды диспетчерского уровня АСУТП.

Предметом исследования являются процессы формирования, изменения и сопровождения конфигурации программной среды диспетчерского уровня АСУТП в процессе эксплуатации.

Методы исследования. Решение поставленных в диссертации задач основано на использовании положений системного анализа и алгоритмических методов. Для формализованного представления конфигурации программной среды и процессов ее изменения использованы элементы теории множеств. Анализ состава программной среды и взаимодействия ее компонентов выполнен с применением методов структурно-функционального и архитектурного анализа.

По тематике, методам исследования, предложенным новым научным положениям диссертация соответствует паспорту специальности научных работников 2.3.3. Автоматизация и управление технологическими процессами и производствами в части:

п. 11. Методы создания, эффективной организации и ведения специализированного информационного и программного обеспечения АСУТП,

АСУП, АСТПП и др., включая базы данных и методы их оптимизации, промышленный интернет вещей, облачные сервисы, удаленную диагностику и мониторинг технологического оборудования, информационное сопровождение жизненного цикла изделия;

п. 13. Методы планирования, оптимизации, отладки, сопровождения, модификации и эксплуатации задач функциональных и обеспечивающих подсистем АСУТП, АСУП, АСТПП и др., включающие задачи управления качеством, финансами и персоналом;

п. 15. Теоретические основы, методы и алгоритмы диагностирования (определения работоспособности, поиск неисправностей и прогнозирования) АСУТП, АСУП, АСТПП и др.

Научная новизна работы. Научная новизна работы характеризуется следующими результатами:

1. Разработана модель формализованного представления конфигурации программной среды диспетчерского уровня АСУТП, которая ориентирована на задачи эксплуатационного сопровождения и обеспечивает сопоставимость фактической и нормативной конфигураций за счет их представления в едином параметрическом виде;

2. Разработаны модели управления конфигурацией программной среды АСУТП, включающие модель формирования регламентных корректирующих воздействий на основе сопоставления фактической и нормативной конфигураций и модель их исполнительного выполнения в условиях эксплуатации;

3. Разработаны методы автоматизированного сопровождения программной среды диспетчерского уровня АСУТП, основанные на формализованном конфигурационном описании программных компонентов и разработанных моделях управления конфигурацией, обеспечивающие поддержание согласованности их состава и параметров в процессе эксплуатации;

4. Разработан метод архитектурной организации централизованного сопровождения программной среды АСУТП, основанный на вынесении функций

сопровождения в специализированный контур, функционально независимый от прикладных программных компонентов.

Теоретическая значимость работы заключается в развитии научных основ построения и сопровождения программной среды автоматизированных систем управления технологическими процессами. В работе сформулированы и обоснованы теоретические положения, направленные на формализацию представления конфигурации программной среды и процессов ее приведения и поддержания в процессе эксплуатации, что расширяет теоретическую базу проектирования архитектур и организации сопровождения сложных программных комплексов автоматизированных систем управления.

Практическая значимость заключается в возможности применения разработанных методов и программного комплекса при эксплуатации программной среды диспетчерского уровня АСУТП для формализованного контроля ее конфигурационного состояния. Реализация предложенных решений обеспечивает автоматизированное выявление конфигурационных отклонений, формирование проекта корректирующих воздействий и поддержку процедур восстановления нормативной конфигурации в регламентированных условиях эксплуатации. Использование системы позволяет снизить трудоемкость контрольно-диагностических и восстановительных операций, а также уменьшить риск накопления скрытых конфигурационных расхождений.

Практическая значимость работы подтверждается 6 свидетельствами о государственной регистрации программ для ЭВМ, зарегистрированными в Федеральной службе по интеллектуальной собственности Российской Федерации.

Основные научные положения, выносимые на защиту

1. Формализованное представление конфигурации программной среды диспетчерского уровня АСУТП, включающее описание состава программных компонентов и их параметров;

2. Императивный и декларативный методы приведения конфигурации программной среды АСУТП к заданному представлению, отличающиеся принципами формирования и применения корректирующих воздействий;

3. Архитектурная организация контура сопровождения программной среды АСУТП, предусматривающая централизованное исполнение процедур сопровождения и различные варианты его размещения в структуре программной среды;

4. Метод автоматизированного сопровождения программной среды диспетчерского уровня АСУТП, основанный на формализованном конфигурационном описании программных компонентов;

5. Программный комплекс сопровождения программной среды АСУТП, обеспечивающий реализацию формализованных методов контроля, приведения и поддержания конфигурации программных компонентов.

Реализация и внедрение научно-технических результатов работы

Результаты диссертационного исследования внедрены в деятельность ООО «Самолет-Ресурс» при эксплуатации технологического водозаборного узла жилого комплекса «Томилино Парк», а также в деятельность специализированной организации ООО «АК-Системы» в части выполнения работ по сопровождению программной среды диспетчерского уровня АСУТП, что подтверждается соответствующими актами внедрения.

В программной инфраструктуре объекта внедрена и эксплуатируется система сопровождения программной среды диспетчерского уровня АСУТП, реализующая разработанные методы и модели и основанная на формализованном описании конфигурации программных компонентов и автоматизации процедур сопровождения. Система интегрирована с компонентной SCADA-системой и применяется в условиях промышленной эксплуатации. Практическая эксплуатация подтвердила корректность разработанных решений и их применимость для сопровождения программной среды АСУТП.

Апробация работы. Основные результаты диссертационной работы докладывались на XXV Международном научно-практическом форуме «SMARTEX» (Иваново, 2022), 56-й Международной научно-технической конференции преподавателей и студентов ВГТУ (Витебск, 2023), Всероссийской научной конференции молодых исследователей с международным участием

«Инновационное развитие техники и технологий в промышленности» (ИНТЕКС-2023, Москва), XV Международной интернет-конференции «Инновационные технологии: теория, инструменты, практика» (InnoTech 2023, Пермь), научно-практической конференции имени профессора Я.В. Мильмана (Москва, 2023, 2024), III Междисциплинарной студенческой научно-практической конференции «Студенческая научно-исследовательская лаборатория: современное состояние и перспективы» (Краснодар, 2024), а также на IV Международной научно-практической конференции «Цифровая трансформация социальных и экономических систем» – DIGITAL2025 (Москва, 2024).

Личный вклад автора: представленные в данной диссертационной работе исследования, разработки и эксперименты являются результатом работы, выполненной лично автором.

Публикации. По теме диссертации опубликовано 16 печатных научных работ, в том числе 4 статей в изданиях, входящих в «Перечень ВАК» и 2 статьи «SCOPUS». Получено 6 свидетельств о государственной регистрации программ для ЭВМ.

Структура и объем работы. Диссертационная работа состоит из введения, 4 глав, заключения, списка терминов, списка литературы из 158 наименований и 8 приложений. Работа содержит 159 страниц основного текста, включая 77 рисунков и 6 таблиц.

ГЛАВА 1. АРХИТЕКТУРА ПРОГРАММНОЙ СРЕДЫ ДИСПЕТЧЕРСКОГО УРОВНЯ АСУТП И ОРГАНИЗАЦИЯ ЕЕ СОПРОВОЖДЕНИЯ

1.1 Программная среда диспетчерского уровня АСУТП

Автоматизированные системы управления технологическими процессами (АСУТП) предназначены для автоматизированного сбора, хранения и обработки информации о ходе технологического процесса, а также для формирования управляющих воздействий на объект управления (ОУ) в соответствии с заданными критериями [1].

В составе современных интегрированных автоматизированных систем управления (ИАСУ) АСУТП реализуются в виде многоуровневой иерархической системы [3, 4]. Функции оперативного контроля, визуализации состояния технологического объекта, регистрации событий, архивирования параметров и взаимодействия с оператором сосредоточены на диспетчерском уровне [2, 14].

Реализация указанных функций осуществляется средствами SCADA-систем [5, 6], относящихся к прикладному программному обеспечению (ПО) и функционирующих в составе программной среды АСУТП [7, 8] (рисунок 1).

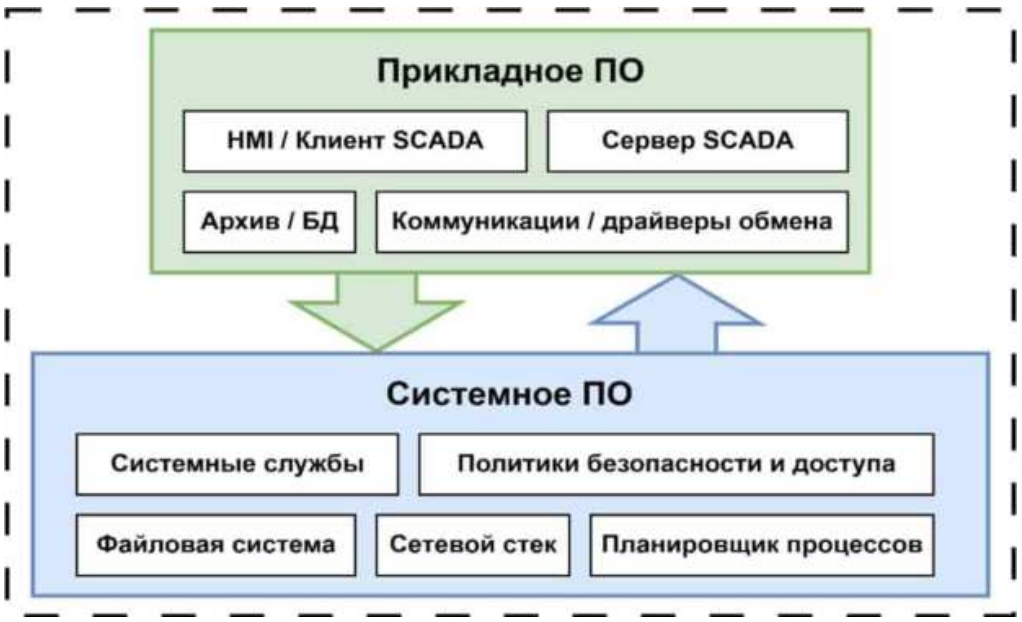

Рисунок 1 – Программная среда диспетчерского уровня АСУТП

На диспетчерском уровне программная среда формируется совокупностью системного и прикладного ПО [13], характеристики которых определяют условия выполнения функций SCADA-систем и характер их взаимодействия [9, 14].

К системным параметрам среды относятся характеристики операционной системы (ОС), настройки распределения вычислительных ресурсов, файловых систем и подсистем хранения данных, а также параметры сетевого взаимодействия, включая IP-адресацию, маршрутизацию, используемые протоколы и политики безопасности [28].

Конфигурация прикладной части включает состав и версии компонентов SCADA-систем, настройки серверных и клиентских модулей, средства архивирования, коммуникационные драйверы. В составе конфигурации также входит проектная часть, определяемая структурой технологических тегов, алгоритмами их обработки и визуализации [28].

Требования к программному и информационному обеспечению, а также к функциям системы формулируются в техническом задании (ТЗ) [13] и проектной документации (ПД).

Подтверждение соответствия реализованной системы установленным требованиям и проектным решениям выполняется в рамках испытаний в соответствии с ГОСТ Р 59792-2021 и оформляется протоколами испытаний.

Совокупность сведений, зафиксированных в утвержденной проектной и эксплуатационной документации и подтвержденных результатами испытаний, в работе рассматривается как нормативное конфигурационное описание программной среды, принятое в качестве исходного при промышленной эксплуатации. Однако оно может актуализироваться при регламентированных изменениях.

1.2 Эксплуатационное сопровождение программной среды

С момента ввода диспетчерской подсистемы АСУТП в промышленную эксплуатацию наряду с ее использованием по функциональному назначению,

осуществляется комплекс регламентных и внеплановых работ по ее сопровождению, направленных на поддержание корректного функционирования системы [13].

Эксплуатационное сопровождение представляет собой совокупность организационно-технических мероприятий, направленных на поддержание работоспособности [29] и эксплуатационной целостности [30] программной среды диспетчерского уровня АСУТП [13]. Оно ориентировано преимущественно на экспертный контроль фактического состояния программной среды, оценка которого формируется на основе анализа совокупности функциональных и эксплуатационных признаков, которые проявляются в процессе ее функционирования.

К основным диагностическим признакам относятся доступность программных модулей, работоспособность каналов информационного взаимодействия, а также полнота и достоверность реализации информационных и управляющих функций системы в текущих условиях эксплуатации [13].

Обеспечение эксплуатационной целостности основывается на поддержании конфигурации программной среды в соответствии с верифицированным нормативным конфигурационным описанием [10, 13], подтверждение которого в условиях эксплуатации требует выполнения специализированных процедур анализа и проверки [11-13].

Длительная эксплуатация системы сопряжена с потенциальным влиянием дестабилизирующих факторов, связанных с аппаратными отказами, внешними воздействиями и неявными дефектами ПО [11, 13]. Под их воздействием возникают риски постепенного отклонения среды от нормативной конфигурации (накопление незарегистрированных изменений, рассогласование компонентов). Возможен риск появления дискретных критических событий [14, 31], к которым относятся установка некорректных обновлений, ошибки интеграции новых модулей или несанкционированное изменение параметров системы [31].

Главной особенностью таких воздействий является отложенный характер проявления последствий, когда конфигурационные расхождения долгое время

остаются скрытыми и обнаруживаются лишь в моменты критических нарушений. Это существенно затрудняет своевременную диагностику, увеличивая сложность и длительность восстановительных процедур [14]. В ситуациях, когда по наблюдаемым функциональным и эксплуатационным признакам явные отклонения не выявляются, но при этом целостность информационной модели диспетчерской подсистемы нарушена [14], возникает необходимость выполнения процедуры углубленного конфигурационного аудита.

Данный анализ предполагает идентификацию фактической конфигурации программной среды и ее поэтапную верификацию на соответствие нормативному конфигурационному описанию. Согласно нему проверяются следующие элементы:

– состав и параметры прикладного и системного ПО (версии программных компонентов, наличие обновлений, целостность исполняемых файлов);

– конфигурация сетевого взаимодействия (права доступа, настройки протоколов обмена, политики безопасности);

– доступность и корректность функционирования системных служб ОС;

– наличие аномалий использования вычислительных ресурсов, влияющих на функционирование программных компонентов.

1.3 Влияние архитектуры исполнения SCADA-систем на организацию процедур сопровождения программной среды

1.3.1 SCADA как программный комплекс с целостным исполнением

Существенное влияние на трудоемкость и структуру процедур сопровождения программной среды диспетчерского уровня АСУТП оказывает архитектура SCADA-системы, являющейся функциональным ядром прикладной части данной программной среды [38, 39].

Во многих существующих диспетчерских системах АСУТП SCADA реализуется на основе единого программного комплекса. В этом случае

компоненты, отвечающие за визуализацию, обмен данными, обработку информации и взаимодействие с оборудованием, объединяются в одну исполняемую структуру, которая устанавливается и поддерживается как единое целое [40]. В дальнейшем такая организация программной среды будет называться монолитной архитектурой.

Классическими ее примерами выступают промышленные SCADA-системы предыдущих поколений, такие как Siemens WinCC (версии до перехода к платформенным решениям – WinCC v6, v7) и MasterSCADA 3.x, где все основные функции интегрированы в единую программную структуру (рисунок 2) и сопровождаются как неделимое целое.

Монолитная SCADA-система может переноситься между вычислительными узлами, при этом любые операции сопровождения – установка, обновление, восстановление и проверка работоспособности – всегда выполняются для всего комплекса целиком [41]. Архитектура таких систем не предусматривает изолированного сопровождения или развития отдельных функций, поэтому изменения, как правило, внедряются разработчиком только через обновление всего программного комплекса.

Рисунок 2 – Структура монолитной архитектуры программной среды

В результате программное обеспечение представлено как единый артефакт. Контроль его работоспособности возможен либо с помощью метрик, предоставляемых непосредственно разработчиком ПО, либо за счет внедрения

специализированных систем мониторинга и телеметрии, что требует затраты дополнительных ресурсов [42].

Сопровождение среды исполнения монолитной SCADA-системы заключается в обеспечении строгой детерминированности программной платформы. В силу неделимости архитектурных компонентов, критически важным становится поддержание неизменности программного окружения – версий ОС, системных библиотек и драйверов. Любое неконтролируемое вмешательство в среду – от установки стороннего ПО до обновления второстепенных компонентов – несет риск непредсказуемого изменения поведения всей системы или нарушения ее функциональной связности [13, 14].

При возникновении критических сбоев, не выявляемых штатными средствами диагностики, восстановление диспетчерского комплекса часто исключает возможность точечного исправления и требует полной репликации системы из контрольной точки.

1.3.2 SCADA как совокупность исполняемых компонентов

Помимо монолитной архитектуры, где все функции реализованы в едином исполняемом модуле, SCADA-система может быть организована как совокупность отдельных компонентов [43]. Каждый из них выполняет строго определенную задачу и функционирует независимо от остальных [44]. Такая архитектура наглядно демонстрируется на схеме рисунка 3, где компоненты представлены как автономные блоки, взаимодействующие друг с другом через стандартизированные интерфейсы и каналы связи [45, 54].

В компонентной архитектуре к самостоятельным элементам относятся интерфейсы визуализации, модули обработки телеметрии, средства регистрации данных, а также компоненты для интеграции с внешними источниками информации [46]. Каждый блок устанавливается как отдельная исполняемая единица, позволяя выполнять обновление, диагностику и обслуживание без

остановки всей системы [47]. Визуализация подобной архитектуры подчеркивает раздельность элементов и возможность независимой работы каждого из них [48].

Рисунок 3 – Структура компонентной архитектуры

Преимущество компонентной организации заключается в локализации процедур сопровождения и управления на уровне отдельных исполняемых компонентов [49]. Отклонения и сбои выявляются и устраняются на уровне отдельных элементов, не влияя на работоспособность остальной программной среды [50]. Модульность архитектуры обеспечивает возможность независимого отслеживания фактической конфигурации каждого компонента [18, 51]. Задачи сопровождения, а также масштабирование, резервирование и включение новых функций могут осуществляться поэтапно, без воздействия на функционирующие элементы системы.

Сопровождение компонентной архитектуры приводит к усложнению задачи по синхронизации версий программных модулей, контролю совместимости и поддержанию точного конфигурационного состава диспетчерской системы [54]. Критически важным становится аудит параметров взаимодействия компонентов друг с другом, поскольку нарушение их согласованности может приводить к деградации функционирования диспетчерской системы и нарушению целостности информационной модели [13, 14].

1.4 Ограничения существующих процедур сопровождения программной среды

Рассмотренные в разделе 1.2 процедуры сопровождения программной среды диспетчерского уровня АСУТП ориентированы преимущественно на подтверждение корректности функционирования системы по наблюдаемым функциональным и эксплуатационным признакам.

Оценка состояния программной среды выполняется на основе экспертного анализа:

1. работоспособности компонентов прикладного уровня диспетчерской системы;

2. наличия и корректности информационного обмена между диспетчерским и полевым уровнями;

3. корректности реализации заданных функций диспетчерской системы в части настройки информационных и управляющих связей между элементами человеко-машинного интерфейса и технологическим оборудованием, определенных проектной и нормативной документацией.

Программная среда диспетчерского уровня, как показано в разделе 1.1, представляет собой сложный конфигурационный объект, включающий совокупность системных и прикладных компонентов, а также их параметров и взаимосвязей, формально заданных в нормативном конфигурационном описании. Соответствие программной среды указанному описанию в условиях промышленной эксплуатации не может быть однозначно подтверждено исключительно по результатам функционального наблюдения, поскольку такие наблюдения не обеспечивают полноты и формальной воспроизводимости конфигурационного анализа.

Экспертный характер существующих процедур сопровождения обуславливает отсутствие формализованного механизма сопоставления фактической конфигурации программной среды с ее нормативным описанием. Заключение о корректности состояния системы формируется на основе

интерпретации косвенных диагностических признаков и в значительной степени зависит от квалификации, опыта и субъективных оценок обслуживающего персонала. В результате одни и те же конфигурационные отклонения могут по-разному интерпретироваться либо оставаться невыявленными при отсутствии явных функциональных нарушений.

Указанная особенность иллюстрируется на рисунке 4, где показано, что при отсутствии формализованного критерия сопоставления соответствие фактической конфигурации нормативному описанию подменяется оценкой наблюдаемого функционирования системы.

Рисунок 4 – Проблема сопоставления нормативной и фактической конфигураций программной среды при экспертно-ориентированном сопровождении

Ввиду экспертно-ориентированного характера процедур сопровождения и отсутствия формализованных средств конфигурационного анализа усложнение архитектуры современных программных сред приводит к увеличению трудоемкости процедур сопровождения. Увеличение числа программных

компонентов, их версий, параметров и связей, а также усложнение проектной части SCADA-системы приводят к существенному росту объема проверяемых элементов.

В этих условиях выполнение углубленного конфигурационного анализа в задачах сопровождения требует значительных временных затрат и, как правило, носит нерегулярный характер.

Существенным ограничением существующих процедур сопровождения является их преимущественно реактивная организация. Инициация детального анализа конфигурационного состояния программной среды, как правило, осуществляется после возникновения функциональных нарушений либо при выявлении явных отклонений в работе системы. При этом конфигурационные расхождения, не имеющие немедленных эксплуатационных проявлений и не выявляемые в рамках текущих процедур функционального и эксплуатационного контроля, могут накапливаться в течение длительного времени и обнаруживаться лишь на поздних стадиях эксплуатации либо в момент критического отказа.

Конфигурационное расхождение в части связности компонентов является критичным прежде всего для компонентных архитектур SCADA-систем, рассмотренных в разделе 1.3.2, где нарушение согласованности параметров взаимодействия между автономными модулями может не приводить к немедленному отказу отдельных компонентов, но при этом нарушать целостность информационной модели диспетчерского уровня.

Для обобщения выявленных ограничений и анализа их влияния на операции сопровождения программной среды итоги рассмотрения сведены в таблицу 1.

Таблица 1. Классификация ограничений процедур сопровождения программной среды

Группа ограничений	**Причина**	**Следствие**
Организационные	Экспертная ориентированность процедур и зависимость от квалификации обслуживающего персонала	Невозможность формирования формализованного, однозначного и воспроизводимого представления фактического конфигурационного состояния программной среды

Продолжение таблицы 1

Группа ограничений	Причина	Следствие
Методические	Ориентация контроля на наблюдаемые и интерпретируемые обслуживающим персоналом функциональные и эксплуатационные признаки	Возможность длительного сохранения скрытых конфигурационных расхождений без своевременного выявления по функциональным признакам
Ресурсные	Рост числа программных компонентов и объема контролируемых параметров	Ограничение полноты и регулярности выполнения процедур конфигурационного контроля
Временные	Преимущественно реактивная инициация процедур анализа	Позднее выявление конфигурационного дрейфа и усложнение восстановительных процедур

В таблице представлены группы ограничений, причины их возникновения, а также результаты их влияния на эксплуатацию.

1.5 Анализ научной проработанности задач сопровождения программной среды

Анализ научных исследований в области АСУТП и их программно-технического обеспечения показывает, что в ныне существующих работах основной акцент делается на развитии функциональности прикладного программного обеспечения, обработке технологических данных и алгоритмах управления процессами. Программная среда включается в рассмотрение преимущественно как средство обеспечения функционирования прикладных подсистем, а следовательно – задачи ее эксплуатационного сопровождения и конфигурационной целостности при этом не получают достаточной научной проработки.

В рамках данного направления исследований прорабатываются вопросы изменения архитектуры и состава программных компонентов автоматизированных систем управления (АСУ), включая модернизацию и развитие функциональных подсистем. Эти положения развиты, в частности, в диссертационной работе Абдалова А.В. [15], где рассматриваются методы автоматизации модернизации и

средства поддержки принятия решений при плановых преобразованиях программной структуры. Указанные результаты формируют методологическую и архитектурную основу анализа программных систем, однако не ориентированы на формализованный контроль соответствия фактической конфигурации программной среды нормативному конфигурационному описанию в условиях эксплуатации.

В то же время в смежной области информационных технологий вопросы сопровождения программной среды получили более глубокую проработку. Рост сложности информационных систем послужил предпосылкой развития методов, ориентированных на контроль и верификацию конфигурации как неотъемлемых элементов архитектуры программных комплексов [18]. Одним из результатов такого развития стала методология DevOps [16, 17], направленная на согласование процессов разработки, внедрения и эксплуатации ПО.

С опорой на данную методологию в ряде отечественных исследований рассматриваются вопросы организации рабочих окружений и управления их параметрами в многозадачных и распределенных вычислительных системах. В работах [19-23] предложены алгоритмы, обеспечивающие воспроизводимость условий функционирования программных компонентов и управление параметрами вычислительной среды, распределение вычислительных ресурсов при динамическом масштабировании инфраструктуры.

Результаты указанных исследований демонстрируют высокий уровень проработанности методов сопровождения программных сред в области информационных технологий (ИТ). При этом данные методы разрабатывались для условий эксплуатации, не предъявляющих требований к детерминированности выполнения операций и неизменности конфигурации в процессе функционирования системы.

В условиях эксплуатации программной среды диспетчерского уровня АСУТП, где нарушение конфигурационной целостности непосредственно влияет на корректность информационных процессов диспетчерского управления, задачи формализованного контроля и подтверждения соответствия фактической

конфигурации нормативному конфигурационному описанию не получили систематического решения в рамках существующих исследований.

1.6 Постановка задачи управления конфигурацией программной среды

На основании анализа архитектуры программной среды диспетчерского уровня АСУТП, процедур ее эксплуатационного сопровождения, выявленных ограничений и уровня научной проработанности задач формализованного контроля конфигурационного состояния, в настоящей работе формулируется задача управления конфигурацией программной среды диспетчерского уровня АСУТП.

Объектом управления в работе является конфигурация программной среды, рассматриваемая как совокупность параметров системного и прикладного ПО, а также характеристик их взаимосвязей, определяющих условия функционирования диспетчерской системы в процессе эксплуатации.

Управление реализуется на основе формализованного представления фактического конфигурационного состояния, формируемого в процессе эксплуатации, и его сопоставления с нормативным конфигурационным описанием.

Результатом такого сопоставления является идентификация отклонений и формирование проекта корректирующих воздействий, направленных на восстановление конфигурационного соответствия.

Однако в условиях промышленной эксплуатации АСУТП применение корректирующих воздействий в отношении конфигурации программной среды не всегда может быть выполнено автоматически и требует учета текущего технологического режима.

В связи с этим в предлагаемой постановке задачи функции диагностики конфигурационного состояния, выявления отклонений и формирования корректирующих воздействий делегируются автоматизированной системе сопровождения, тогда как применение сформированных корректирующих воздействий осуществляется ответственным эксплуатационным персоналом в

пределах допустимых технологических режимов и в отведенные регламентом эксплуатационные окна.

Схема процесса управления конфигурацией программной среды диспетчерского уровня АСУТП представлена на рисунке 5.

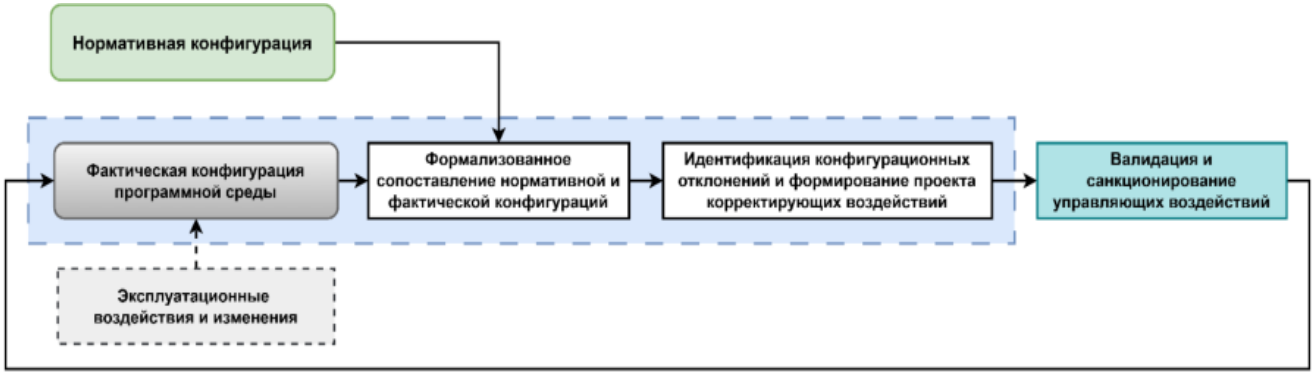

Рисунок 5 – Автоматизированный контур управления конфигурацией программной среды диспетчерского уровня АСУТП

На схеме отражена взаимосвязь нормативной и фактической конфигураций, а также этапы формирования корректирующих воздействий.

1.7 Концепция и архитектура контура сопровождения

Реализация задачи управления конфигурацией программной среды диспетчерского уровня АСУТП требует архитектурного оформления автоматизированной части контура управления в виде самостоятельного функционального элемента, интегрированного в диспетчерский контур, но не входящего в состав исполняемых компонентов SCADA-системы.

В настоящей работе данный элемент рассматривается как система сопровождения программной среды, архитектурно обособленная от компонентов диспетчерской системы и предназначенная для реализации автоматизированной части данного контура управления.

Положение системы сопровождения относительно обслуживаемой программной среды и связь с основными компонентами диспетчерского контура АСУТП представлены на рисунке 6.

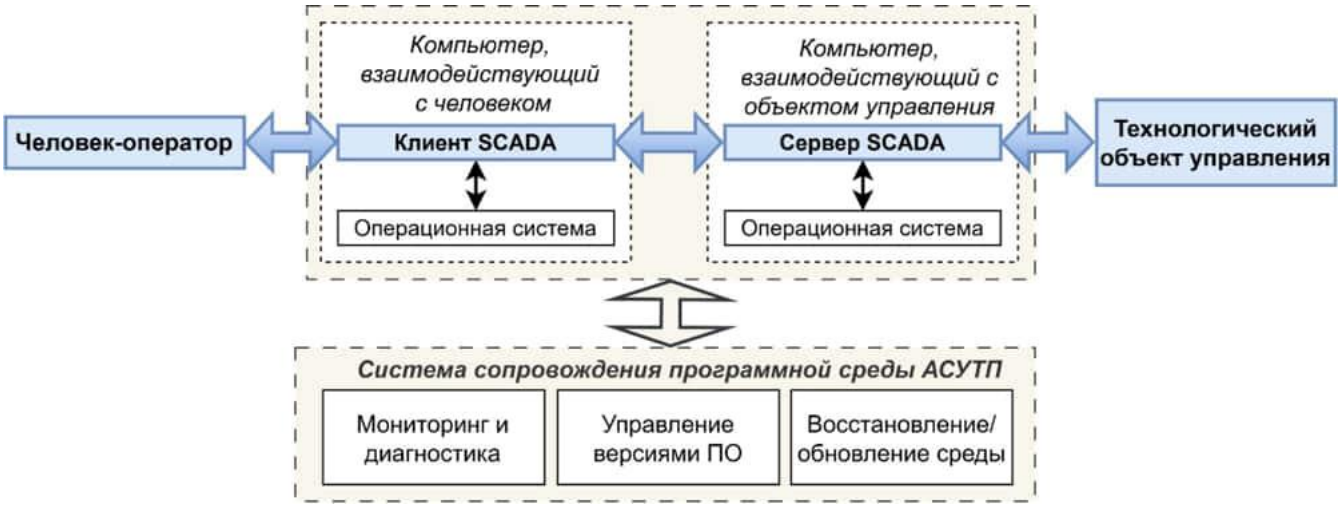

Рисунок 6 – Функциональная структура программной среды АСУТП с выделением системы сопровождения

На функциональной структуре показано, что система сопровождения функционально и логически обособлена от компонентов диспетчерской системы и не входит в состав исполняемых модулей SCADA. Однако она взаимодействует с обслуживаемой программной средой на уровне получения диагностической информации и контроля параметров программной платформы.

В рамках предложенной концепции рассмотрены и проанализированы три основных архитектурных варианта размещения системы сопровождения относительно обслуживаемой программной среды [32].

1.7.1 Замкнутая архитектура

В качестве базового варианта в данной работе рассматривается замкнутая архитектура. Под ней понимается конфигурация, при которой система сопровождения и программная среда АСУТП размещаются на одном вычислительном узле (рисунок 7). Для такой архитектуры характерна полная

интеграция сопровождающего слоя с обслуживаемой программной средой, а выполнение всех операций сопровождения осуществляется с использованием ресурсов общего вычислительного узла.

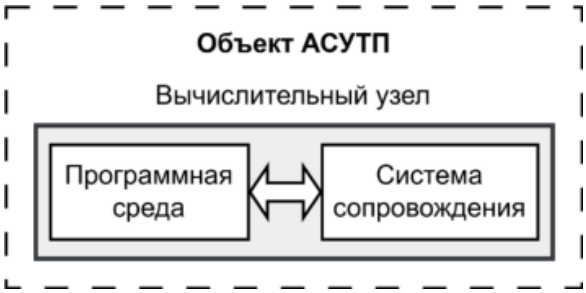

Рисунок 7 – Замкнутая архитектура сопровождения

При такой организации критически важные функции могут быть реализованы лишь при условии, что поддержание работоспособности и регламентированных процедур восстановления обеспечиваются средствами самого вычислительного узла. Автоматизация операций сопровождения возможна на уровне локальных процедур обновления, резервного копирования и инициализации, формализованных в виде сценариев или шаблонов. Однако автоматизированные функции сопровождения сохраняют работоспособность только при исправном состоянии данного вычислительного узла. В случае аппаратного сбоя или критических нарушений программной среды исполнения (например, отказ ОС или разрушение инфраструктуры узла) автоматизированное восстановление компонентов становится невозможным, поскольку система сопровождения замыкается внутри одного вычислительного узла. В подобных обстоятельствах восстановление возможно исключительно вручную – с использованием локальных резервных копий и заранее подготовленных сценариев.

Представленный вариант размещения актуален для небольших систем, а также для выделенных сегментов крупных производственных объектов при отсутствии потребности во внедрении распределенного управления.

В таких условиях замкнутая архитектура системы сопровождения является достаточной для локальных и автономных диспетчерских систем с ограниченным

масштабом [12], обеспечивая требуемый уровень управления конфигурацией при минимальной сложности реализации.

1.7.2 Архитектура с внутренним контуром сопровождения

При рассмотрении интеграции системы сопровождения в более крупные и распределенные диспетчерские системы выявляются ограничения замкнутой архитектуры, связанные с утратой возможности выполнения процедур сопровождения при отказах вычислительного узла.

Для устранения указанных ограничений целесообразно использовать размещение системы сопровождения вне исполняемых компонентов программной среды, но внутри объекта автоматизации.

Такой вариант архитектурного исполнения в настоящей работе определяется как архитектура с внутренним контуром сопровождения.

В рамках объекта АСУТП система сопровождения может быть размещена либо на отдельном вычислительном узле в пределах сегмента локальной сети, либо на том же физическом ресурсе при обеспечении его изоляции от диспетчерской программной среды (рисунок 8).

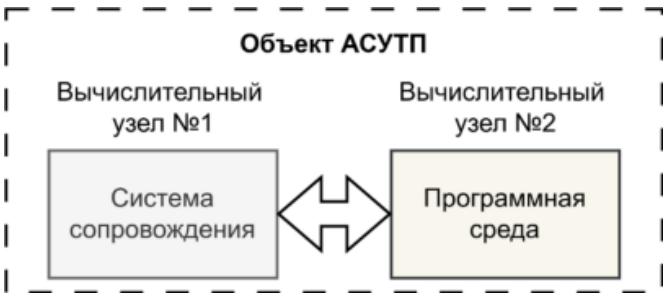

Рисунок 8 – Архитектура с внутренним контуром сопровождения

Следует учитывать, что при выделении системы сопровождения за пределы среды исполнения прикладного ПО АСУТП сохраняется риск утраты доступа к ней в случае отказа всей инфраструктурной площадки. В такой ситуации выполнение процедур сопровождения и восстановления затрудняется, поскольку вместе с

основной инфраструктурой может быть утрачен доступ к резервным системам и каналам связи.

Для повышения отказоустойчивости узла рекомендуется предусматривать предусмотреть резервные системы обеспечения, включая автономные источники электропитания [33] и независимые каналы связи [34]. Автономность системы сопровождения по отношению к диспетчерской программной среде позволяет поддерживать его работоспособность и выполнение критически важных функций независимо от состояния основной инфраструктуры.

Данная архитектура ориентирована на контроль конфигурации программных компонентов диспетчерской подсистемы в пределах одного объекта и обеспечивает централизованное сопровождение без привлечения внешних систем. Реализация данной архитектуры наиболее эффективна для локальных сетей и изолированных сегментов АСУТП [12], однако предъявляет повышенные требования к отказоустойчивости инфраструктуры и наличию резервных копий, используемых при восстановлении после отказов.

1.7.3 Архитектура с вынесенным контуром сопровождения

Для минимизации рисков, связанных с хранением и обработкой всех данных исключительно внутри объекта автоматизации, система сопровождения размещается вне пределов объекта АСУТП. Такой способ организации означает, что контур сопровождения вынесен за периметр объекта и размещается на удаленных ресурсах, например, в центре обработки данных (ЦОД) [35]. Это обеспечивает возможность централизованной организации процедур сопровождения независимо от территориального расположения объекта. Архитектурная организация представлена на рисунке 9.

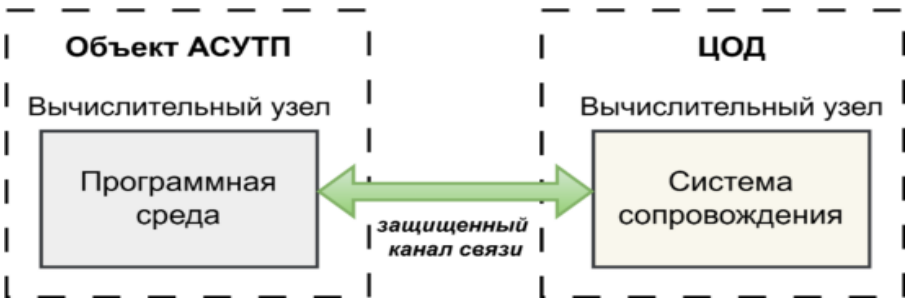

Рисунок 9 – Архитектура с вынесенным контуром сопровождения

Такая архитектура обеспечивает независимость механизмов сопровождения от состояния диспетчерской программной среды внутри объекта. В случае нарушения работоспособности сохраняется возможность выполнения процедур восстановления и повторного ввода программной среды в эксплуатацию без физического присутствия на площадке. Операции сопровождения выполняются извне по заранее определенным каналам связи и возможны только при наличии устойчивого взаимодействия между системой сопровождения и программной средой объекта АСУТП.

Возложение функций сопровождения программной среды на внешнюю специализированную систему устраняет необходимость поддержки соответствующих механизмов непосредственно внутри объекта АСУТП.

Архитектура с вынесенным контуром позволяет централизованно организовать сопровождение программных сред АСУТП на нескольких объектах одновременно. Это обеспечивает своевременное выявление отклонений от заданных параметров, поддержание единых версий ПО и интеграцию процедур сопровождения с корпоративными системами мониторинга, аудита и учета. Данный вариант особенно актуален для распределенных организаций [36], эксплуатирующих несколько технологических площадок.

Основным ограничением применения вынесенного контура сопровождения является невозможность его использования на объектах с повышенными требованиями к изоляции. При наличии в системе конфиденциальных данных размещение контура сопровождения за пределами периметра объекта автоматизации АСУТП может увеличить риск компрометации чувствительной

информации [12, 31]. В таких случаях целесообразно использовать архитектуры с замкнутым или внутренним контуром либо предусматривать дополнительные меры защиты, включая шифрование каналов связи, механизмы аутентификации [31, 37] и регламентирование доступа внешних систем сопровождения к программной среде объекта АСУТП [12].Оптимальный выбор архитектуры размещения системы сопровождения определяется балансом между требованиями к безопасности, степенью централизованной организации сопровождения и особенностями конкретных объектов автоматизации. Таблица 2 содержит сравнительный анализ характеристик, ограничений и областей применения возможных архитектур размещения слоя сопровождения.

Таблица 2 – Сравнение архитектур реализации контура среды сопровождения

Параметр	**Замкнутая архитектура**	**Внутренний контур сопровождения**	**Вынесенный контур сопровождения**
Зависимость от среды исполнения	Высокая	Средняя	Низкая
Зависимость от каналов связи	Отсутствует	В пределах локальной сети	Высокая
Безопасность и изоляция	Высокая	Средняя, зависит от мер защиты локальной инфраструктуры	Высокая, требуются дополнительные меры безопасности
Сфера применения	Малые объекты, объекты с жесткими требованиями к изоляции	Средние и крупные объекты с локальной инфраструктурой	Распределенные объекты с централизованным управлением

Согласно проведенному сравнительному анализу ни одна из рассмотренных архитектур не является универсальной для всех случаев применения. Выбор конкретной схемы определяется спецификой объекта автоматизации и предъявляемыми требованиями к сопровождению программной среды АСУТП.

Выводы по 1 главе

Установлено, что программная среда диспетчерского уровня АСУТП представляет собой сложный интеграционный объект, корректность функционирования которого определяется согласованностью программных компонентов и параметров их конфигурации, формирующих целостную информационную модель технологического процесса.

Показано, что текущее эксплуатационное сопровождение программной среды в условиях промышленной эксплуатации ориентировано преимущественно на экспертную оценку функционального состояния системы и не обеспечивает формализованного и воспроизводимого подтверждения соответствия фактической конфигурации нормативному описанию.

Выявлено, что существующие процедуры сопровождения характеризуются рядом организационных, методических и временных ограничений, обусловленных их преимущественно экспертно-ориентированным и реактивным характером, что затрудняет своевременное выявление конфигурационных отклонений.

Показано, что задачи формализованного контроля и управления конфигурацией программной среды диспетчерского уровня АСУТП в условиях эксплуатации не получили систематической научной проработки.

На основании проведенного анализа сформулирована задача управления конфигурацией программной среды диспетчерского уровня АСУТП и обоснована целесообразность ее реализации в виде специализированного слоя сопровождения.

Предложена классификация архитектурных вариантов организации слоя сопровождения, определяющая область их рационального применения.

ГЛАВА 2. МЕТОДОЛОГИЧЕСКИЕ ОСНОВЫ ОРГАНИЗАЦИИ УПРАВЛЕНИЯ И АВТОМАТИЗАЦИИ ЗАДАЧ СОПРОВОЖДЕНИЯ ПРОГРАММНОЙ СРЕДЫ АСУТП

2.1 Формализация конфигурации программной среды АСУТП

2.1.1 Модель фактической конфигурации программной среды

Для задач автоматизированного контроля конфигурации программной среды диспетчерского уровня АСУТП требуется формализованное представление реализованного конфигурационного состава, допускающее его сопоставление с нормативным конфигурационным описанием.

Такое сопоставление возможно только при использовании единого формального представления; в противном случае автоматизированная оценка соответствия оказывается неполной либо некорректной.

Конфигурационные параметры программной среды в процессе ее функционирования на вычислительной платформе не представлены в явном виде, непосредственно пригодном для формального анализа [26]. В связи с этим в рамках настоящей работы вводится процедура воспроизведения конфигурационного состояния, обеспечивающая формирование формализованного представления конфигурации программной среды на основе эксплуатационно доступных данных [26].

Оперируя представлением состава программной среды и ее нормативного описания, фактическая конфигурация должна отражать проекцию тех же конфигурационных параметров и установок [24, 26], значения которых активны в текущий момент.

Результатом выполнения указанной процедуры должно являться формализованное представление фактической конфигурации программной среды [24] в момент времени t, обозначаемое как $S(t)$ и задаваемое в виде множества параметров (2.1).

$$S(t) = \{s_1(t), s_2(t), \dots, s_n(t)\}, \quad (2.1)$$

где $s_i(t)$ – параметр формализованного представления, формируемый на основе эксплуатационно доступных данных исполняемой программной среды.

На рисунке 10 показано формирование формализованного представления фактической конфигурации $S(t)$ как отображения параметров программной среды в область конфигурационного контроля.

Рисунок 10 – Формирование представления фактической конфигурации программной среды АСУТП

В дальнейшем представление, заданное выражением (2.1), рассматривается как модель фактической конфигурации программной среды в момент времени t, используемая в задачах анализа, сопоставления и управления конфигурацией.

Компоненты программной среды, не задействованные в текущем режиме функционирования либо не оказывающие влияния на контролируемое состояние программной среды, могут не включаться в модель $S(t)$, если иное не требуется целями конфигурационного контроля и сопоставления.

Параметры модели (2.1) $s_i(t)$ рассматриваются как структурированные объекты конфигурации, отражающие состав программных компонентов, их иерархию, привязки и ограничения. Допускается представление параметров как в виде элементарных значений, так и в виде составных описаний с рекурсивной

вложенностью. Рекурсивное описание позволяет формализовать древовидные и иерархические зависимости программных компонентов. В результате взаимосвязи между элементами программной среды учитываются при сопоставлении и преобразовании параметров конфигурации.

В общем случае параметр $s_i(t)$ может быть представлен в виде структурированного объекта (кортежа), задаваемого выражением (2.2).

$$s_i(t) = \langle id_i, type_i, V_i(t), D_i(t) \rangle, \quad (2.2)$$

где:

– id_i – идентификатор объекта конфигурации (сервер, сервис, проект и т.д.);

– $type_i$ – тип объекта (SCADA-сервер, сервис, библиотека, вычислительный узел и др.);

– $V_i(t)$ – совокупность собственных параметров объекта в момент времени t (версия, путь, порт, режим функционирования и др.);

– $D_i(t)$ – совокупность вложенных структур и зависимых элементов, каждый из которых может быть представлен в аналогичном виде.

2.1.2 Критерий полноты формализованного представления фактической конфигурации

Критерием полноты модели $S(t)$ является возможность ее использования для выполнения всех предусмотренных процедур автоматизированного сопоставления с нормативным описанием в заданной области конфигурационного контроля.

При этом модель считается полной, если:

1. включает все программные компоненты и параметры, наличие и значения которых подлежат контролю в соответствии с нормативным конфигурационным описанием;

2. допускает однозначное сопоставление элементов нормативного конфигурационного описания с соответствующими элементами фактической конфигурации;

3. допускает формальное задание структуры конфигурации и интерпретацию параметров в терминах требований и ограничений нормативного конфигурационного описания;

4. не требует привлечения дополнительной экспертной интерпретации при выполнении процедур контроля соответствия.

Полнота формализованного представления определяется относительно нормативного конфигурационного описания и области конфигурационного контроля и не предполагает исчерпывающего описания всех частей программной среды, несущественных для решаемых задач.

2.1.3 Критерий достоверности результата воспроизведения фактической конфигурации

Достоверность модели $S(t)$ определяется результатом автоматизированной проверки, выполняемой после завершения процедуры воспроизведения конфигурационного состояния.

Результат воспроизведения считается достоверным, если одновременно выполняются следующие условия:

– каждому элементу формализованного представления соответствует однозначно идентифицируемый программный компонент или сервис, наличие и состояние которого подтверждаются эксплуатационно доступными признаками функционирования;

– значения параметров получены из заранее определенных источников данных и интерпретированы в соответствии с установленными правилами формирования формализованного представления;

– структура представления не содержит внутренних противоречий и удовлетворяет заданным инвариантам целостности и ограничениям допустимых значений параметров;

– повторное выполнение процедуры воспроизведения в сопоставимых эксплуатационных условиях приводит к эквивалентному по структуре и значениям контролируемых параметров представлению конфигурационного состояния.

Результаты воспроизведения фактической конфигурации, не удовлетворяющие приведенным условиям, признаются недостоверными и исключаются из дальнейших процедур анализа и управления конфигурацией программной среды.

Только результаты воспроизведения, признанные достоверными в соответствии с данным критерием, используются в последующих процедурах анализа и управления конфигурацией программной среды.

Таким образом, на основе модели $S(t)$ могут быть реализованы методы управления ее конфигурационным состоянием в процессе эксплуатации.

2.2 Методы управления конфигурацией программной среды АСУТП

Требуемое конфигурационное состояние, соответствующее нормативному конфигурационному описанию, далее обозначается как $S_{\text{цел}}$. Формализованное представление $S_{\text{цел}}$ задается в той же структуре и системе параметров, что и представление фактической конфигурации $S(t)$ [25], описанной выражением (2.1).

2.2.1 Императивный метод выполнения операций сопровождения

На исполнительном уровне управление конфигурацией осуществляется путем выполнения регламентированного набора операций сопровождения, соответствующего установленному порядку преобразования конфигурации программной среды [55-57]. Данная последовательность формализуется в виде набора операций A_1, A_2, …, A_n, где каждая операция A_i представляет собой регламентированное действие, изменяющее параметры конфигурации

программной среды в соответствии с установленным порядком сопровождения [55-57].

К таким операциям относятся установка/обновление программных компонентов, изменение параметров прикладного диспетчерского проекта, процедуры резервного копирования, перезапуск сервисов, удаление устаревших элементов, а также верификация конфигурации после внесения изменений. Указанные действия определяют последовательность шагов и ориентированы на задание порядка выполнения действий, что соответствует императивному принципу управления [55-57].

Результат применения очередной операции A_i к текущей конфигурации обозначается как $S(t+i)$. Соответственно, последовательность преобразований конфигурации может быть описана следующими соотношениями:

$$S(t+1) = A_1(S(t)) \quad (2.3)$$

$$S(t+2) = A_2(S(t+1)) \quad (2.4)$$

$$S(t+3) = A_3(S(t+2)) \quad (2.5)$$

$$\dots$$

$$S_{\text{цел}} = A_n(S(t+n-1)) \quad (2.6)$$

Данная формализация позволяет однозначно фиксировать результаты выполнения каждой операции сопровождения и воспроизводить полную последовательность изменений конфигурации программной среды.

На базе модели, на рисунке 11 представлен метод последовательного выполнения операций сопровождения, реализующий приведенную формальную модель управления конфигурацией программной среды. Он иллюстрирует преобразование исходной конфигурации $S(t)$ в требуемое конфигурационное состояние $S_{\text{цел}}$ посредством регламентированной цепочки операций.

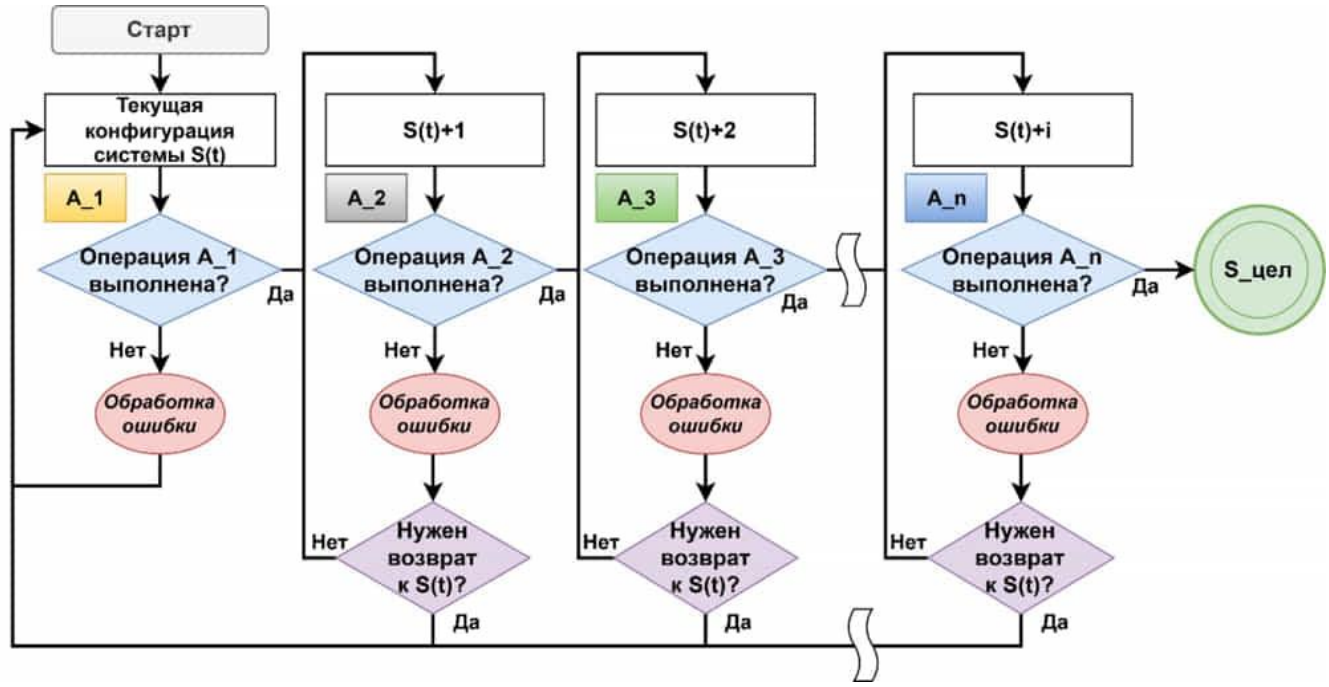

Рисунок 11 – Императивный метод управления

На представленном методе последовательного выполнения операций сопровождения демонстрируется приведение программной среды к нормативной конфигурации посредством выполнения регламентированной цепочки действий. Промежуточные операции, формирующие требуемую конфигурацию $S_{\text{цел}}$ могут быть конкретизированы на абстрактном примере следующим образом:

1. A_1 – установка сервера SCADA;
2. A_2 – настройка сетевого стека и параметров маршрутизации;
3. A_3 – настройка процедур резервного копирования;
4. A_4 – обновление версии сервера опроса технологических данных.

В дальнейшем императивный метод рассматривается в рамках задач исполнения регламентированных операций сопровождения, при которых состав и порядок действий задаются заранее и реализуются без учета динамического анализа текущего состояния программной среды.

Такое представление позволяет зафиксировать исполнительный уровень управления конфигурацией программной среды и использовать его в качестве базовой модели при дальнейшем анализе организации процессов сопровождения.

2.2.2 Декларативный метод формирования корректирующих воздействий

В развитие рассмотренного исполнительного уровня управления, реализуемого в рамках императивной модели, в настоящей работе разработана декларативная модель, основанная на одноименном принципе [58, 59]. В рамках данной модели основным механизмом является автоматизированное сопоставление текущей конфигурации программной среды с заранее заданной нормативной конфигурацией. На основании выявленных расхождений между фактическими и нормативными значениями параметров формируется совокупность корректирующих воздействий, направленных на приведение программной среды к согласованной конфигурации в процессе эксплуатации.

Нормативная конфигурация программной среды представляется в виде множества параметров, однозначно задающих требования к составу и значениям характеристик программной среды, и далее используется в качестве опорного представления $S_{\text{цел}}$.

Механизм формирует управляющие воздействия на основании выявленных отклонений и обеспечивает достижение заранее установленной конфигурации $S_{\text{цел}}$, представленной в виде множества нормативных параметров (2.7).

$$S_{\text{цел}} = \{r_1, r_2, \dots, r_n\}, \tag{2.7}$$

где каждый r_i соответствует установленному значению для определенной характеристики программной среды АСУТП.

Для дальнейшего изложения рассмотрим вектор требуемой конфигурации программной среды $S_{\text{цел}} = \{r_1, r_2, r_3, r_4\}$ как абстрактный набор параметров, каждый из которых соответствует:

- r_1 – версия программного компонента SCADA-сервера;
- r_2 – значение параметра сетевой конфигурации в виде ip-адреса;
- r_3 – путь к хранилищу технологических данных;
- r_4 – установленная политика доступа и параметры безопасности.

В качестве допустимых форматов могут выступать строковые значения, числовые идентификаторы, структурированные записи (например, в формате JSON или XML), либо иные формализованные представления, позволяющие однозначно интерпретировать соответствующую характеристику программной среды АСУТП.

Процесс сопровождения в рамках декларативной модели сводится к автоматическому выявлению отклонений между актуальными и требуемыми параметрами [58, 59], а после – к формированию проекта корректирующих воздействий. Для описания этого процесса определяется вектор отклонений (2.8).

$$e(t) = \delta(S_{\text{цел}}, S(t)) \tag{2.8}$$

Здесь вектор $e(t)$ отражает разницу по каждому параметру между фактическим и требуемым, а δ – оператор покомпонентного сопоставления параметров конфигурации, возвращающий нулевое значение при совпадении параметров и формализованное описание расхождения при их несовпадении.

Для иллюстрации формирования вектора $e(t)$ в таблице 3 приведен абстрактный пример, в котором для некоторого момента времени t зафиксированы конфигурационные значения программной среды АСУТП.

Таблица 3 – Структура вектора отклонений e(t) на абстрактных данных программной среды АСУТП

Параметр	$\boldsymbol{S}_{\text{цел}}$	$\boldsymbol{S(t)}$	$\boldsymbol{e(t)}$
r_1	v1.8.3	v1.7.0	δ(v1.8.3, v1.7.0)
r_2	10.20.28.67	10.20.10.98	δ(10.20.28.67, 10.20.10.98)
r_3	/mnt/data	/opt/archive	δ(/mnt/data, /opt/archive)
r_4	admin:ro	admin:rw	δ(admin:ro, admin:rw)

Таким образом, вектор отклонений $e(t)$ содержит формализованные различия между требуемой и фактической конфигурацией программной среды АСУТП по каждому параметру r_i.

Полученные значения $e(t)$ далее определяют состав и параметры проекта корректирующих управляющих воздействий.

Для их вычисления вводится функция F, зависящая от S_t и $e(t)$, описанная (2.9).

$$\Delta S = F\big(S(t), e(t)\big) \tag{2.9}$$

где ΔS – совокупность корректирующих управляющих воздействий, определяющих требования к изменению параметров программной среды и реализуемых в виде набора операций A_i посредством императивного метода сопровождения.

Относительно приведенного выше примера формирования $e(t)$, функция F формирует совокупность корректирующих управляющих воздействий ΔS, реализация которых на исполнительном уровне осуществляется в виде перечня императивных операций A_i (2.10).

$$\Delta S = \{A_1, A_2, A_3, A_4\}, \tag{2.10}$$

где:

- A_1 – обновление версии программного компонента SCADA-сервера до версии v1.8.3;
- A_2 – переназначение сетевого адреса узла на нормативное значение 10.20.28.67;
- A_3 – переопределение пути к каталогу хранения данных на /mnt/data;
- A_4 – ограничение прав доступа до режима «только чтение (ro)».

В общем случае функция F, оперируя выявленными отклонениями между $S(t)$ и $S_{\text{цел}}$, формирует совокупность регламентированных корректирующих воздействий, определяемых правилами сопровождения и технической документацией. Тем самым обеспечивается однозначная трансляция выявленных конфигурационных расхождений в исполнимый набор императивных операций сопровождения.

Вызов функции F осуществляется при наличии ненулевого вектора отклонений $e(t)$. В случае отсутствия конфигурационных расхождений, то есть при $e(t) = 0$, формирование корректирующих воздействий не требуется.

Таким образом, результат работы функции F представлен в виде формализованного перечня корректирующих воздействий (транзакций),

однозначно сопоставленных выявленным конфигурационным отклонениям и предназначенных для реализации на исполнительном уровне сопровождения.

Переход к $S_{\text{цел}}$ выражается как результат применения проекта корректирующих воздействий ΔS к фактической конфигурации $S(t)$ (2.11).

$$S_{\text{цел}} = \Delta S(S(t)) \tag{2.11}$$

На основе разработанной модели сформирован декларативный метод управления, представленный в виде алгоритма перехода к $S_{\text{цел}}$. Блок-схема метода приведена на рисунке 12.

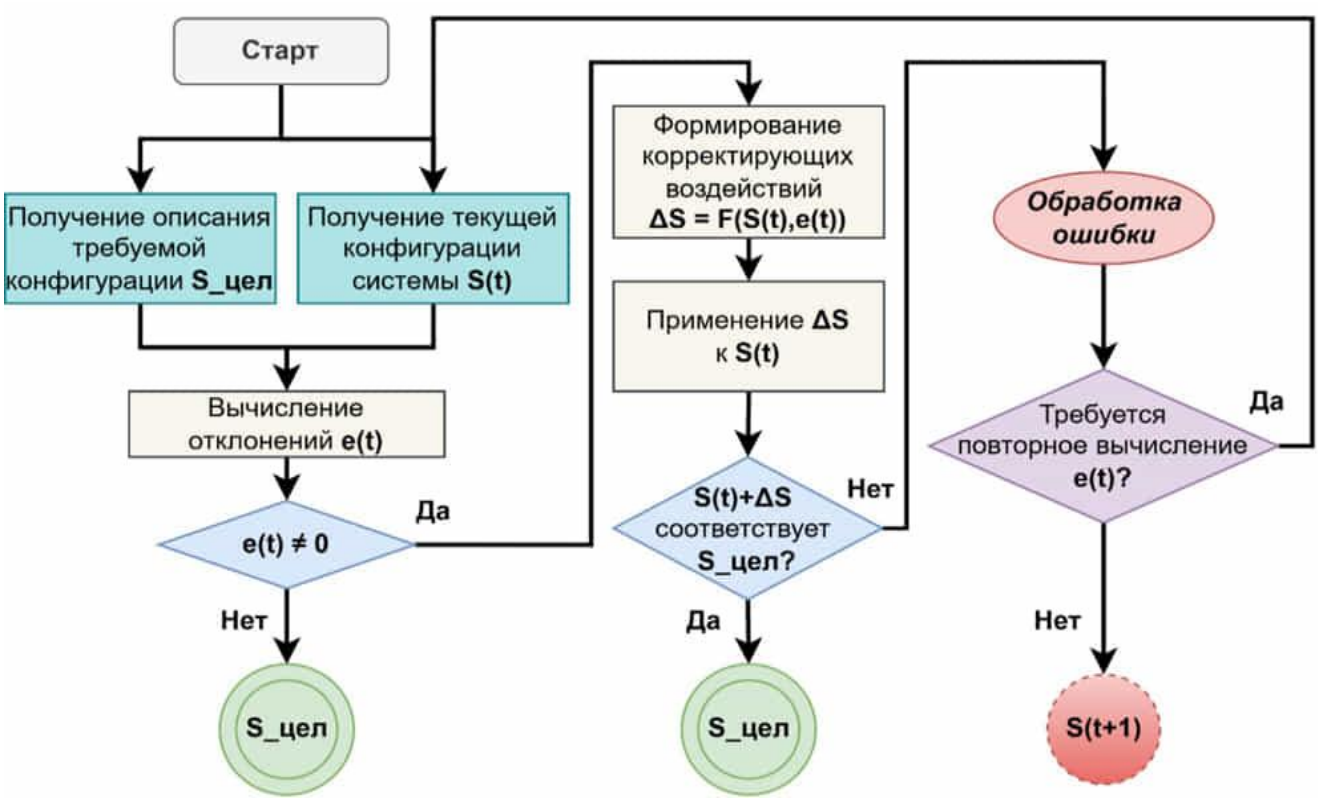

Рисунок 12 – Декларативный метод управления

В результате представленная в работе схема отражает процесс сопровождения программной среды как последовательность формализованных преобразований конфигурации, при которых каждое корректирующее воздействие направлено на минимизацию выявленных отклонений и приведение параметров программной среды к нормативным значениям. Разработанный декларативный метод обеспечивает централизованное сопоставление $S(t)$ и $S_{\text{цел}}$, автоматизацию

корректировки параметров на основании выявленных расхождений и существенное снижение риска накопления несогласованных изменений.

Однако, стоит учитывать, что корректность и однозначность механизма сопоставления отклонений определяются возможностью формализации всех значимых параметров конфигурации программной среды. Для слабо структурированных характеристик или параметров с динамическими взаимосвязями может быть затруднено построение универсального оператора δ и соответствующей функции сопоставления F.

2.3 Стандартизация среды исполнения компонентов программной среды

2.3.1 Изолированное исполнение компонентов и воспроизводимость среды

Формирование $S(t)$ осуществляется в контексте среды исполнения программных компонентов. При размещении компонентов в общем программном окружении параметры этой среды изменяются в ходе регламентных и внеплановых операций сопровождения, выполняемых в интересах отдельных компонентов программной среды. Такие изменения включают обновление библиотек, настройку системных сервисов и корректировку параметров окружения, общих для нескольких компонентов.

В результате наблюдаемые конфигурационные параметры отражают совокупное состояние общей среды исполнения и не могут быть однозначно соотнесены с конкретным объектом конфигурационного контроля, что затрудняет формирование параметра $s_i(t)$ для отдельного программного компонента в виде структурированного объекта, соответствующего выражению (2.2).

На рисунке 13 представлена схема общего окружения исполнения, при которой все компоненты используют единую среду, что увеличивает вероятность возникновения несовместимостей, обусловленных различиями в версиях используемых библиотек и конфигурационных параметрах.

Рисунок 13 – Общее окружение исполнения компонентов программной среды

Решением указанных проблем является применение технологий программной виртуализации и контейнеризации, получивших широкое распространение в информационных технологиях и позволяющих создавать изолированные среды исполнения для отдельных программных компонентов [60].

Как известно, изоляция среды исполнения обеспечивает локализацию параметров окружения в границах конкретного компонента, за счет чего достигается независимость параметров исполнения, упрощается контроль и обеспечивается воспроизводимость поведения модулей программной среды.

Однако, несмотря на очевидные преимущества, указанные технологии до сих пор редко встречаются в архитектурах программных сред АСУТП. Вместе с тем применение изолированных сред позволяет задавать фиксированные параметры исполнения, полностью устранять влияние изменений извне и значительно повышать воспроизводимость процессов запуска компонентов программной среды АСУТП. На рисунке 14 представлена схема, иллюстрирующая организацию изолированных сред исполнения.

Рисунок 14 – Изолированные окружения компонентов

Виртуализация обеспечивает создание независимых операционных сред, каждая из которых содержит собственный набор программных и системных компонентов [61]. Управление виртуальными машинами (ВМ) реализуется посредством гипервизора [62], формирующего виртуальные аппаратные ресурсы для каждой отдельной инстанции. Такой механизм позволяет добиться полной изоляции рабочих окружений и широко применяется для одновременного запуска разнородных приложений на единой физической платформе [63]. Вместе с тем, для функционирования каждой виртуальной машины требуется полноценная ОС, что приводит к существенному увеличению аппаратных затрат [64]. Масштабирование такой архитектуры сопровождается значительным ростом потребления вычислительных ресурсов и снижением эффективности инфраструктуры. Это затрудняет создание гибких и управляемых изолированных сред исполнения, особенно при массовом развертывании однотипных компонентов [65].

В результате обозначенные ограничения традиционной виртуализации стимулируют развитие альтернативных решений, нацеленных на экономию ресурсов и ускорение процессов внедрения.

На рисунке 15 показана архитектура виртуализации, при которой гипервизор запускает несколько независимых виртуальных машин с собственными ОС и приложениями.

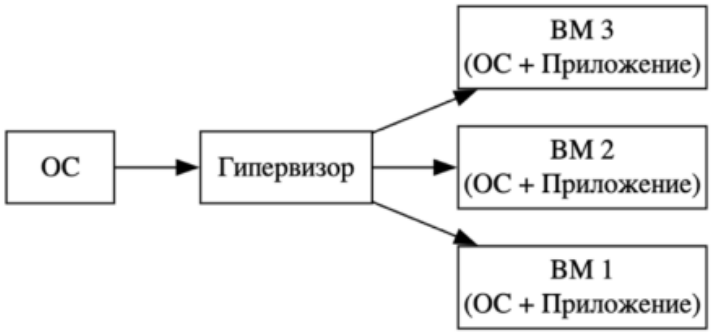

Рисунок 15 – Принципиальная схема виртуализации

Для преодоления обозначенных ограничений виртуализации в области создания изолированных сред исполнения активно развивается программная контейнеризация [66]. Эта технология обеспечивает независимость компонентов за счет использования механизмов ядра операционной системы, устраняя

необходимость выделения отдельной гостевой ОС для каждого экземпляра приложения [67]. Тем самым контейнеризация существенно повышает эффективность использования ресурсов и значительно упрощает поддержание нормативных параметров программной среды [68, 69].

В частности, для ОС Windows разработана технология Windows Containers, предназначенная для изолированного выполнения компонентов в рамках отдельного контейнера. Однако, в российских промышленных системах ее применение невозможно из-за санкционной политики компании Microsoft [70], которая включает прекращение официальной поддержки [71], блокировку обновлений и ограничение предоставления сервисов промышленным предприятиям [72].

В этих условиях реализация изолированных сред исполнения ориентируется преимущественно на операционные системы семейства Linux [73], включая сертифицированные решения для промышленной автоматизации (РЕД ОС, Astra Linux, Альт, ОСнова) [74, 75]. Эти платформы предоставляют встроенные механизмы контейнеризации [76] и позволяют формировать независимые окружения для исполнения программных компонентов [77].

В Linux изоляция достигается за счет использования пространств имен (namespaces) и контрольных групп (cgroups) [78]. Пространства имен обеспечивают логическую изоляцию процессов, файловых систем и сетевых интерфейсов, формируя самостоятельные контексты для каждого контейнера. Контрольные группы регулируют использование вычислительных ресурсов, таких как процессорное время, объем оперативной памяти и дисковое пространство, предотвращая ситуации, когда один компонент может негативно повлиять на работу других за счет чрезмерного потребления ресурсов.

Указанные механизмы реализованы на уровне ядра Linux и формируют базу для построения контейнерных сред, в которых компоненты функционируют независимо и обладают управляемым доступом к выделенным ресурсам. На рисунке 16 представлена схема, иллюстрирующая организацию взаимодействия

между компонентами ядра Linux [79], пространствами имен и контрольными группами, формирующими изолированное контейнерное окружение.

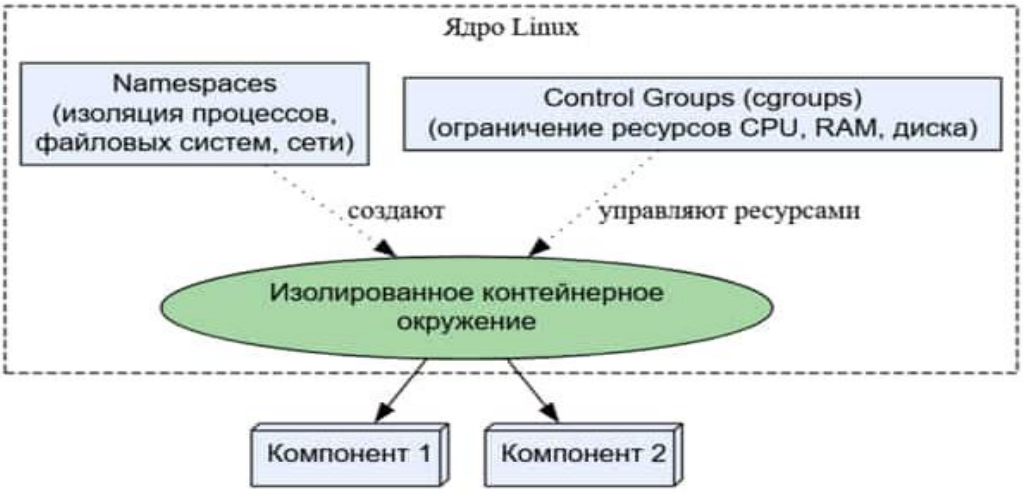

Рисунок 16 – Принципиальная схема изоляции компонентов в Linux с использованием пространств имен и контрольных групп (cgroups)

На основе данных механизмов формируется модель контейнерного окружения [79], где каждый компонент разворачивается в изолированном контейнере с фиксированными параметрами среды. Контейнер создается на основе образа (файла), содержащего операционную оболочку, исполняемые файлы, библиотеки и конфигурационные данные.

Контейнерный образ выступает в роли шаблона, из которого формируются экземпляры окружений исполнения, обеспечивая воспроизводимость и переносимость компонентов независимо от параметров инфраструктуры [77].

Для управления изолированными окружениями используются специализированные программные средства, предназначенные для создания, запуска и сопровождения контейнеров [67]. Одним из наиболее зрелых и широко используемых решений является Docker [80-82], предоставляющий полный стек контейнеризации [83]. Он включает инструменты для сборки образов (Dockerfile), механизм выполнения контейнеров (Docker Engine) и систему хранения и доставки образов (Docker Registry) [80]. Визуализация схемы контейнеризации приведена на рисунке 17.

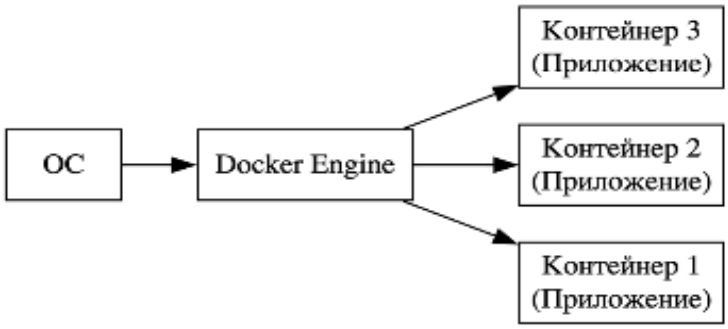

Рисунок 17 – Принципиальная схема контейнеризации

В Docker контейнерная среда строится как многослойный образ, где каждый слой содержит только те изменения, которые были внесены на соответствующем этапе сборки. Такая структура предотвращает накопление избыточных данных и полностью устраняет влияние изменений одного компонента на другие [84]. Контейнерный образ представляет собой неизменяемую совокупность слоев, включающую системные библиотеки, конфигурационные файлы и исполняемые компоненты [85]. Все изменения фиксируются в отдельном слое, который может быть удален или пересобран без воздействия на остальные слои.

В типовой структуре контейнерного образа на базовом уровне размещен образ операционной системы, определяющий программную базу дистрибутива Linux. Следующий слой предназначен для установки дополнительных зависимостей и других необходимых компонентов. Далее размещается прикладной код, включающий исполняемые модули и файлы приложения. Завершающим элементом является слой, обеспечивающий запуск и изоляцию контейнера в целевой среде.

Наряду с Docker в настоящее время используется ряд альтернативных решений для организации изолированных сред исполнения, каждое из которых ориентировано на определенные сценарии применения [86]. Одним из таких инструментов является Podman [87, 88] – контейнерный движок, совместимый с синтаксисом Docker CLI благодаря пакету podman-docker [89]. Принципиальной особенностью Podman является возможность управления контейнерами без запуска отдельного фонового демона, что снижает нагрузку на хостовую систему и

позволяет реализовать модель минимального доверия, востребованную в критически важных сегментах промышленной автоматизации.

Еще одним примером являются системы LXC и LXD [90], реализующие изоляцию на уровне операционной системы. Эти средства предоставляют структуру контейнеров, максимально приближенную к виртуальным машинам, что особенно важно в ситуациях, когда необходим детальный контроль за параметрами среды исполнения.

Для задач кластеризации используется контейнерный движок CRI-O [91], полностью совместимый со спецификацией OCI (Open Container Initiative) [86] и предназначенный для интеграции с платформой Kubernetes. Этот инструмент не применяется как самостоятельное средство контейнеризации, а выступает компонентом инфраструктуры для управления контейнерами в распределенных системах [92].

При необходимости сборки и распространения контейнерных образов без использования фонового демона Docker применяются специализированные утилиты, такие как Buildah [93] и Skopeo [94, 95]. Buildah обеспечивает создание образов в формате OCI, а Skopeo предназначен для управления образами и их репликации между различными хранилищами.

Таким образом, применение контейнеризации в средах на базе Linux позволяет формировать независимые и устойчивые окружения исполнения для компонентов программной среды АСУТП. Этот механизм существенно снижает влияние внешних изменений и минимизирует риск возникновения конфликтов между элементами прикладного программного обеспечения. Наиболее зрелыми и широко используемыми решением в данной области является Docker [92] для одиночных серверов и Kubernetes для кластерных систем.

В рамках настоящего исследования именно эти средства рассмотрены в качестве основной технологической базы для построения воспроизводимых и управляемых сред исполнения. Они отличаются развитой экосистемой и обширной документацией, существенно облегчая внедрение контейнеризации в задачи сопровождения программной среды промышленной автоматизации.

2.3.2 Модель стандартизированной среды исполнения программных компонентов

В целях обеспечения изолированного и воспроизводимого исполнения программных компонентов АСУТП в настоящем исследовании разработана модель стандартизированной среды исполнения, реализуемая средствами контейнеризации. Основной особенностью предложенной модели выступает четкая послойная структура контейнерного образа, в которой помимо минималистичного базового слоя [96] выделяются отдельные слои с набором библиотек и зависимостей, слоем программного обеспечения программной среды АСУТП, а также слоем конфигурационных и надстроечных параметров.

Стремление к минимизации размера контейнерных образов реализуется за счет выделения слоя библиотек и зависимостей в отдельный уровень, который позволяет повторно использовать этот слой при сборке различных компонентов с одинаковыми требованиями к окружению. Данный механизм существенно снижает объем дублируемых данных и повышает эффективность сопровождения программной среды.

Модель среды I_{model} исполнения формализована в виде совокупности независимых слоев (2.12):

$$I_{model} = I_{base} + I_{deps} + I_{app} + I_{conf} \quad (2.12)$$

где:

1. I_{base} – базовый слой (минималистичный дистрибутив);
2. I_{deps} – слой библиотек и зависимостей;
3. I_{app} – слой программного компонента;
4. I_{conf} – слой конфигурации.

Визуализация контейнерной модели представлена на рисунке 18.

Рисунок 18 – Контейнерная модель компонентов программной среды АСУТП

В случае обновления или модификации программного компонента, когда меняется слой исполняемых файлов I_{app}, новый контейнерный образ I_{model} формируется на основе предыдущих неизменных слоев – базового I_{base}, слоя зависимостей I_{deps} и слоя конфигурации I_{conf}, к которым добавляется обновленный прикладной слой. Такая структура выражается соотношением (2.13).

$$I^*_{model} = I_{base} + I_{deps} + I^*_{app} + I_{conf} \quad (2.13)$$

где все слои, кроме I^*_{app}, идентичны их значениям в исходном образе.

Для количественной оценки введем обозначения. Полный размер контейнерного образа – V_{full}, размер обновленного слоя – V^*_{layer}, данных, требующий передачи или обработки при обновлении – V_{update}. В рамках предложенной модели выполняется условие минимизации – при изменении компонента пересобирается только модифицированный слой (2.14).

$$V_{update} = V^*_{layer}, \quad V^*_{layer} \ll V_{full} \quad (2.14)$$

Данная модель позволяет существенно сократить объем передаваемых данных [99] и ускорить обновление компонентов программной среды АСУТП за счет повторного использования неизменных слоев контейнерного образа.

Визуализация процесса обновления при изменении только прикладного слоя представлена на рисунке 19.

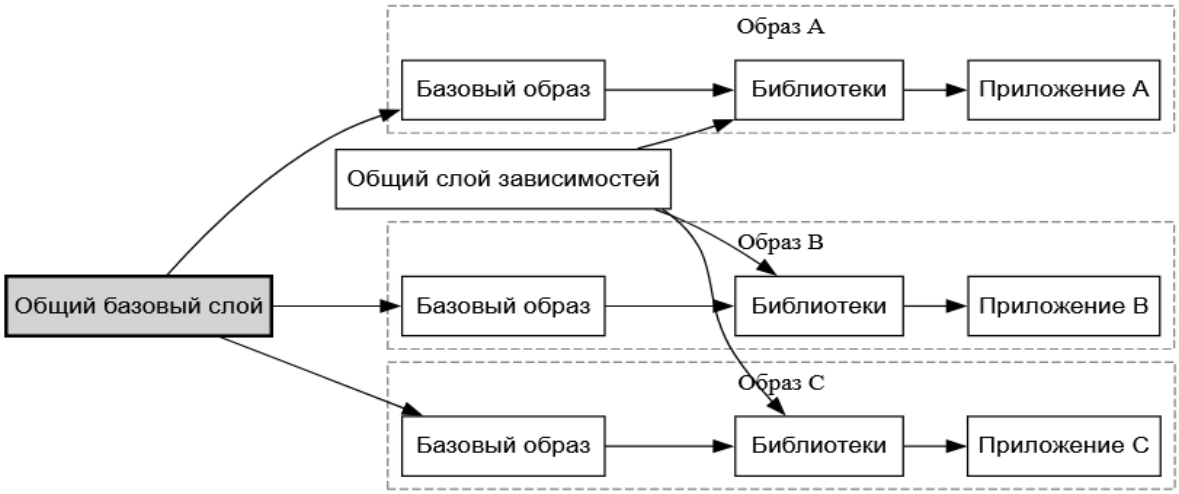

Рисунок 19 – Повторное использование слоев в модели

Включение многоступенчатой сборки (multistage build) в модель среды исполнения позволяет дополнительно минимизировать размер образа и исключить накопление избыточных компонентов [97]. Это предполагает разделение процесса на этап компиляции и финальное формирование контейнера. Если программный компонент поставляется в виде исходного кода, на первом этапе осуществляется сборка с использованием всех необходимых зависимостей и инструментальных средств, в результате чего формируется исполняемый файл, впоследствии переносящийся в итоговый образ [98]. После завершения компиляции из финального контейнера удаляются временные файлы и вспомогательные утилиты, что позволяет получить компактный и оптимизированный образ, не содержащий лишних элементов.

В случаях, когда программный компонент распространяется в виде готового исполняемого файла, этап компиляции пропускается, и готовый файл сразу переносится в финальный контейнерный образ.

Оба варианта представлены на рисунке 20.

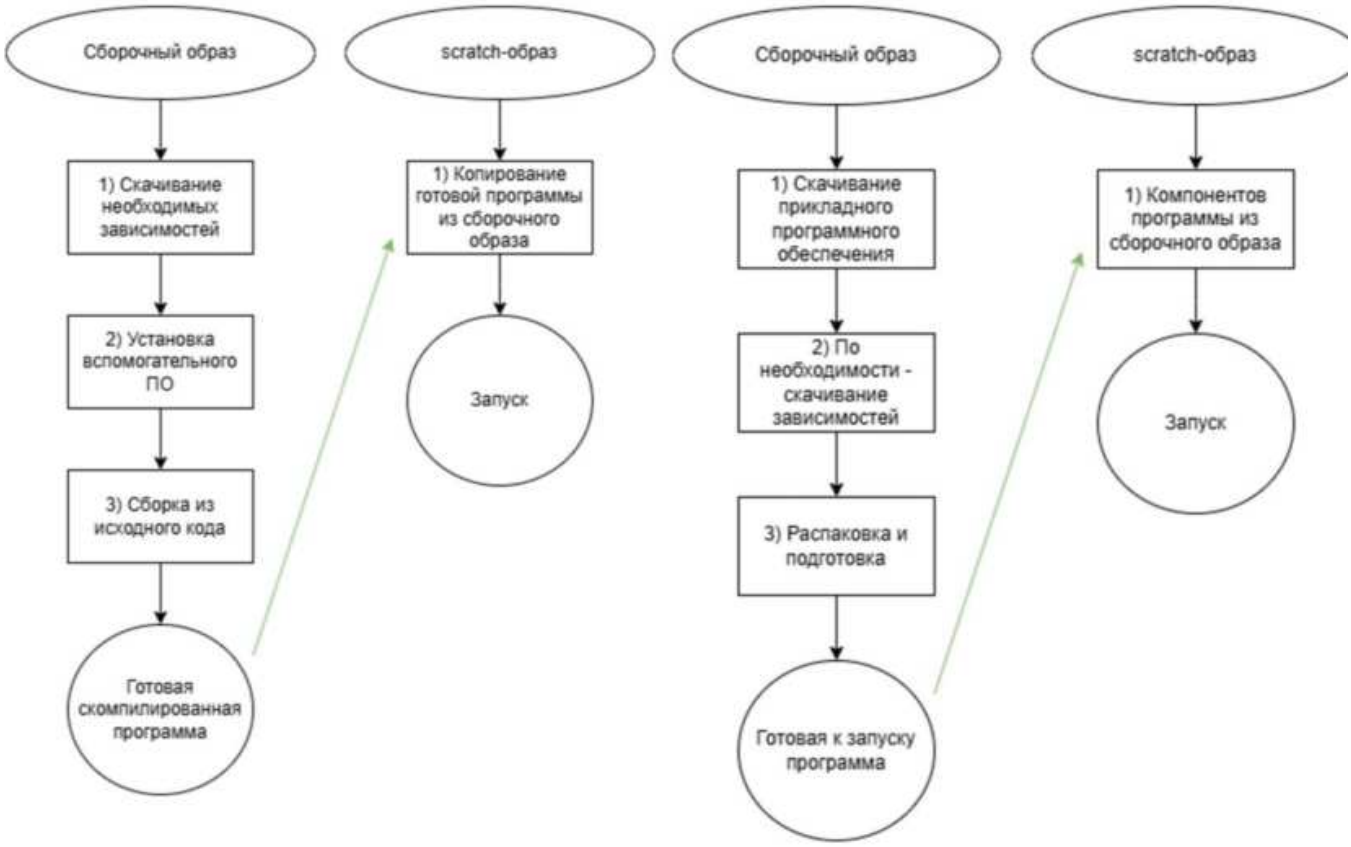

Рисунок 20 – Варианты применения многоступенчатой сборки контейнерного образа (с исходным кодом и с готовым исполняемым файлом)

Общим правилом для обоих сценариев остается исключение всех вспомогательных элементов, не связанных с непосредственным исполнением приложения, способствуя компактности и воспроизводимости среды.

2.4 Автоматизация сопровождения конфигурации программной среды АСУТП

2.4.1 Модель системы сопровождения

Для формализованного представления процессов эксплуатации программной среды АСУТП в работе предлагается система сопровождения (см. раздел 1.7), ориентированная на фиксацию параметров конфигурации программной среды, мониторинг изменений с выявлением отклонений от $S_{\text{цел}}$ и автоматизацию

формирования проекта корректирующих воздействий для устранения расхождений.

Система сопровождения определяется как структурированный механизм, обеспечивающий непрерывный контроль за $S(t)$, регистрацию изменений конфигурационных параметров и предотвращение накопления отклонений.

В связи с этим актуализируется задача выбора организационных принципов, позволяющих обеспечить непрерывное сопровождение программной среды и поддержание согласованной конфигурации.

Для программных сред АСУТП специализированные и формализованные концепции автоматизации указанных процессов в настоящее время не получили широкого распространения, что обусловливает целесообразность обращения к принципам, сформированным в смежных областях ИТ.

В настоящем исследовании в основу концептуальной модели системы сопровождения программной среды АСУТП положены принципы CI/CD (Continuous Integration / Continuous Delivery) [100], интерпретированные с позиций системного сопровождения и эксплуатации программных компонентов. Их применение обеспечивает формализованный, воспроизводимый и контролируемый процесс интеграции изменений, снижает вероятность ошибок при обновлениях, сокращает время перехода к новым версиям и поддерживает согласованность среды исполнения (рисунок 21) [101].

Подтвержденная эффективность принципов CI/CD при управлении сложными распределенными системами [102] послужила основанием для их адаптации к задачам промышленной автоматизации, в рамках которых акцент смещается с приоритета максимальной скорости внедрения изменений на достижение корректности и согласованности конфигурации программной среды.

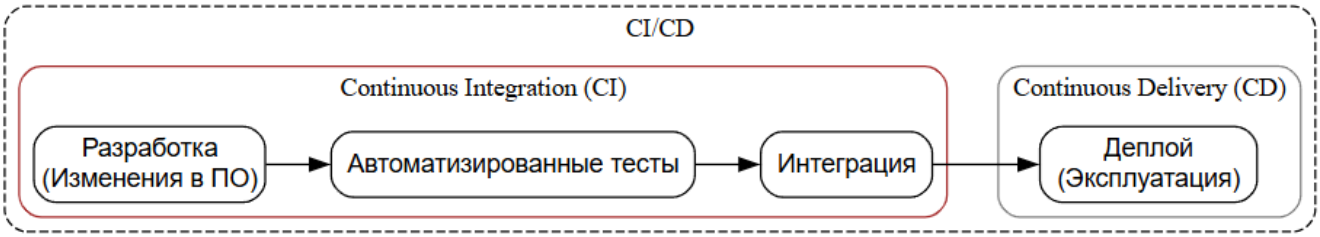

Рисунок 21 – Стандартная модель CI/CD

В разрабатываемой системе формируется замкнутый контур управления $S(t)$, в рамках которого автоматизированный контроль изменений, синхронизация и восстановление параметров среды исполнения осуществляются в соответствии с установленными характеристиками [103].

В отличие от традиционного применения CI/CD в области ИТ, где основной акцент делается на скорости и частоте поставки новых версий, в АСУТП данные механизмы ориентированы на поддержание корректности программной среды исполнения компонентов.

Взаимодействие с программной средой выполняется в строго регламентированном порядке и фиксируется в системе контроля версий, давая возможность последующего аудита и гарантированного восстановления $S_{\text{цел}}$ в случае конфигурационного расхождения. Централизация управления позволяет минимизировать объем ручных операций и повысить предсказуемость функционирования программной среды.

На рисунке 22 представлена модифицированная структура CI/CD, адаптированная для задач автоматизации сопровождения программной среды АСУТП и реализующая замкнутый контур контроля конфигурации и восстановления программной среды.

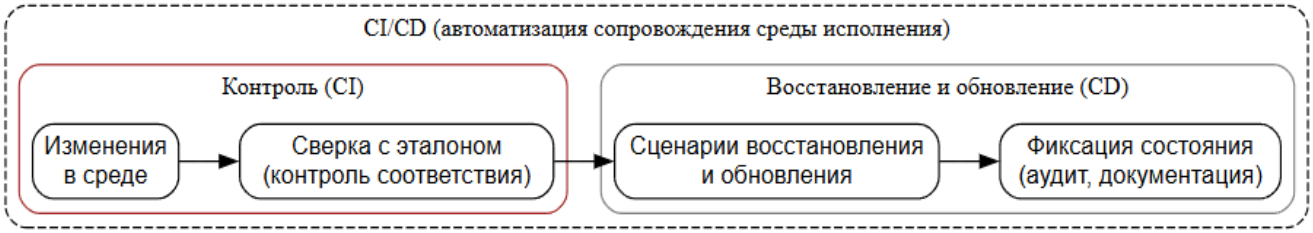

Рисунок 22 – Модель слоя сопровождения для АСУТП

Для реализации журналирования и архивирования истории изменения параметров, в работе используется система контроля версий [104, 105], обеспечивающая хранение формализованных описаний конфигурации и фиксацию всех изменений. В качестве базового инструмента выбран Git [106].

В настоящем исследовании в качестве целевой организационно-технологической площадки реализации слоя сопровождения выбрана платформа GitLab [103, 107], распространяемая как свободное программное обеспечение (редакция Community Edition). Платформа объединяет средства контроля версий с функциональностью CI/CD [108], обеспечивая централизованное управление обновлениями, восстановлением компонентов и аудитом операций сопровождения.

Возможность развертывания экземпляра GitLab на собственных вычислительных ресурсах позволяет учитывать требования промышленной безопасности и формировать изолированную инфраструктуру, адаптированную к условиям промышленной автоматизации.

Функциональные возможности GitLab обеспечивают создание единого пространства управления, в рамках которого автоматизируются все основные этапы сопровождения программной среды АСУТП. Платформа предоставляет средства контроля целостности изменений, управления обновлениями, восстановления компонентов и аудита операций сопровождения [109]. Структурная схема представлена на рисунке 23.

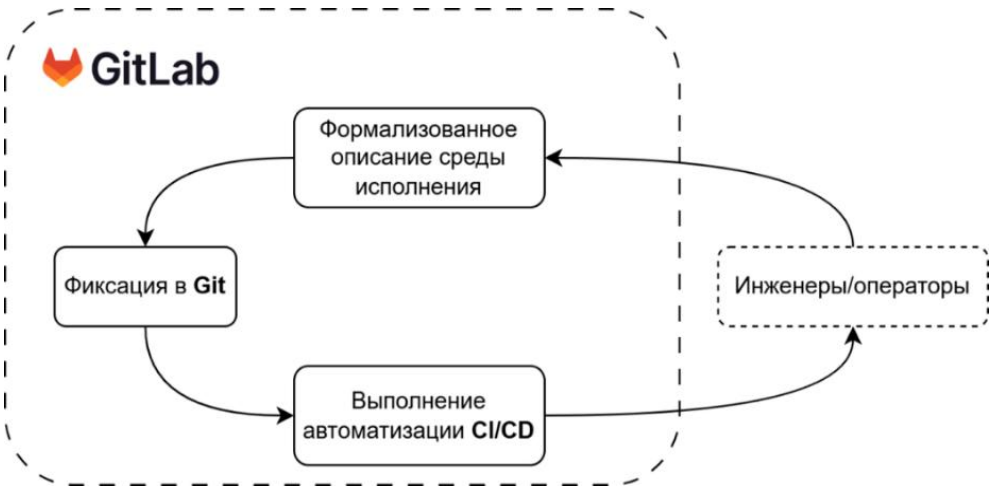

Рисунок 23 – Структура процессов автоматизации сопровождения

На схеме отражены взаимодействия и последовательность автоматизированных процессов, реализованных на базе GitLab.

2.4.2 Архитектурные варианты размещения и ограничения применения системы сопровождения

Порядок размещения и эксплуатации системы сопровождения среды исполнения на базе GitLab [103] определяется архитектурой ее реализации, как это сформулировано ранее в разделе 1.7 и особенностями технологической площадки, под которой в настоящей работе понимается локализованный объект автоматизации, оснащенный собственными вычислительными и сетевыми ресурсами для размещения компонентов среды исполнения.

В замкнутой архитектуре система сопровождения и программная среда АСУТП располагаются на одном вычислительном узле. Такой вариант допускается в тех случаях, когда вычислительный ресурс выступает в роли промышленного сервера и обладает необходимым уровнем производительности и устойчивости. Все процедуры сопровождения выполняются локально, без привлечения внешних каналов связи.

Следует отметить, что возможны ситуации, когда все компоненты программной среды АСУТП размещаются на одном устройстве, одновременно выполняющем функции сервера и автоматизированного рабочего места (АРМ) оператора (рисунок 24).

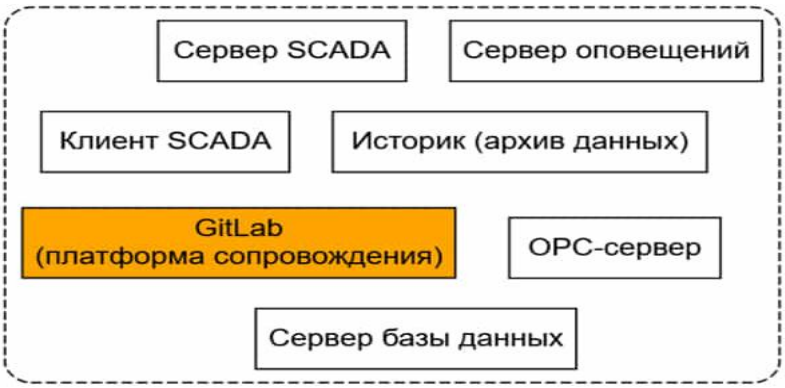

Рисунок 24 – Пример перегрузки программной среды на одном АРМ оператора

Такой способ размещения системы сопровождения следует рассматривать исключительно как технически допустимый, но эксплуатационно нежелательный вариант.

В архитектуре с внутренним управляющим контуром система сопровождения размещается на отдельном вычислительном узле в пределах локальной сети АСУТП. Такой вариант обеспечивает централизованный контроль программной среды и автоматизацию процедур обновления, восстановления и фиксации изменений, не выводя систему сопровождения за пределы локального сегмента.

Платформу GitLab возможно использовать для координации сопровождения нескольких узлов в рамках одной технологической площадки (рисунок 25), сохраняя требуемый уровень изоляции и управляемости. Организация взаимодействия между программной средой АСУТП и системой сопровождения осуществляется по внутренним каналам связи без риска передачи данных за пределы объекта автоматизации.

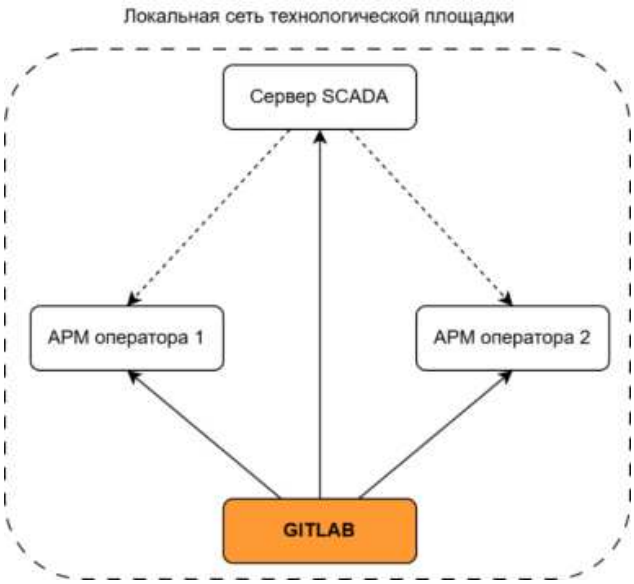

Рисунок 25 – Пример замкнутой архитектуры

В случае применения архитектуры с вынесенным управляющим контуром система сопровождения размещается вне обслуживаемой площадки (рисунок 26). Такая модель применяется для координации сопровождения программными средами АСУТП сразу на нескольких технологических площадках.

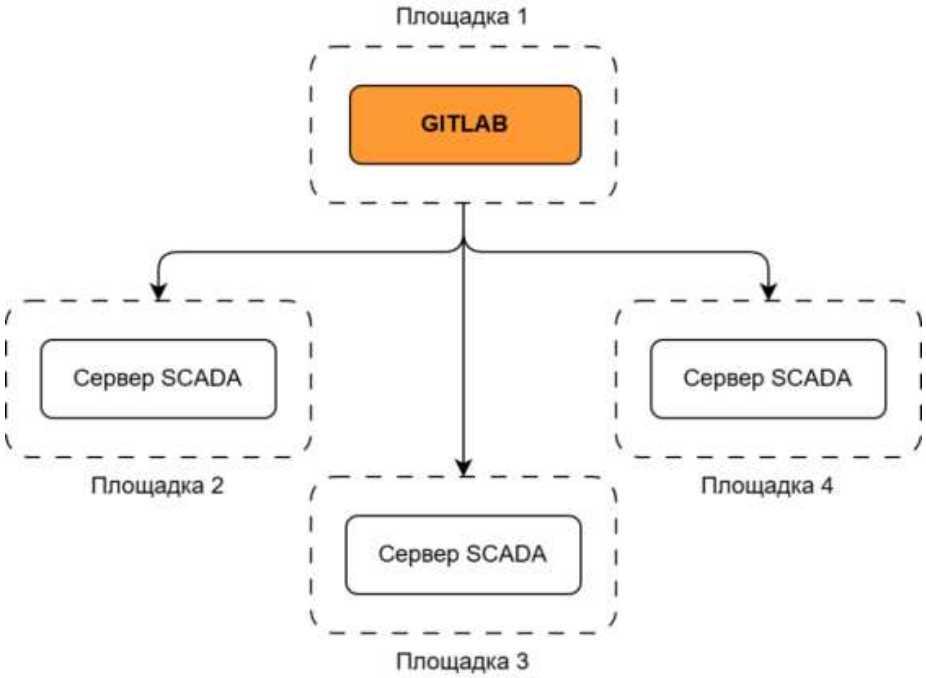

Рисунок 26 – Пример вынесенного контура сопровождения

В данной конфигурации требуется обеспечить защищенность каналов связи, разграничение доступа и выполнение требований к информационной безопасности, поскольку обмен служебными данными осуществляется между удаленными площадками и центральной системой сопровождения. Применимость такой схемы ограничивается случаями, когда допускается передача информации об используемой конфигурации программной среды АСУТП за пределы технологической площадки, а также когда организационные и технические условия позволяют централизовать сопровождение.

2.4.3 Функционирование замкнутого контура сопровождения программной среды

На основе описания требуемой конфигурации $S_{\text{цел}}$ в системе GitLab создается проект, в котором данное описание фиксируется и используется в качестве опорного представления для последующего контроля конфигурационного состояния программной среды.

Декларативная модель управления определяет процедуру регулярного сопоставления $S(t)$ с $S_{\text{цел}}$, реализуемую с использованием механизмов CI/CD, интегрированных в платформу GitLab. Проверка выполняется по установленному регламенту либо инициируется при возникновении событий, указывающих на возможные конфигурационные отклонения.

В случае фиксации расхождений $e(t)$ между $S_{\text{цел}}$ и $S(t)$, а также при возникновении нештатных ситуаций формируются записи об инцидентах и проект корректирующих воздействий ΔS, направленный на приведение программной среды к $S_{\text{цел}}$ в соответствии с выражением (2.11).

Как было указано в постановке задачи управления (см. раздел 1.6), применение проекта корректирующих воздействий осуществляется ответственным лицом в согласованные временные интервалы с целью исключения влияния на информационную модель технологического процесса.

Результаты выполнения процедур сопровождения, сведения о восстановлении конфигурации, внедренных обновлениях и зарегистрированных инцидентах сохраняются в структурированном виде, что обеспечивает возможность последующего аудита и анализа.

В результате формируется система сопровождения с замкнутым контуром управления, представленная на рисунке 27.

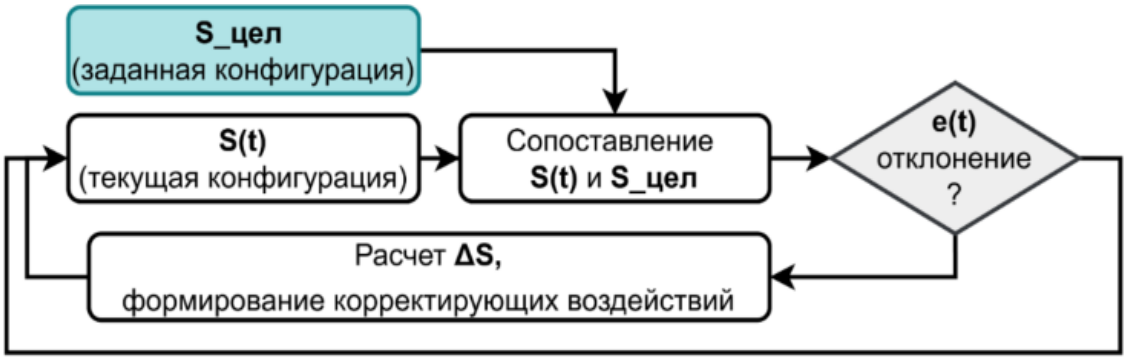

Рисунок 27 – Система сопровождения с замкнутым контуром управления

Система обеспечивает централизованную организацию процедур контроля и приведение программной среды к $S_{\text{цел}}$ в процессе эксплуатации.

Выводы по 2 главе

Разработана модель фактической конфигурации программной среды, описывающая текущую совокупность контролируемых параметров программной среды и допускающая структурированное описание объектов конфигурации с учетом их идентификаторов, типов, значений параметров и иерархических зависимостей. Обоснованы критерии полноты и достоверности формализованного представления, определяющие его пригодность для автоматизированного сопоставления с нормативным конфигурационным описанием без привлечения экспертной интерпретации.

Разработан императивный метод выполнения операций сопровождения для управления конфигурацией программной среды АСУТП, обеспечивающий регламентированное поэтапное преобразование конфигурации программной среды посредством заранее заданной последовательности операций сопровождения и реализующий исполнение корректирующих воздействий на исполнительном уровне.

Разработан декларативный метод формирования корректирующих воздействий для управления конфигурацией программной среды АСУТП, основанный на автоматизированном сопоставлении фактической и нормативной конфигураций, формировании формализованного описания отклонений и синтезе проекта корректирующих воздействий, подлежащих реализации с использованием императивного метода выполнения операций сопровождения.

Обосновано, что воспроизводимость конфигурационного состояния и корректность формирования параметров фактической конфигурации непосредственно определяются организацией среды исполнения программных компонентов. Показано, что использование изолированных сред исполнения обеспечивает локализацию параметров окружения в границах отдельных компонентов, снижает влияние изменений общего программного окружения и повышает однозначность конфигурационного контроля. Рассмотрены технологии

виртуализации и контейнеризации как средства реализации изолированных сред исполнения программных компонентов.

Разработана модель автоматизации сопровождения программной среды, формирующая замкнутый контур контроля фактической конфигурации. Модель обеспечивает фиксацию нормативного конфигурационного описания, регулярное сопоставление фактического состояния с целевым, регистрацию конфигурационных отклонений и формирование проекта корректирующих воздействий и обеспечение их регламентированной реализации ответственным эксплуатационным персоналом.

Рассмотрены архитектурные варианты размещения системы сопровождения и ограничения их применения в условиях различных технологических площадок, что обеспечивает возможность применения разработанной модели в реальных условиях эксплуатации.

ГЛАВА 3. РАЗРАБОТКА СИСТЕМЫ СОПРОВОЖДЕНИЯ ПРОГРАММНОЙ СРЕДЫ АСУТП ДЛЯ ТЕХНОЛОГИЧЕСКОГО ОБЪЕКТА ВОДОЗАБОРНОГО УЗЛА

3.1. Условия экспериментальной верификации и требования к программной среде

3.1.1 Экспериментальный объект и среда внедрения

Для экспериментальной верификации разработанных методов сопровождения, реализованных в виде системы сопровождения программной среды АСУТП, использованы условия реальной эксплуатации диспетчерского уровня при непрерывном режиме функционирования.

В качестве экспериментальной базы использована программная среда диспетчерского уровня АСУТП, реализованная средствами SCADA-системы и функционирующая в составе водозаборного узла, обеспечивающего круглосуточное водоснабжение жилого комплекса «Томилино-Парк» и сопутствующих промышленных объектов.

Водозаборный узел в рамках данного исследования рассматривается как среда эксплуатации программной среды диспетчерского уровня и источник технологических данных, формируемых в процессе функционирования оборудования и поступающих в диспетчерскую подсистему.

Технологическая структура объекта включает две группы артезианских скважин (СКВ), относящихся к Подольскому и Алексинско-Протвинскому водоносным горизонтам, каждая из которых, в свою очередь, состоит из трех скважин.

Вода из скважин поступает на три станции водоподготовки, где осуществляется обезжелезивание, реагентная обработка и очистка с применением установок обратного осмоса [156, 157].

Структурная схема водозаборного узла приведена на рисунке 28.

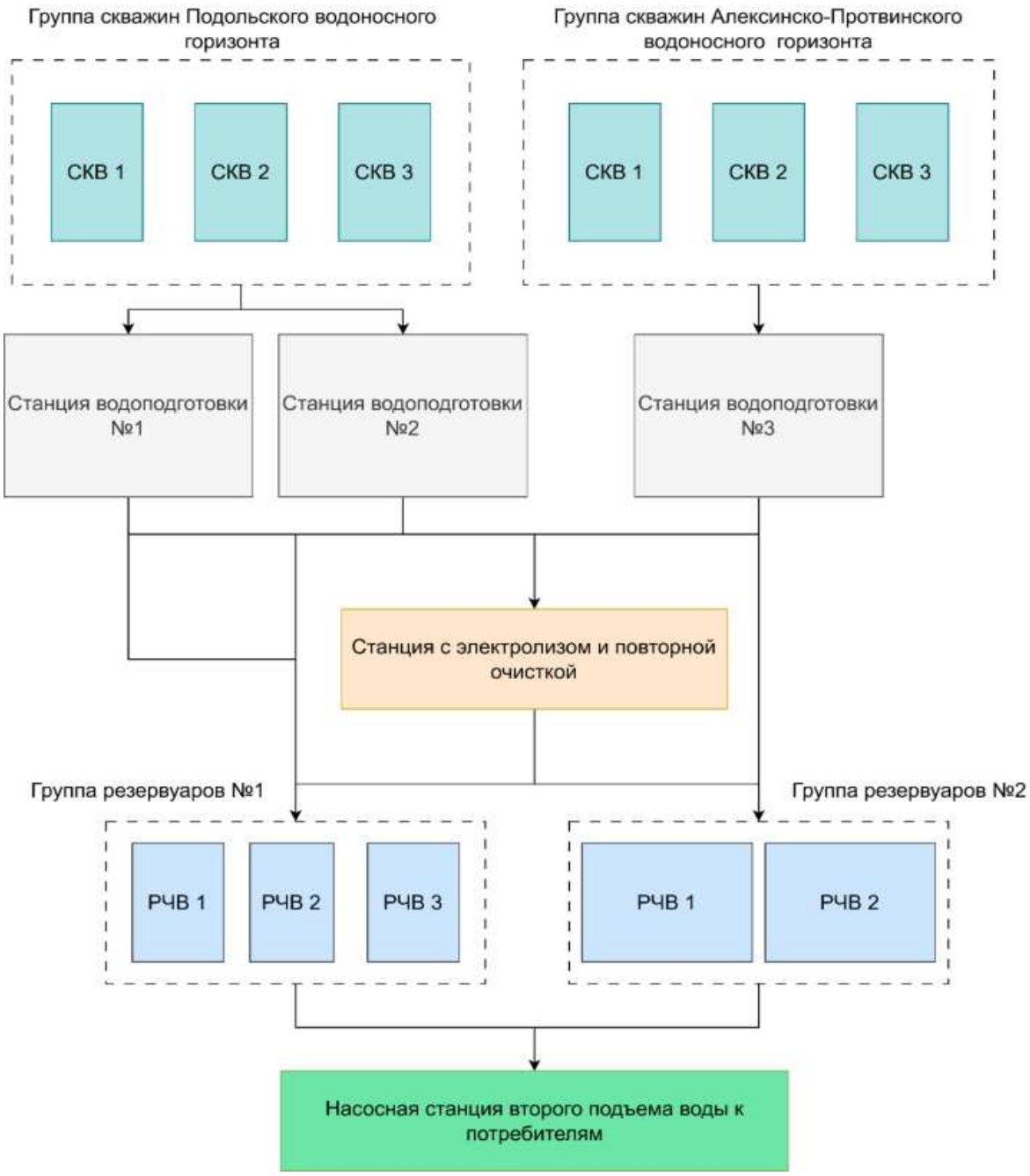

Рисунок 28. Структурная схема объекта водозаборного узла

После водоподготовки очищенная вода распределяется по двум группам резервуаров чистой воды (РЧВ), откуда подается на насосную станцию второго подъема. Насосные агрегаты оснащены частотно-регулируемыми приводами, обеспечивающими поддержание заданных параметров давления [157, 158].

Часть воды, не соответствующая установленным требованиям либо подлежащая повторной обработке, направляется на отдельную станцию электролизной обработки и дополнительной очистки [156].

3.1.2 Условия проведения экспериментальной верификации методов

В процессе эксплуатации водозаборного узла на момент начала исследования была зафиксирована необходимость модернизации диспетчерской программной среды, обусловленная расширением и усложнением структуры технологического комплекса, а также изменением требований к программной платформе диспетчерского уровня АСУТП. В частности, в качестве исходных условий были приняты требования по использованию отечественных программных платформ и операционных систем семейства Linux в рамках программы импортозамещения Российской Федерации [12, 114].

Эксплуатировавшаяся SCADA-система не обеспечивала возможности развития информационно-технического покрытия водозаборного узла и находилась на завершающем этапе жизненного цикла, что обусловило необходимость ее замены. В рамках модернизации была разработана и внедрена новая диспетчерская программная среда, использованная в качестве экспериментальной платформы для верификации методов сопровождения.

Внутренняя архитектура новой диспетчерской программной среды была реализована в виде компонентной модели, тогда как ранее эксплуатировавшаяся система имела монолитное исполнение.

Как показано в разделе 1.3, компонентная организация программной среды охватывает операции сопровождения монолитных систем и дополняет их задачей контроля связей между элементами программной среды, обеспечивая репрезентативность экспериментальной верификации методов сопровождения для обеих архитектур.

3.1.3 Архитектура программной среды АСУТП верификационной площадки

В рассматриваемой реализации программная среда диспетчерского уровня АСУТП представлена в виде совокупности автономных программных модулей [52, 112, 113] с явно определенными интерфейсами взаимодействия. Архитектура

среды исполнения предусматривает функциональную обособленность отдельных элементов и их взаимодействие посредством регламентированных интерфейсов и каналов обмена данными.

Для каждого программного модуля задано отдельное конфигурационное описание, определяющее параметры его функционирования и связи с другими программными модулями [52, 53]. Такое представление обеспечивает формализацию состава программной среды и структуры взаимодействий между ее элементами, а также возможность их согласованного сопровождения в процессе эксплуатации.

Структурная схема реализованной программной среды АСУТП в компонентном исполнении [53] приведена на рисунке 29.

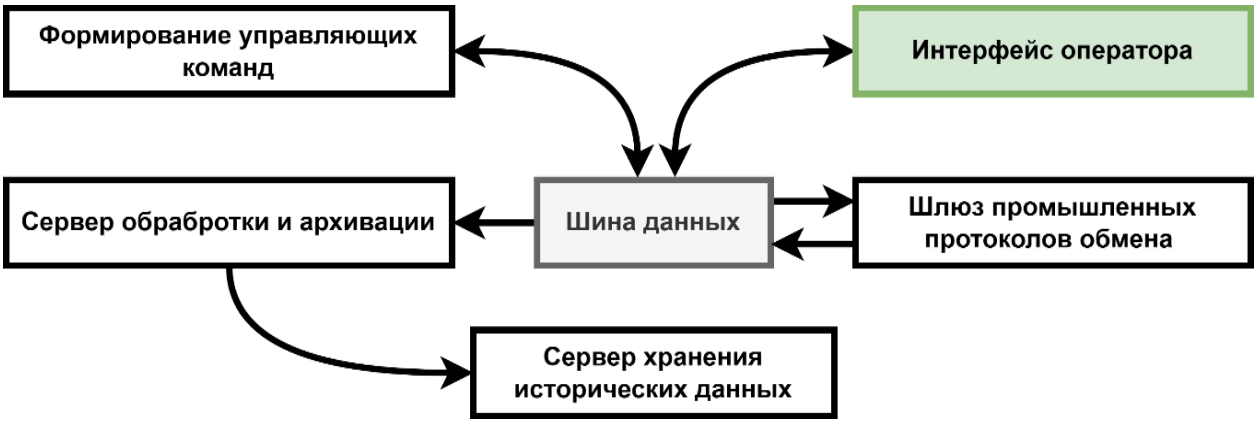

Рисунок 29. Структурная схема компонентной архитектуры программной среды

Детальное описание состава программных модулей и особенностей их взаимодействия представлено в последующих разделах главы.

3.1.4 Критерии успешной экспериментальной верификации

Для экспериментальной верификации разработанных методов управления конфигурацией программной среды диспетчерского уровня АСУТП водозаборного узла принимается совокупность критериев, характеризующих корректность функционирования контура управления конфигурацией в терминах

формализованных представлений $S(t)$ и $S_{\text{цел}}$, оператора сопоставления δ, вектора отклонений $e(t)$ и функции формирования корректирующих воздействий F.

Первый критерий заключается в корректности выполнения процедур сопровождения при отсутствии конфигурационных отклонений.

При эквивалентности фактической и нормативной конфигураций программной среды в заданной области конфигурационного контроля ($S(t) \equiv S_{\text{цел}}$) повторное выполнение процедур сопровождения не должно приводить к изменению состава программных компонентов, значений их конфигурационных параметров и связей между ними.

Иными словами, процедуры сопровождения в данном случае должны обладать свойством идемпотентности по отношению к контролируемому конфигурационному состоянию программной среды.

Второй критерий заключается в корректности автоматизированного выявления и формализации конфигурационных отклонений.

При наличии расхождений между фактической и нормативной конфигурациями программной среды в заданной области конфигурационного контроля система сопровождения должна обеспечивать выявление факта несоответствия $S(t) \neq S_{\text{цел}}$ и формирование формализованного представления отклонений в виде ненулевого вектора $e(t)$.

При этом вектор $e(t)$ должен однозначно отражать локализацию и характер выявленных конфигурационных отклонений по объектам конфигурации и их параметрам без привлечения экспертной интерпретации.

Третий критерий заключается в корректности формирования проекта корректирующих воздействий и достижении нормативного конфигурационного состояния.

На основании сформированного вектора отклонений $e(t)$ должна быть вычислена совокупность корректирующих воздействий ΔS, однозначно определяющая действия, необходимые для приведения конфигурации

программной среды к нормативному конфигурационному описанию в заданной области конфигурационного контроля (2.10)-(2.11).

Применение сформированного проекта корректирующих воздействий в области конфигурационного контроля формализуется выполнением условия, описанного выражением (3.15).

$$\delta\left(S_{\text{цел}}, \Delta S\left(S(t)\right)\right) = 0 \tag{3.15}$$

Данное выражение обеспечивает достижение эквивалентности фактической и нормативной конфигураций программной среды.

3.2. Реализация компонентов программной среды АСУТП

В данном разделе рассматриваются программные компоненты диспетчерского уровня АСУТП, реализованные в рамках экспериментальной программной среды и используемые для практической апробации предложенных методов сопровождения.

3.2.1 Опрос полевых устройств и публикация телеметрии

Для обеспечения централизованного сбора технологических данных с устройств нижнего уровня, взаимодействующих по протоколу Modbus TCP, разработан и внедрен модуль опроса, функционирующий как независимый компонент программной среды. Полученные параметры оборудования передаются в транспортную подсистему системы диспетчеризации с использованием протокола MQTT [115, 116].

Функциональное разделение задач сбора телеметрии и последующих операций обработки, хранения и визуализации минимизирует связанность между источниками данных и конечными потребителями.

Modbus Reader реализован в виде сервера опроса, поддерживающего параллельное взаимодействие с множеством устройств. Программная реализация

алгоритма выполнена на языке программирования Python [117], зарекомендовавшего себя широкой распространенности языка в инженерных системах [118] и наличию необходимых сетевых и промышленно-ориентированных библиотек [111, 119, 120]. Управление процессом обмена обеспечивается специализированной библиотекой Modbus TCP собственной разработки [121, 122].

Реализованный механизм позволяет быстро добавлять новые устройства и воспроизводить конфигурацию при развертывании системы.

Пример структуры конфигурационного файла приведен на рисунке 30.

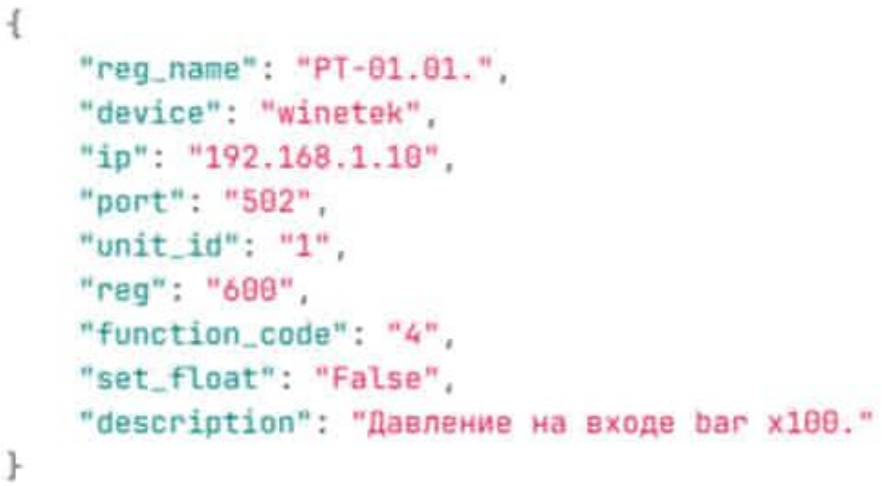

Рисунок 30 – Структура файла с заданием опроса устройств

Параметры опроса определяются описательной структурой в формате JSON [123], что ускоряет внесение изменений при изменении состава оборудования или перенастройке объекта. В конфигурационном файле для каждого устройства задаются IP-адрес, порт, unit ID, диапазон регистров и параметры интерпретации данных (тип, формат, масштаб и другие).

Программная архитектура модуля включает компоненты для выполнения опроса устройств по конфигурации, чтения регистров Modbus, хранения результатов обмена, преобразования данных для публикации и передачи сообщений в MQTT-брокер.

Алгоритм работы программы представлен на рисунке 31.

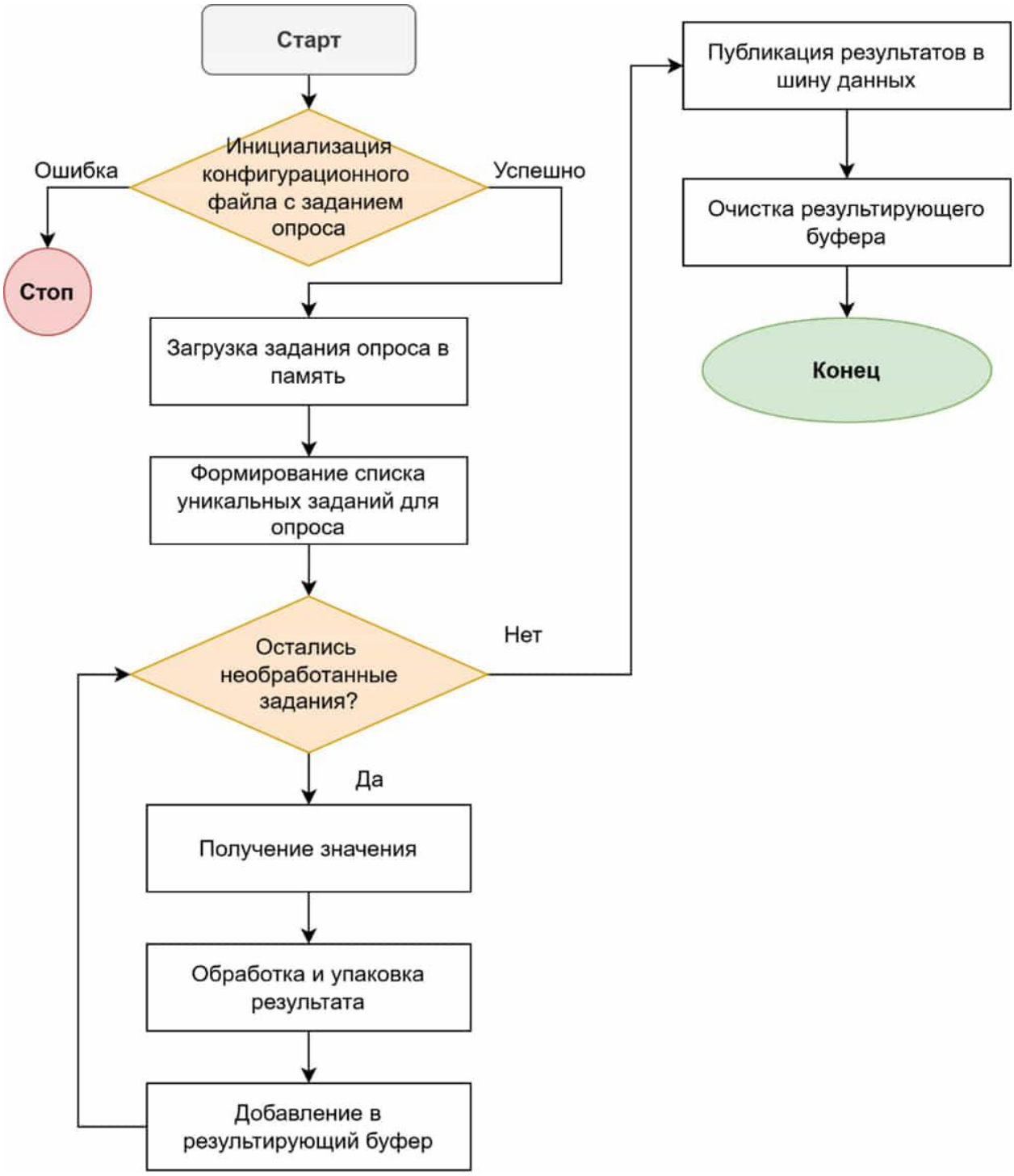

Рисунок 31 – Алгоритм работы программы Modbus Reader

После завершения цикла опроса данные структурируются и передаются в шину данных на базе брокера сообщений MQTT [116], которая предоставляет независимость управления потоков данными между компонентами системы, включая хранение, визуализацию, диагностику и управление [124].

3.2.2 Долговременное хранение данных телеметрии

В составе программной среды АСУТП имеется разработанный модуль ТопикЭкспортер, предназначенный для автоматизированного долговременного хранения технологических данных, поступающих от полевого уровня через транспортную единую шину данных [125, 126]. Данный модуль обеспечивает прием, нормализацию и запись телеметрических сообщений в централизованное реляционное хранилище на базе системы управления базами данных (СУБД) PostgreSQL [126-128].

Объектовая шина MQTT аккумулирует телеметрию, поступающую от полевых устройств, опрошенных Modbus Reader (см. раздел 3.2.1). Каждое опубликованное сообщение отражает текущее состояние одного или нескольких регистров контроллера. Для обеспечения последующего анализа, построения трендов и формирования отчетности, поступающие значения архивируются в СУБД [129].

Архитектура модуля строится по принципу разделения интерфейсов, допуская адаптацию под альтернативные источники данных и различные системы хранения без модификации основной логики.

Логика работы программы последовательная:

1. получение параметров подключения;
2. инициализация соединения с шиной данных и СУБД PostgreSQL;
3. автоматическая проверка и создание таблиц хранения на основе типизированной модели [130];
4. чтение данных в режиме непрерывного ожидания сообщений.

В компоненте реализованы механизмы автоматического восстановления соединений и идемпотентной инициализации структуры таблиц, обеспечивающие безопасный повторный запуск модуля без нарушения целостности архива.

Совокупность работы программного модуля и реляционной СУБД PostgreSQL формируют базу исторических данных для последующего построения

аналитических модулей, автоматической отчетности и аудита технологических процессов.

Экранная форма на рисунке 32 иллюстрирует процесс хранения и организации данных телеметрии в реляционной СУБД, отражая последовательное накопление параметров технологических процессов с фиксацией времени и идентификаторов.

	id [PK] integer	timestamp timestamp without time zone	reg_name text	data double precision
1	1	2025-05-14 13:32:47.74795	s2p_pump1_freq	44.3
2	2	2025-05-14 13:32:47.747966	s2p_pump2_I	57.6
3	3	2025-05-14 13:32:47.747972	s2p_rm5_2_rate	71
4	4	2025-05-14 13:32:47.747977	s2p_p_out	3.859375
5	5	2025-05-14 13:32:47.747982	gss_skv_2_I	66.6
6	6	2025-05-14 13:32:47.747986	gss_skv_2_level	82.3
7	7	2025-05-14 13:32:47.74799	ion_chlor	328.1116027832031
8	8	2025-05-14 13:32:47.747998	temp_chlor	12.425025939941406
9	9	2025-05-14 13:32:47.748004	gss_rahod_vchas2	56.567405700683594
10	10	2025-05-14 13:32:47.748009	gss_osmos_pe	2.957399845123291
11	11	2025-05-14 13:32:47.748013	gss_osmos_pe1	1.2740000486373901
12	12	2025-05-14 13:32:47.748016	gss_osmos_pe4	8.65625
13	13	2025-05-14 13:32:47.74802	gss_elektro	320
14	14	2025-05-14 13:32:47.748024	rm5_rashod	124.75
15	15	2025-05-14 13:32:47.748027	concentrat1_300	14.931866645812988
16	16	2025-05-14 13:32:47.74803	concentrat2_300	16.656116485595703
17	17	2025-05-14 13:32:47.748034	aks_elektro2	7
18	18	2025-05-14 13:32:48.777434	s2p_pump2_freq	45.2
19	19	2025-05-14 13:32:48.777449	gss_skv_3_I	66.7
20	20	2025-05-14 13:32:48.777454	gss_skv_1_level	82.4
21	21	2025-05-14 13:32:48.777457	gss_skv_3_level	83.1
22	22	2025-05-14 13:32:48.777461	gss_rahod_vchas3	56.32235336303711
23	23	2025-05-14 13:32:48.777464	gss_osmos_pe1	1.2860000133514404
24	24	2025-05-14 13:32:48.777467	gss_osmos_pe2	14.325000762939453
25	25	2025-05-14 13:32:48.777471	gss_elektro	297.5

Рисунок 32 – Демонстрация записи архивируемых данных телеметрии в системе долговременного хранения

Алгоритм работы программы представлен на рисунке 33.

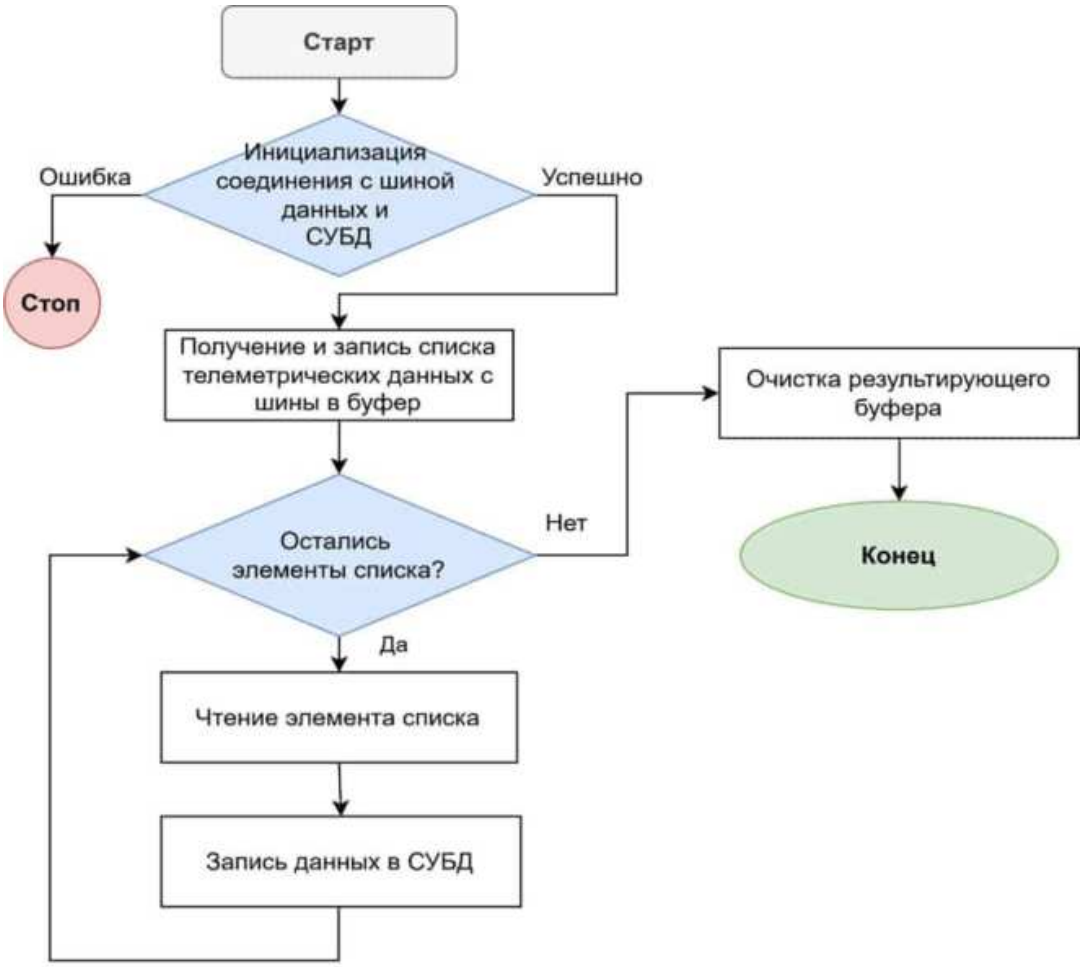

Рисунок 33 – Алгоритм работы программы ТопикЭкспортер

Механизм обработки сообщений использует буферизацию, обеспечивая устойчивость к временным задержкам и снижая риск потери данных при пиковых нагрузках. Каждое сообщение декодируется, проверяется и сохраняется в реляционной СУБД с фиксацией времени и техническим статусом связи.

3.2.3 Визуализация параметров в интерфейсе оператора

В составе программной среды АСУТП реализован компонент, обеспечивающий визуализацию текущего состояния технологического объекта посредством современного веб-интерфейса [131]. Реализация основана на приеме

телеметрических сообщений по шине MQTT, последующей трансформации данных для отображения [132] и предоставлении информации операторам через защищенное HTTPS-соединение [133].

Данные, поступающие в систему, агрегируются на сервере, проходят обработку и становятся доступны для отображения в интерактивных панелях оператора. Также реализована поддержка графического отображения динамики параметров с возможностью анализа истории изменений и наблюдения за трендами посредством интеграции с внешними средствами визуализации.

Механизм оперативного реагирования реализован с помощью аудиосигналов, информирующих персонал при наступлении нештатных условий или отклонений параметров от допустимых значений [134]. В системе реализовано логическое разделение задач обработки телеметрии на сервере и формирования интерфейса на клиентской стороне, что позволяет обеспечить высокую производительность и гибкую адаптацию под различные эксплуатационные сценарии [53].

Передача данных между компонентами организована посредством стандартизированных протоколов, что обеспечивает совместимость с другими подсистемами АСУТП и облегчает расширение функциональности программной среды. Экранная форма интерфейса панели оператора представлена на рисунке 34, где показано визуальное представление актуальных параметров.

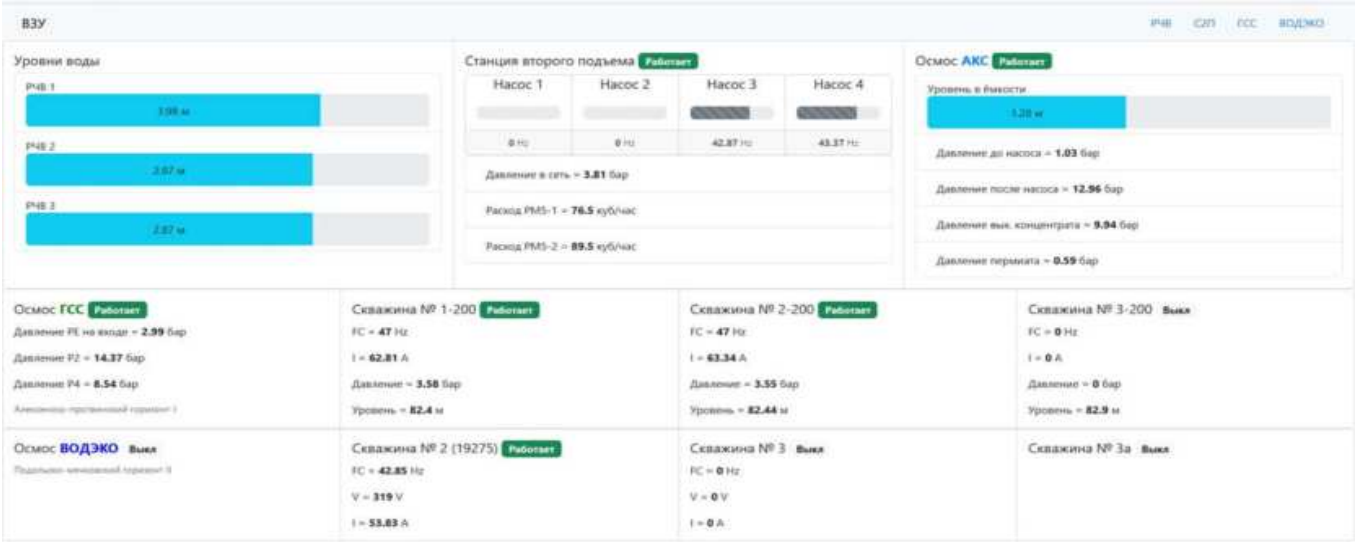

Рисунок 34 – Экранная форма интерфейса панели оператора

Пользовательский интерфейс панели оператора обеспечивает наглядное и удобное отображение параметров, поддерживает мониторинг состояния технологического объекта и своевременное информирование персонала о критических событиях. В реализованной системе предусмотрена возможность отображения как текущих значений параметров, так и исторических данных, что существенно повышает информативность и позволяет оператору проводить анализ динамики изменений технологических процессов.

На рисунке 35 приведен пример отображения графиков исторических данных.

Рисунок 35 – Экранная форма панели оператора с отображением графиков параметров технологического объекта водозаборного узла (ВЗУ)

Данная форма иллюстрирует возможность анализа трендов и облегчает принятие обоснованных решений в процессе эксплуатации объекта.

3.3. Реализация системы автоматизированного сопровождения конфигурации программной среды АСУТП

Для решения задач управления конфигурацией программной среды целесообразно использование инструментальных средств автоматизации, получивших широкое распространение в области ИТ [25, 27]. К числу наиболее востребованных средств данного класса относятся Terraform и Ansible [110].

В качестве исполнительного механизма для доставки и выполнения сформированных корректирующих воздействий используется программное средство Ansible, рассматриваемое в качестве исполнительного механизма доставки и применения корректирующих воздействий на уровне программной среды.

В рамках реализованной системы область конфигурационного контроля ограничена параметрами, поддающимися формализованному описанию в $S_{\text{цел}}$ и автоматизированному воспроизведению в $S(t)$.

В рассматриваемой реализации в область контроля включены: состав прикладных компонентов и их версии/идентификаторы; параметры запуска и размещения (контейнерные образы, переменные окружения, тома, каталоги); параметры сетевого взаимодействия (порты, адреса, маршрутизация, настройки доступа); параметры файловой системы, влияющие на размещение данных (тип ФС, монтирование, inode, права доступа); параметры прикладного диспетчерского проекта (состав, активные элементы). Параметры, не имеющие регламентированного машинно-интерпретируемого представления, в область контроля не включаются [25].

3.3.1 Параметризация нормативной конфигурации компонентов

В процессе автоматизации установки программных компонентов среды исполнения используется система логически изолированных ролей Ansible,

описанных в playbook-файле, который реализует и кодирует нормативное описание программной среды ($S_{\text{цел}}$).

Каждая роль включает структуру шаблонов, описание задач, переменные по умолчанию и вычисляемые значения. Наиболее значимым элементом является шаблон YAML-фрагмента, предназначенный для включения в итоговую конфигурацию среды исполнения на базе манифеста docker-compose [135]. В шаблоне фиксируются параметры запуска контейнера, в том числе путь к образу, значения переменных окружения, параметры томов, сетевые настройки и параметры контроля состояния.

Гибкость достигается с помощью трехуровневой параметризации на базе механизмов Ansible. Значения $default\_*$ формируют типовое поведение роли, параметры $override\_*$ могут быть заданы при вызове роли из управляющего сценария [136] и служат для переопределения базовых значений, а параметры $local\_*$ вычисляются в процессе исполнения роли при помощи фильтра $default()$, что обеспечивает приоритет и корректное наследование параметров из разных источников. На рисунке 36 приведен пример вычисления переменной внутри роли.

```
local_a:
  image: "{{ override_a.image default(default_a.image) }}"
```

Рисунок 36 – Расчет переменной

Механизм трехуровневой параметризации позволяет использовать единую структуру роли для конфигурирования компонентов в различных эксплуатационных режимах. Все вычисления выполняются на уровне самой роли, а управляющий сценарий определяет только те значения, которые должны отличаться от стандартных.

Процесс генерации начинается с получения актуальных значений переменных, их подстановки в шаблон части манифеста docker-compose – service.yml.j2 и формирования YAML-файла с параметрами сервиса. Шаблонизация

реализуется с применением встроенного модуля template [27], пример работы которого представлен на рисунке 37.

```
- name: generate service "{{ local_a.service_name }}"
  template:
    src: "service.yml.j2"
    dest: "{{ AGGREGATOR_TEMP_FOLDER }}/docker-compose-{{ local_a.service_name }}"
```

Рисунок 37 – Формирование описательного манифеста сервиса

При отсутствии обязательных параметров генерация шаблона немедленно останавливается с выдачей диагностического сообщения, исключая появление некорректных конфигураций.

После формирования всех фрагментов, подготовленных компонентными ролями, с помощью агрегационной роли выполняется склеивание из сгенерированных описаний элементов общего манифеста service.yml.j2 в итоговый манифест docker-compose. Итоговый файл сохраняется в целевом каталоге и обеспечивает запуск контейнеризированных компонентов среды исполнения.

Общая схема генерации и агрегации конфигурации приведена на рисунке 38.

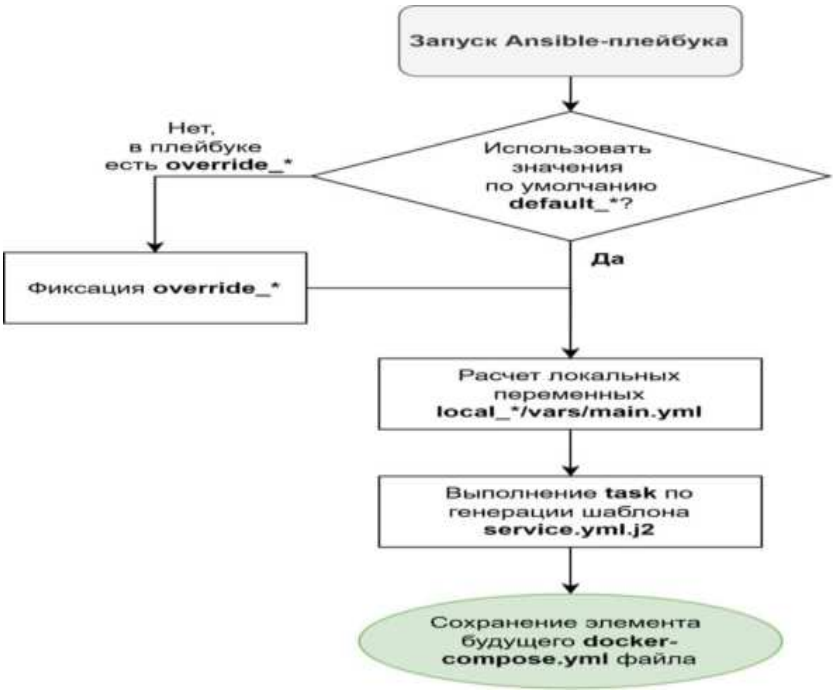

Рисунок 38 – Схема генерации конфигурации описания программной среды

Благодаря централизованному описанию переменных, шаблонному механизму генерации и отказу от ручных правок, процесс формирования конфигурации является детерминированным – при одинаковом наборе входных данных повторный запуск исполняемого сценария всегда приводит к идентичному результату.

Пример вызова роли с переопределенными параметрами компонента modbus_reader показан на рисунке 39, итоговый фрагмент конфигурации – на рисунке 40.

```
- role: prepare_service_modbus_reader
  vars:
    override_prepare_service_modbus_reader:
      docker_image: "{{ docker_registry }}/aks5612403/services/modbus_reader:feature-timestamp"
      ENVIRONMENT:
        CONFIG_PATH: "configs_modbus_reader/emulator_tomilino"
        REGS_FILE: "tomilino.json"
        MQTT_USER: "{{ credentials_mqtt.login }}"
        MQTT_PASSWORD: "{{ credentials_mqtt.password }}"
        MQTT_TOPIC: "{{ mqtt_topics.topic_1 }}"
      di:
        mqtt:
          MQTT_BROKER: "{{ server_ip }}"
          MQTT_PORT: "{{ intercomm_mqtt_server.api.external_port1 }}"
```

Рисунок 39 – Вызов роли и переопределение переменных в Ansible-Playbook

```
version: '3.7'
services:
  modbus_reader:
    image: petrovi4corp.space/aks5612403/services/modbus_reader:feature-timestamp
    restart: always
    volumes:
      - /opt/services/tomilino_test/pull_servers/modbus_reader_EXPERIMENTAL/data:/modbus_reader/data/
      - /opt/services/tomilino_test/pull_servers/modbus_reader_EXPERIMENTAL/logs:/modbus_reader/logs/
    ports:
      - "10013:3000"
    environment: <6 items>
    healthcheck: <5 keys>
    networks:
      - pull_servers_experimental-backend-main
    ## BEGIN of ansible SWARM EOF deploy configuration
    logging:
      driver: json-file
networks:
  pull_servers_experimental-frontend-main:
    name: pull_servers_experimental-frontend-main
  pull_servers_experimental-backend-main:
    name: pull_servers_experimental-backend-main
```

Рисунок 40 – Пример итогового docker-compose-фрагмента

Таким образом, параметризованное описание ролей Ansible и шаблонов конфигурации образует машинно-интерпретируемое представление нормативной конфигурации программной среды $S_{\text{цел}}$ в терминах, введенных в разделах 2.1 и 2.2. Совокупность параметров ролей, шаблонов и вычисляемых значений однозначно определяет состав программных компонентов, их параметры и условия межкомпонентного взаимодействия в заданной области конфигурационного контроля.

3.3.2 Согласование параметров взаимодействия компонентов

Когда программная среда АСУТП представлена в виде совокупности связанных компонентов (см. раздел 1.3.2), необходимо учитывать, что взаимодействие между ними осуществляется через протоколы обмена данными. Эти параметры зависят от конкретной схемы установки и могут изменяться при переносе или повторной сборке системы. Статическая фиксация таких параметров в теле конфигурации создает жесткую связанность, приводит к дублированию информации и ограничивает возможность гибкого масштабирования.

При вынесении параметров взаимодействия в переменные, назначаемые вручную, сохраняется риск несогласованности значений между источником и получателем. Так, при изменении имени узла, порта или префикса API необходимо обеспечить актуализацию не только в роли компонента-источника, но и во всех ролях, в которых эти параметры используются. Подобная зависимость становится фактором повышенной вероятности ошибок и некорректной работы системы.

В настоящей работе разработан механизм, обеспечивающий автоматическое согласование параметров интерфейсов между компонентами [102]. Он позволяет передавать значения, определяемые компонентом-источником, в шаблоны конфигурации зависимых компонентов без их дублирования или ручного вмешательства. Механизм получил обозначение intercomm и реализуется в каждой

Ansible-роли, формирующей внешний интерфейс. Пример структуры intercomm_mqtt_server, экспортируемой из роли, приведен на рисунке 41.

```
intercomm_mqtt_server:
  api:
    domain: "{{ local_prepare_service_mqtt_server.service_name }}"
    external_port1: "{{ local_prepare_service_mqtt_server.ports.api.external1 }}"
    external_port2: "{{ local_prepare_service_mqtt_server.ports.api.external2 }}"
```

Рисунок 41 – Образец структуры intercomm, формируемой в роли ведущего сервиса (MQTT Server)

Расчет значений, включаемых в intercomm, производится внутри роли на основании трехуровневой модели параметризации. Параметры local_* вычисляются на основе входных $override\_*$ и значений по умолчанию $default\_*$. Таким образом, итоговая структура intercomm формируется детерминировано, в рамках исполнения роли, и отражает актуальное состояние экспортируемого интерфейса.

Компоненты, использующие внешний интерфейс другого компонента, получают соответствующие параметры через структуру переменных di (dependency injection, зависимые инъекции) [137], передаваемую при вызове роли из исполняемого сценария. Пример структуры di, внедряемой в компонент визуализации HMI, представлен на рисунке 42.

```
default_prepare_service_hmi:

  di:
    mqtt:
      MQTT_BROKER: "mosquitto"
      MQTT_PORT: "1883"
```

Рисунок 42 – Пример структуры переменных DI в роли зависимого компонента.

Параметры, определенные в структуре intercomm одного компонента, автоматически внедряются в структуру di зависимого компонента, что отражено на схеме взаимодействия ролей (рисунок 43).

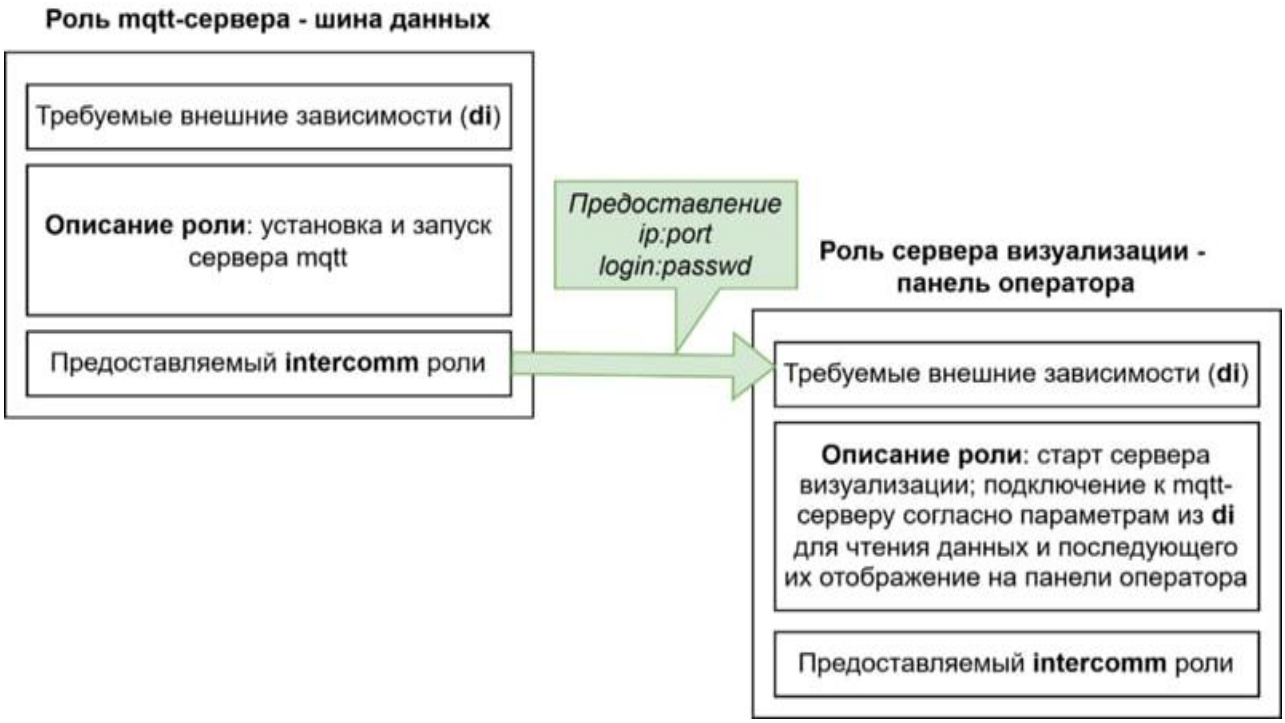

Рисунок 43 – Схема взаимодействия между ролями через intercomm и DI

В процессе развертывания исполняемый сценарий подставляет сформированную структуру intercomm в соответствующую переменную di, обеспечивая тем самым согласованное подключение и актуальность передаваемых значений. Формирование структуры intercomm осуществляется в режиме run-time при запуске ansible-playbook.

Роль зависимого компонента использует полученные через di значения для шаблонизации конфигурационного файла, автоматически подставляя их в итоговое YAML-описание среды исполнения. Такой механизм гарантирует, что, например, параметры MQTT_BROKER и MQTT_PORT всегда соответствуют актуальному состоянию интерфейса MQTT-сервера и не требуют ручной синхронизации в различных частях системы.

Разработанный механизм intercomm обеспечивает формализованное представление параметров межкомпонентных интерфейсов и исключает их дублирование в нормативном описании $S_{\text{цел}}$. Тем самым параметры связей между элементами программной среды задаются детерминированно и учитываются при сопоставлении $S(t)$ и $S_{\text{цел}}$ в составе вектора отклонений $e(t)$.

3.3.3 Получение фактической конфигурации программной среды

Получение фактической конфигурации программной среды $S(t)$ осуществляется в соответствии с моделью, принятой в настоящей работе и описанной в разделе 2.1. Перечень извлекаемых параметров определяется нормативным описанием $S_{\text{цел}}$ в пределах области конфигурационного контроля.

Как было указано в разделе 3.1.2, в качестве операционной системы используется Linux. В связи с этим последующее формирование $S(t)$ в части системного ПО опирается на стандартные механизмы и инструментальные средства ОС семейства Linux, а также на регламентированное размещение системных компонентов и конфигурационных файлов.

Пример получения информации о версии и дистрибутиве операционной системы приведен на рисунке 44, а настроек сети – на рисунке 45.

Рисунок 44 – Получение сведений о версии и дистрибутиве ОС

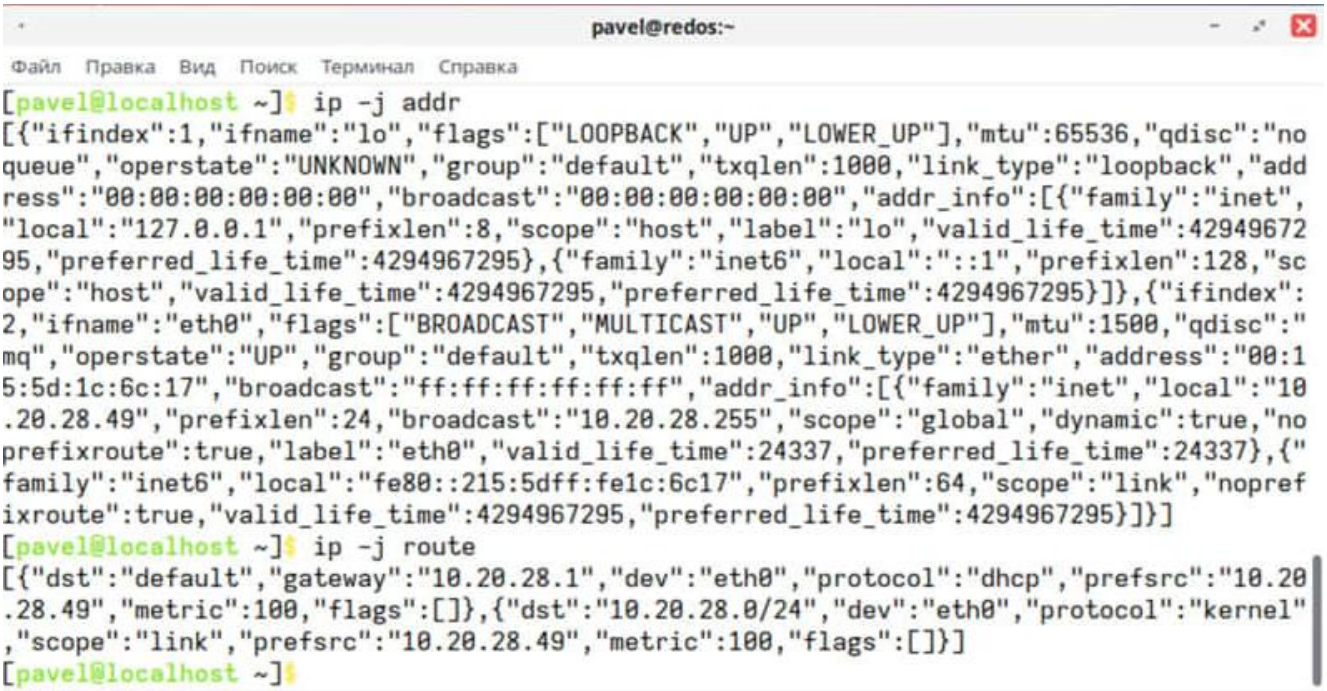

Рисунок 45 – Получение параметров сетевой конфигурации ОС

Параметры файловой системы (ФС) не ограничиваются сведениями о доступном дисковом пространстве и включают совокупность конфигурационных ограничений и структурных характеристик, таких как тип ФС, параметры монтирования, количество и использование индексных дескрипторов (inode), а также права доступа к рабочим каталогам [25] (рисунок 46).

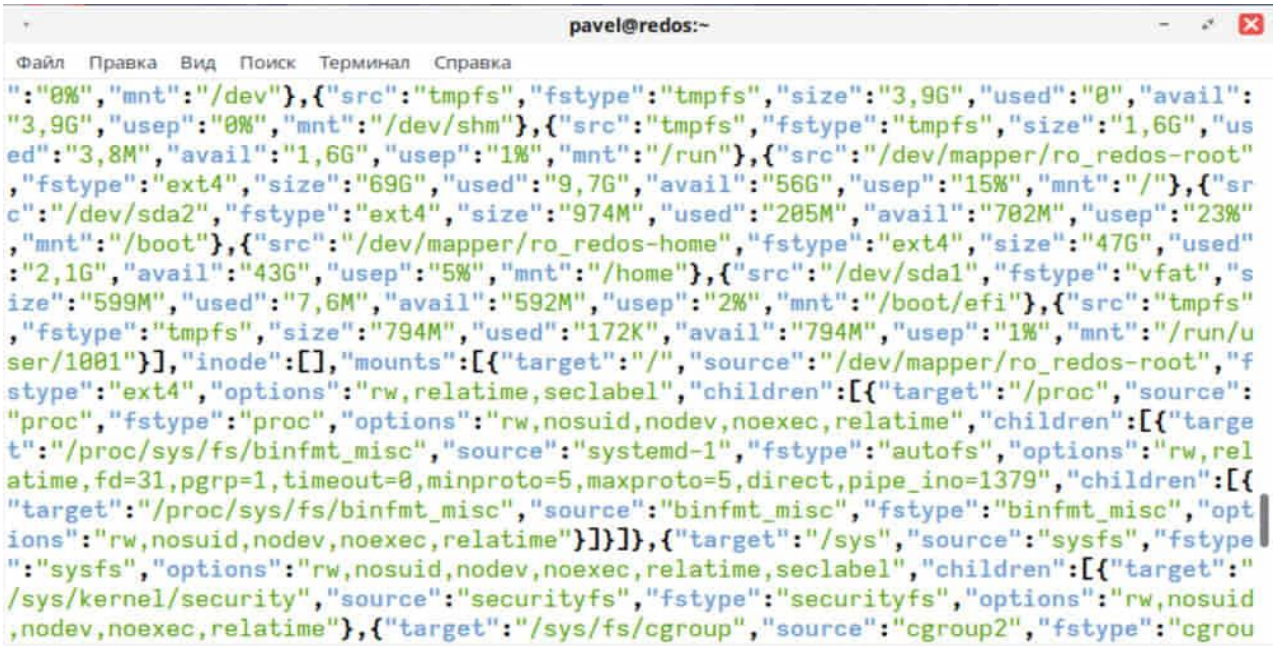

Рисунок 46 – Получение параметров ФС ОС

Указанные параметры являются взаимосвязанными и оказывают непосредственное влияние на возможность размещения и корректного функционирования компонентов программной среды. В частности, исчерпание доступных inode при формальном наличии свободного дискового пространства приводит к нарушению функционирования прикладных компонентов при отсутствии явных диагностических признаков отказа.

Формирование $S(t)$ прикладной части программной среды предполагает получение сведений о составе задействованных программных компонентов, их версиях, параметрах функционирования и реализуемых диспетчерских проектах в объеме, определяемом нормативным конфигурационным описанием $S_{\text{цел}}$.

На этапе проектирования и разработки программных компонентов системы диспетчеризации было принято архитектурное решение о включении в состав каждого компонента программного интерфейса REST API [54], предназначенного для предоставления служебной эксплуатационной информации. Указанный интерфейс реализуется как часть штатной функциональности компонентов и используется для получения параметров, характеризующих конфигурацию программной среды.

К таким параметрам относятся показатели работоспособности компонента (healthcheck), номер используемого сетевого порта, версия программного обеспечения, идентификатор сборки, а также параметры конфигурационного состава и прикладного проекта, реализуемого данным программным компонентом [25].

Формирование $S(t)$ начинается с определения состава запущенного прикладного программного обеспечения, включая идентификацию программных компонентов и их версий. Пример выполнения запроса для получения информации о версии программного компонента интерфейса оператора приведен на рисунке 47.

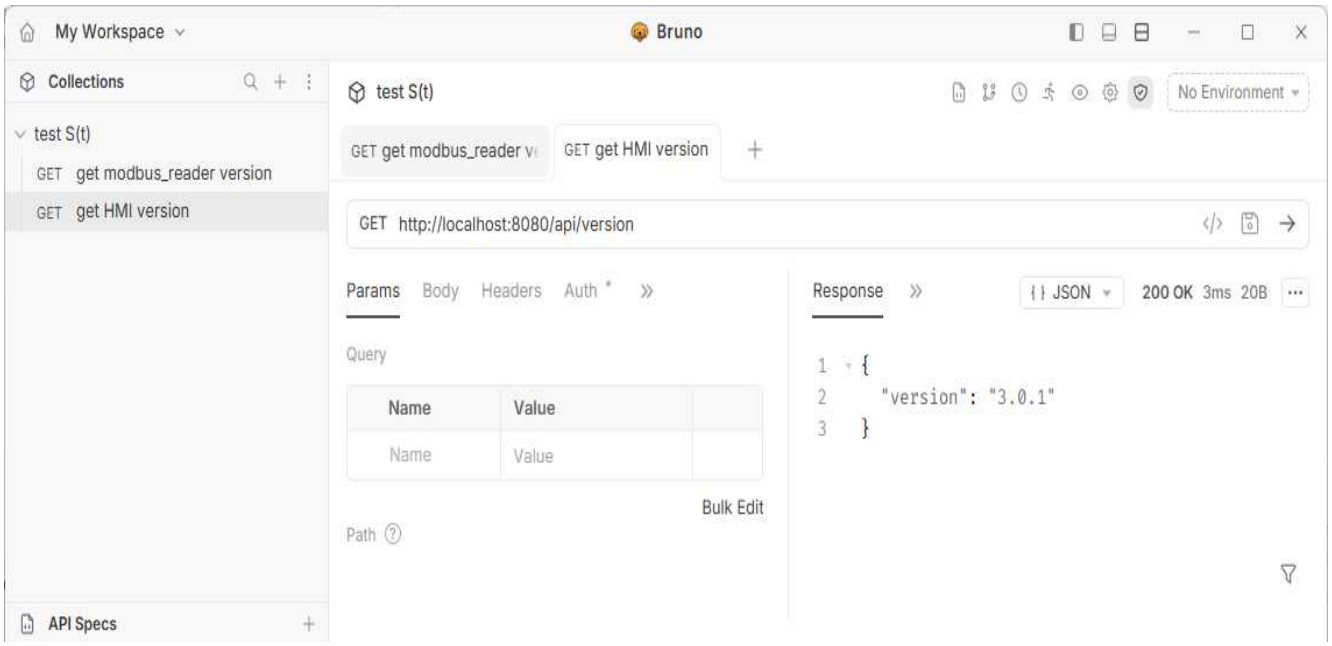

Рисунок 47 – Проверка возможности получения версии компонента HMI

На следующем этапе формирования $S(t)$ посредством программного интерфейса извлекаются сведения о прикладном SCADA-проекте, отображаемом в панели оператора, включая информацию о его составе и активных элементах. Примеры получения информации о SCADA-проекте и результаты запроса об активных тегах приведены на рисунках 48 и 49.

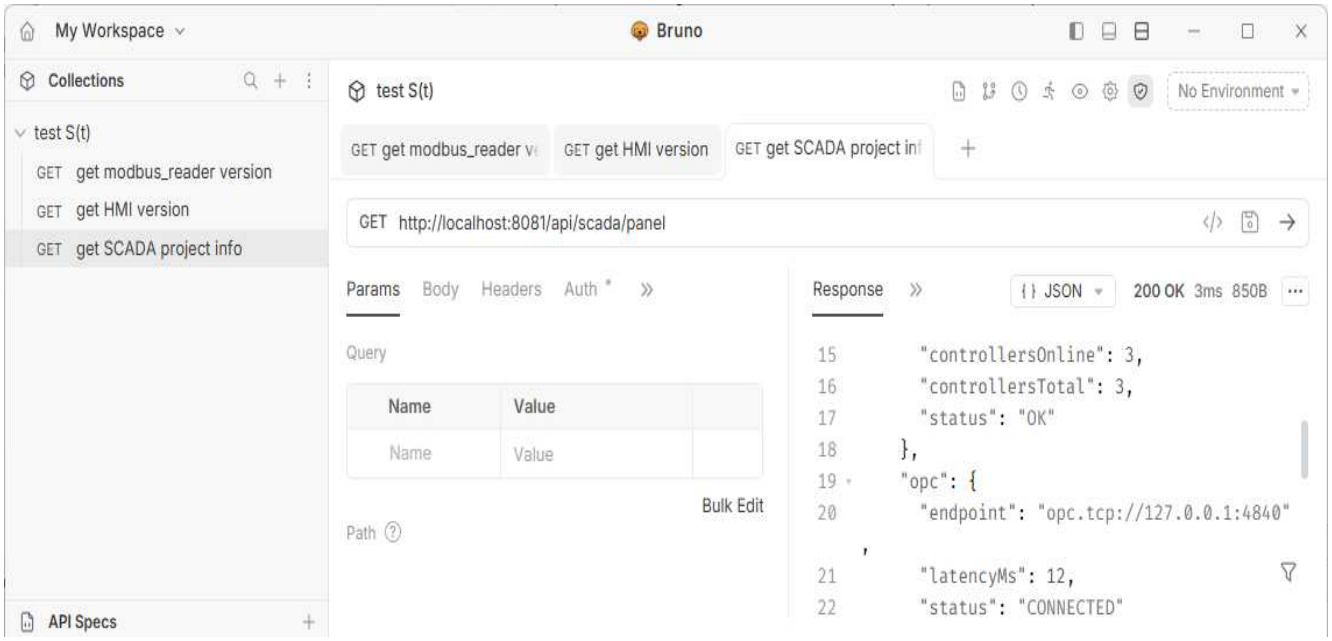

Рисунок 48 – Получение информации о SCADA проекте

```
{
  "count": 9,
  "tags": [
    {
      "objectId": "P-101",
      "quality": "GOOD",
      "tagId": "P101_RUN",
      "unit": "",
      "updatedAt": "2025-03-13T14:36:11.309759+00:00",
      "value": true
    },
    {"objectId": "P-101"...},
    {"objectId": "P-101"...},
    {"objectId": "P-102"...},
    {"objectId": "P-102"...},
    {"objectId": "P-102"...},
    {"objectId": "V-201"...},
    {"objectId": "LT-301"...},
    {"objectId": "FT-401"...}
  ],
  "ts": "2025-03-13T14:36:11.309759+00:00"
}
```


Рисунок 49 – Результат запроса об активных тегах в SCADA-проекте

Полученные данные агрегируются в структурированное представление, соответствующее формату модели фактической конфигурации, заданной выражением (2.2). В результате формируется рекурсивная модель, отражающая состав программных компонентов, значения их параметров и структуру взаимосвязей между ними.

Упрощенное представление результата формирования фактической конфигурации S(t) приведено на рисунке 50.

Полученные на системном и прикладном уровнях параметры формализуются в виде элементов модели $S(t)$, что обеспечивает их сопоставимость с нормативным конфигурационным описанием $S_{\text{цел}}$ и возможность автоматизированной проверки.

```yaml
---
- name: Build formal S(t) model
  hosts: scada_nodes
  gather_facts: true
  become: true

  vars:
    snapshot_dir: "/var/tmp/scada_snapshot"
    snapshot_file: "{{ snapshot_dir }}/S_t_{{ inventory_hostname }}.json"

    S_target_components:
      - id: "hmi"
        type: "SCADA_HMI"
      - id: "modbus_reader"
        type: "SERVICE"

    components:
      - id: "hmi"
        type: "SCADA_HMI"
        base_url: "http://10.8.0.15:8081/api/scada"
        public:
          version: "/project"
          health: "/diagnostics"
        protected:
          project: "/panel"
          tags: "/tags?limit=200"
        listen_ports: [8081]
      - <6 keys>
      - <6 keys>
      - <6 keys>
      - <6 keys>
      - <6 keys>

    allowed_sources:
      - "api_public"
      - "api_protected"
      - "cli"
      - "procfs"
      - "ss"
```

Рисунок 50 – Результат получения фактической конфигурации $S(t)$

Сформированное представление $S(t)$ используется системой сопровождения в качестве входных данных для процедуры формализованного сопоставления $S(t)$ и $S_{\text{цел}}$, реализующей оператор δ, введенный в разделе 2.2. Результатом сопоставления является формирование вектора конфигурационных отклонений $e(t)$, который далее используется для вычисления совокупности корректирующих воздействий ΔS.

3.4 Формирование воспроизводимых сред исполнения

3.4.1 Подготовка контейнерных компонентов

Реализация требований к стандартизированной среде исполнения, сформулированных в разделе 2.3 и направленных на упрощение формирования $S(t)$, обеспечивается с использованием сборочного окружения на базе Buildah.

Все процессы подготовки контейнерных компонентов определены – проводится верификация исходных файлов, анализ структуры и безопасности, автоматизированная сборка и публикация результатов в централизованный реестр (docker registry). Сборка реализована в системе CI/CD с применением параметризованных описаний, где используются переменные версий компонентов, идентификаторы коммитов и параметры инфраструктуры. Уникальные теги, присваиваемые каждому компоненту, отражают вариант исполнения и параметры сборки.

На рисунке 51 представлена блок-схема алгоритма автоматизированной сборки, тестирования и публикации контейнерных компонентов программной среды АСУТП.

Корректность и полнота образа подтверждаются автоматизированным тестированием, в процессе которого проверяется структура, целостность, наличие всех зависимостей и отсутствие уязвимостей.

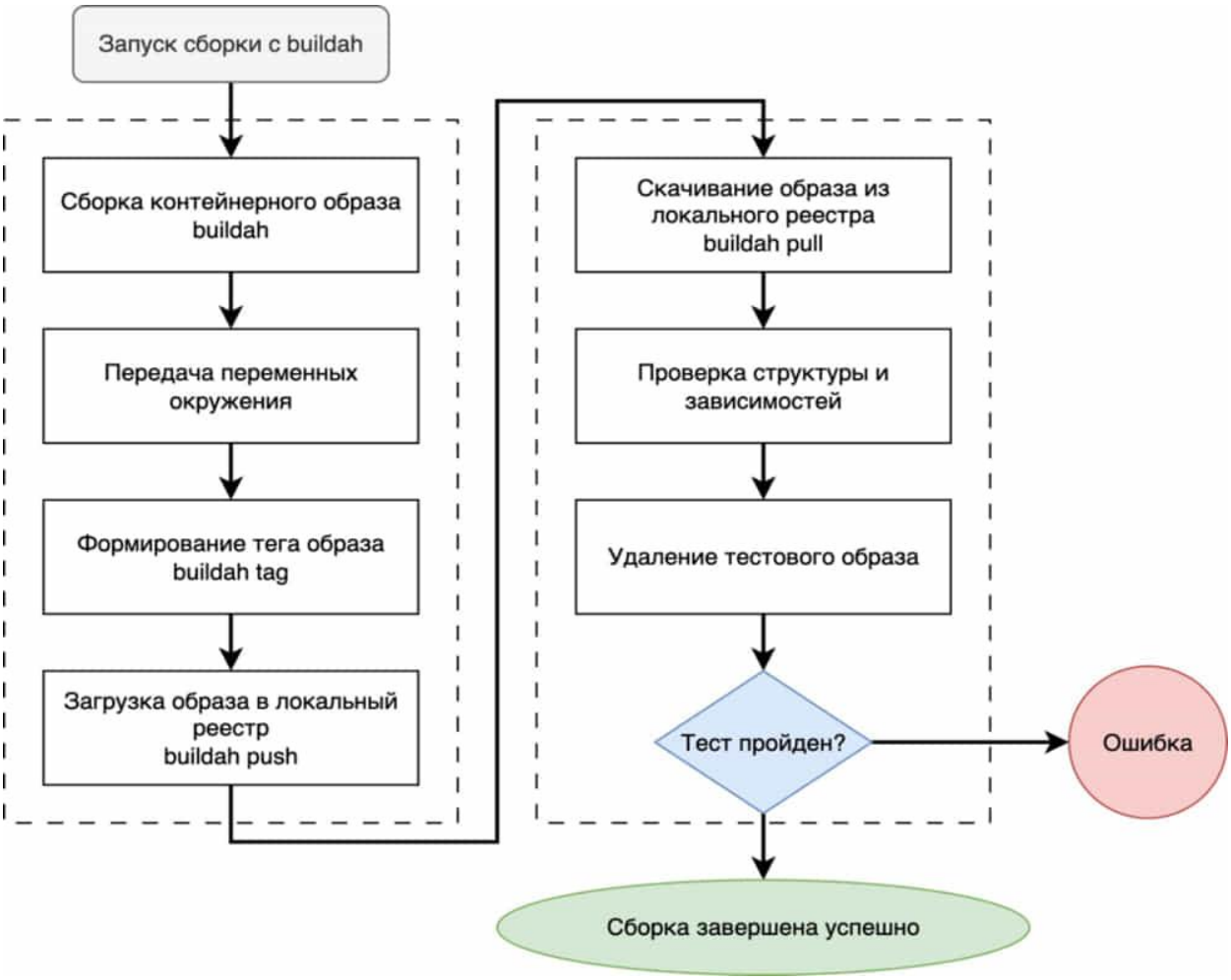

Рисунок 51 – Алгоритм автоматизированной сборки компонентов АСУТП

Публикация контейнера осуществляется только после завершения проверки.

Этап сборки выполняется с передачей параметров через переменные окружения. Для всех стадий используются централизованные внешние файлы переменных и шаблоны.

3.4.2 Контроль исходных данных для формирования образа

Контроль исходных данных при формировании контейнерного образа реализует модель стандартизированной среды исполнения, изложенные в разделе 2.3.2. Последовательность действий и используемые инструменты настраиваются в зависимости от инфраструктуры или особенностей программного обеспечения.

В рассматриваемой системе последовательная проверка исходных материалов формализует структуру, параметры и содержимое будущего образа. В ходе подготовки каждого компонента анализируется полный набор исходных данных, влияющих на состав и параметры формируемого образа.

Исходные программные компоненты, предназначенные для эксплуатации в программной среде АСУТП, размещаются в репозитории Gitlab [109]. В случае распространения программного обеспечения в скомпилированном (бинарном) виде, такие файлы подлежат отдельной маркировке как LFS-объекты [138].

Анализ сборочного манифеста (Dockerfile) занимает центральное место в процедуре проверки. Автоматизированная верификация выполняется с использованием инструмента hadolint [139], который сопоставляет содержимое манифеста с установленными в модели требованиями. Верификация включает проверку версии базового образа, оформление меток LABEL, применение многоступенчатой сборки, отсутствие устаревших или неиспользуемых инструкций, а также обязательное указание пользователя с ограниченными правами (USER) [140, 141].

В качестве исходного программного обеспечения выступает набор компонентов программной среды АСУТП, описанный в разделе 3.2.

Результаты автоматической проверки Dockerfile продемонстрированы на экранной форме рисунка 52. Здесь выявлены ошибки, связанные с отсутствием фиксации версий зависимостей и нарушающие воспроизводимость контейнера. В подобных случаях процесс сборки автоматически прерывается.

Если компонент программной среды АСУТП распространяется разработчиком в виде исходного кода, то на этапе подготовки допускается проведение дополнительного анализа средствами статического контроля, соответствующими языку программирования. Контроль выявляет ошибки структуры и стиля еще до формирования образа, способствуя прозрачности процедур верификации на ранних стадиях.

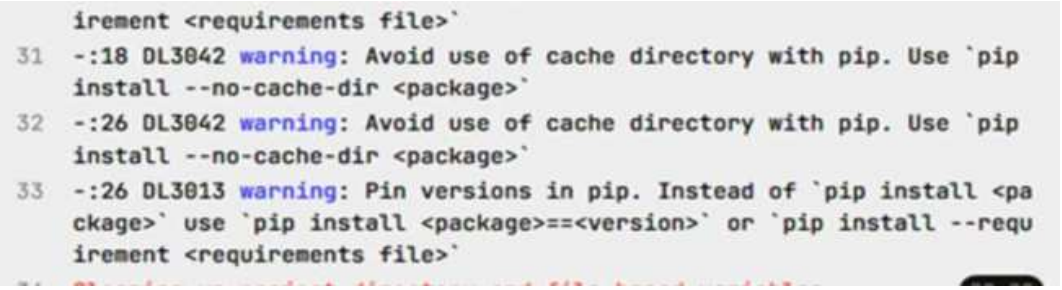

Рисунок 52 – Проверка структуры Dockerfile и выявление отклонений

Программное обеспечение, распространяемое исключительно в скомпилированном виде, не позволяет проводить анализ исходного кода без официального запроса к правообладателю. Поскольку такие продукты защищены режимом интеллектуальной собственности и, как правило, ограничены условиями лицензирования и режимом коммерческой тайны правообладателя, анализ исходного кода в типовых условиях эксплуатации не выполняется. В этих условиях контроль таких компонентов осуществляется по метаданным поставки (версия, контрольные суммы, цифровая подпись) и по результатам анализа сформированного контейнерного образа.

В качестве примера на экранной форме рисунка 53 приведен фрагмент отчета flake8 [142], зафиксировавшего неиспользуемые импорты, некорректные управляющие символы и ошибки форматирования в исходном коде тестовой программы на языке Python.

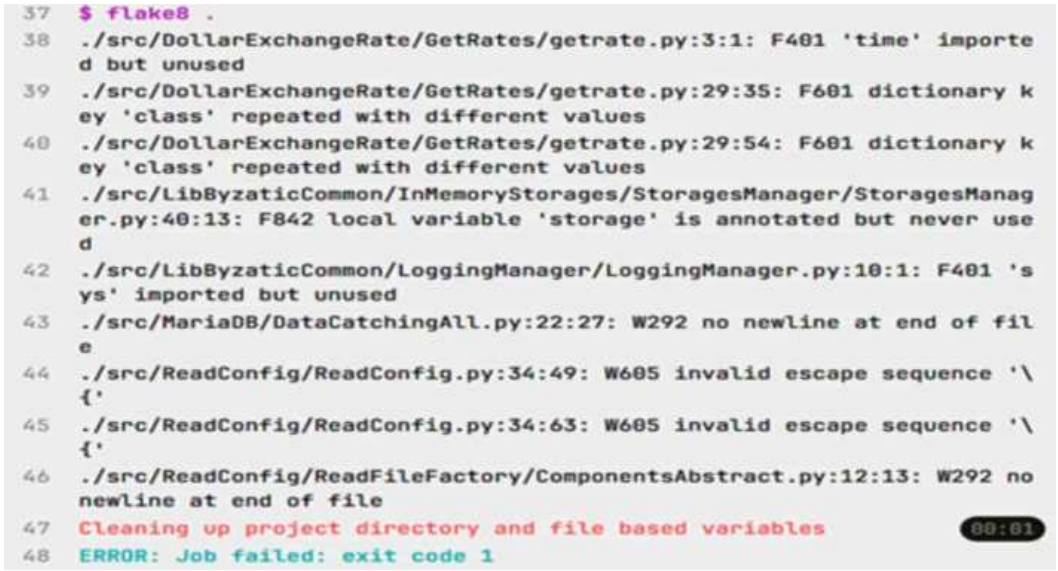

Рисунок 53 – Проверка кода компонента средствами flake8

Среда сопровождения позволяет в случаях, когда допускается сборка при наличии отдельных нарушений, настраивать допустимое поведение через конфигурацию средства контроля.

3.4.3 Проверка структуры и безопасности программных компонентов

Контроль структуры и безопасности программных компонентов на данном этапе ориентирован на исключение включения в контейнерный образ уязвимых либо некорректно сформированных модулей [12, 143]. Методика верификации определяется формой распространения программного компонента – в виде исходного кода или бинарного файла.

Если компонент формируется из исходного кода, применяется статический анализ с использованием инструментов, соответствующих выбранному языку программирования и архитектуре. В верифицируемой программной среде АСУТП, состоящей из разработанных компонентов на языке Python (см. раздел 3.2), используются flake8 для проверки структуры и форматирования, а также bandit [144] для выявления потенциальных уязвимостей.

В случаях, когда программный компонент распространяется только в скомпилированном виде, проводится проверка целостности и происхождения артефакта. Для этого в конвейере фиксируются контрольные суммы (к примеру, MD5 или SHA-256), осуществляется верификация источника получения файла, а также контроль соответствия его заявленной версии и идентификационным данным. При необходимости могут применяться дополнительные инструменты анализа внешних зависимостей и цифровой подписи файла, если это предусмотрено политикой безопасности проекта.

При любом варианте исполнения компонентов итоговый контейнерный образ подвергается анализу с использованием инструмента trivy [145], предназначенного для поиска уязвимостей, конфиденциальных данных [146], а также проверки корректности параметров сборки и исполнения.

На рисунке 54 показан результат анализа файловой системы будущего образа. В составе тестового ПО был обнаружен файл . *env* с конфиденциальными данными, включая четыре токена, три из которых классифицированы как критические. Система блокирует процесс сборки компонента до удаления этих данных.

```
┌──────────┬──────┬──────┬──────┐
│ .env     │ text │  -   │  4   │
└──────────┴──────┴──────┴──────┘
Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)
.env (secrets)
==============
Total: 4 (LOW: 0, MEDIUM: 0, HIGH: 1, CRITICAL: 3)
CRITICAL: GitHub (github-oauth)
──────────────────────────────────────────
GitHub OAuth Access Token
──────────────────────────────────────────
 .env:7
──────────────────────────────────────────
   5   # GitHub
   6   GITHUB_TOKEN=****************************************AB
   7 [ GH_TOKEN=****************************************
```

Рисунок 54 – Обнаружение конфиденциальных данных в файловой системе проекта

Рисунок 55 иллюстрирует проверку Dockerfile с тем же инструментом trivy. Отсутствие команды USER в инструкциях по сборке указывает на запуск контейнера с привилегированным доступом. Также зафиксировано отсутствие HEALTHCHECK, что затрудняет контроль его состояния. Эти нарушения классифицированы как значимые.

```
35 Dockerfile (dockerfile)
36 ======================
37 Tests: 28 (SUCCESSES: 26, FAILURES: 2)
38 Failures: 2 (UNKNOWN: 0, LOW: 1, MEDIUM: 0, HIGH: 1, CRITICAL: 0)
39 AVD-DS-0002 (HIGH): Specify at least 1 USER command in Dockerfile wit
   h non-root user as argument
40 ═══════════════════════════════════════
41 Running containers with 'root' user can lead to a container escape si
   tuation. It is a best practice to run containers as non-root users, w
   hich can be done by adding a 'USER' statement to the Dockerfile.
42 See https://avd.aquasec.com/misconfig/ds002
43 ──────────────────────────────────────
44 AVD-DS-0026 (LOW): Add HEALTHCHECK instruction in your Dockerfile
45 ═══════════════════════════════════════
46 You should add HEALTHCHECK instruction in your docker container image
   s to perform the health check on running containers.
47 See https://avd.aquasec.com/misconfig/ds026
```

Рисунок 55 – Проверка конфигурации Dockerfile с использованием trivy

Для тестового программного компонента, в котором осознанно допущена уязвимость, на рисунке 56 представлен результат анализа исходного кода с использованием bandit.

```
$ bandit -r .
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.11.12
Run started:2025-04-16 17:18:04.510880
Test results:
>> Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
   Severity: Medium   Confidence: Medium
   CWE: CWE-89 (https://cwe.mitre.org/data/definitions/89.html)
   More Info: https://bandit.readthedocs.io/en/1.8.3/plugins/b608_hardcoded_sql_expressions.html
   Location: ./main.py:51:20
50              start_date = end_date - timedelta(days=30)
51              cur.execute(f"SELECT * FROM {table} WHERE date BETWEEN %s AND %s", (start_date, end_date))
52              date_to_display = [start_date.strftime("%d-%m-%Y"), end_date.strftime("%d-%m-%Y")]
```

Рисунок 56 – Анализ Python-кода с использованием bandit и выявление потенциальной SQL-инъекции

Как видно, был явно выявлен участок, содержащий потенциальную SQL-инъекцию [147], эксплуатация которой способна привести к неблагоприятным последствиям. Среда сопровождения автоматически прервала выполнение сборки. До устранения данной уязвимости не будет собрано конвейеризированное программное обеспечения.

Однако, возможны допущения, когда декларативно прописывается спектр уязвимостей, чье наличие допускается в итоговом образе.

3.4.4 Анализ итогового образа

На заключительном этапе автоматизированного контроля выполняется стадия информационной безопасности (ИБ) на анализ содержимого сформированного контейнерного образа [148]. В исследуемой системе реализовано применение

инструмента trivy, выполняющего сканирование всех слоев итогового контейнерного образа. Такая проверка позволяет выявлять уязвимости [149] и утечки конфиденциальных данных как в базовых слоях, например, системных библиотеках, так и среди программных компонентов, добавленных в процессе сборки [150]. Анализ охватывает все элементы, включая компоненты базового образа, сторонние библиотеки и собственные модули [151].

В качестве примера на рисунке 57 приведен фрагмент отчета trivy, обнаружившего уязвимость CVE-2025-29087 [152] в пакете sqlite-libs (версия 3.48.0-r0), относящейся к категории серьезных (high). В результате анализа инструмент зафиксировал необходимость обновления до версии 3.48.0-r1 и предоставил ссылку на описание проблемы.

```
Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)
/image.tar (alpine 3.21.3)
==========================
Total: 1 (LOW: 0, MEDIUM: 0, HIGH: 1, CRITICAL: 0)

┌─────────────┬─────────────────┬──────────┬────────┬───────────────────┬───────────────┐
│   Library   │  Vulnerability  │ Severity │ Status │ Installed Version │ Fixed Version │
├─────────────┼─────────────────┼──────────┼────────┼───────────────────┼───────────────┤
│ sqlite-libs │ CVE-2025-29087  │ HIGH     │ fixed  │ 3.48.0-r0         │ 3.48.0-r1     │
└─────────────┴─────────────────┴──────────┴────────┴───────────────────┴───────────────┘
```

Рисунок 57 – Пример отчета trivy с обнаружением уязвимости в составе контейнерного образа

Применение единого сборочного конвейера фиксирует результаты контроля, а примеры на рисунке 58 подтверждают последовательное выполнение этапов верификации и сборки в среде GitLab CI/CD.

Реализация многостадийного процесса подготовки компонентов на основе автоматизированных процедур контроля обеспечивает структурную целостность и безопасность среды исполнения прикладного программного обеспечения АСУТП.

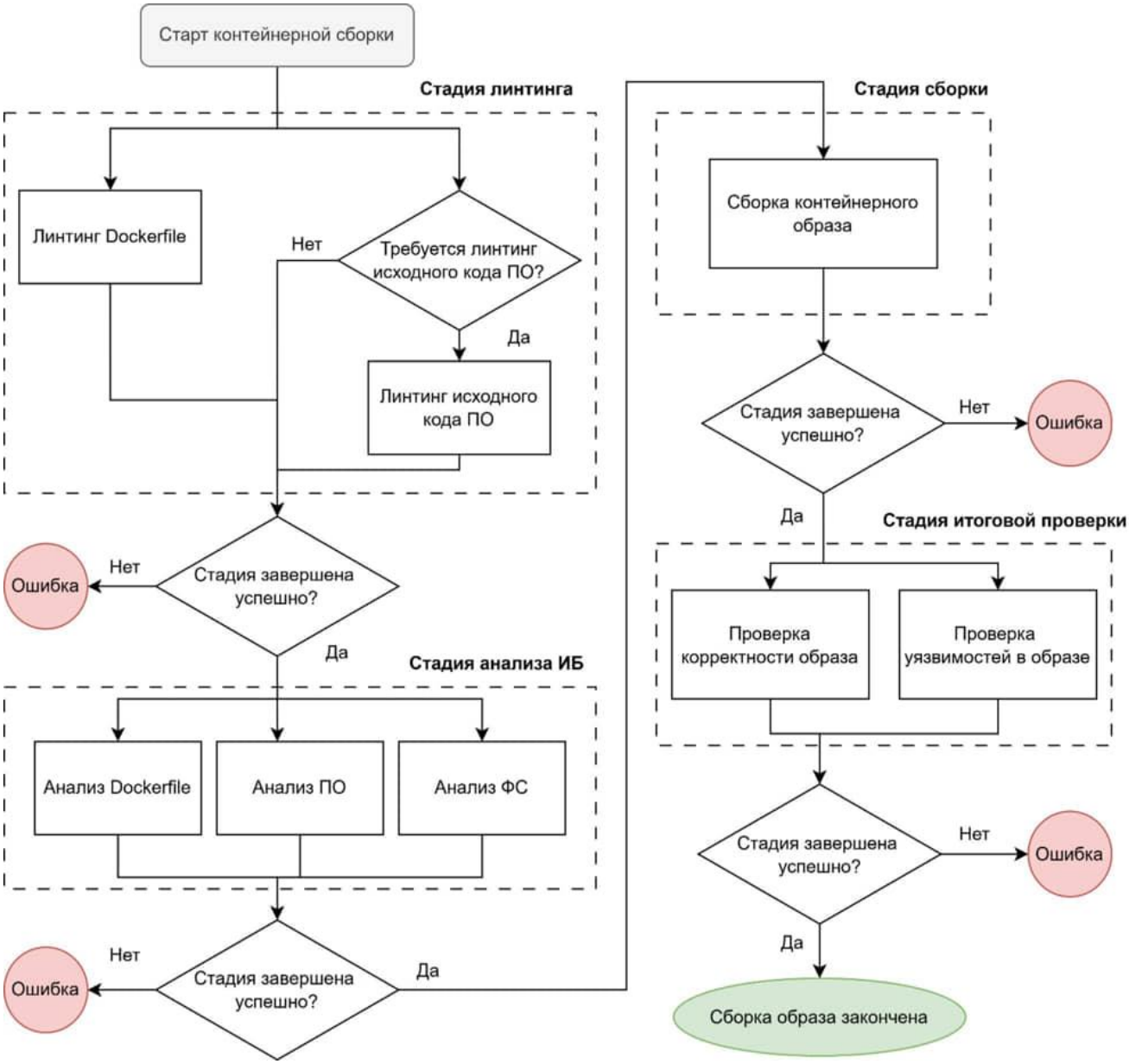

Рисунок 58 – Последовательное выполнение этапов верификации и сборки

Каждый этап сборочного конвейера включен в общий автоматизированный процесс, а переход к следующему этапу возможен только при положительном результате предыдущей проверки, что исключает попадание в производственную среду уязвимых или некорректно сформированных компонентов [153].

Описанная методика реализована в полном соответствии с требованиями, изложенными в разделе 2.3, и обеспечивает дальнейшее развитие автоматизации процедур сопровождения программных компонентов без необходимости ручных вмешательств.

3.5. Организация процедур сопровождения программной среды АСУТП

3.5.1 Этапы сопровождения программной среды и структура программного конвейера

В разработанной системе сопровождения программной средой АСУТП [154] все процессы инициируются и координируются с помощью специализированного программного конвейера, построенного на механизмах автоматизации CI/CD. Структура конвейера охватывает все этапы сопровождения программной среды, начиная с резервного копирования и очистки среды, далее переходя к инициализации структуры хранения и настройке параметров окружения, и завершая установкой, обновлением и запуском прикладных сервисов.

Если сценарий требует восстановления из резервной копии, предусмотрена возможность возврата к зафиксированной ранее конфигурации. Данный алгоритм обеспечивает последовательное выполнение всех этапов сопровождения с момента старта процесса до запуска среды в рабочем режиме. На каждом этапе проводится автоматизированная фиксация состояния и регистрация выполненных операций.

Автоматизация достигается за счет наборов разработанных программных конвейеров, сформировавших библиотеку CIDeploy.

Описание последовательности операций и условий их выполнения формализовано в управляющих конфигурационных файлах (рисунок 59).

```
# ———————— DESTROY ————————
destroy database:
  extends:
    - .ansible executor
    - .allow destroy database
  stage: destroy database
  variables:
    ANSIBLE_SERVICE_DEPLOY_SCRIPT: "prod.database.service_down.sh"
# ———————— DEPLOY ————————
deploy database:
  extends:
    - .ansible executor
    - .allow deploy database
  stage: deploy database
  variables:
    ANSIBLE_SERVICE_DEPLOY_SCRIPT: "prod.database.service_up.sh"
```

Рисунок 59 – Содержимое управляющего конфигурационного файла

Для каждой стадии указывается перечень запускаемых сценариев и параметры, определяющие особенности исполнения.

Реализованная система сопровождения программной среды АСУТП обеспечивает единый детерминированный цикл управления всеми процедурами, представленный на рисунке 60.

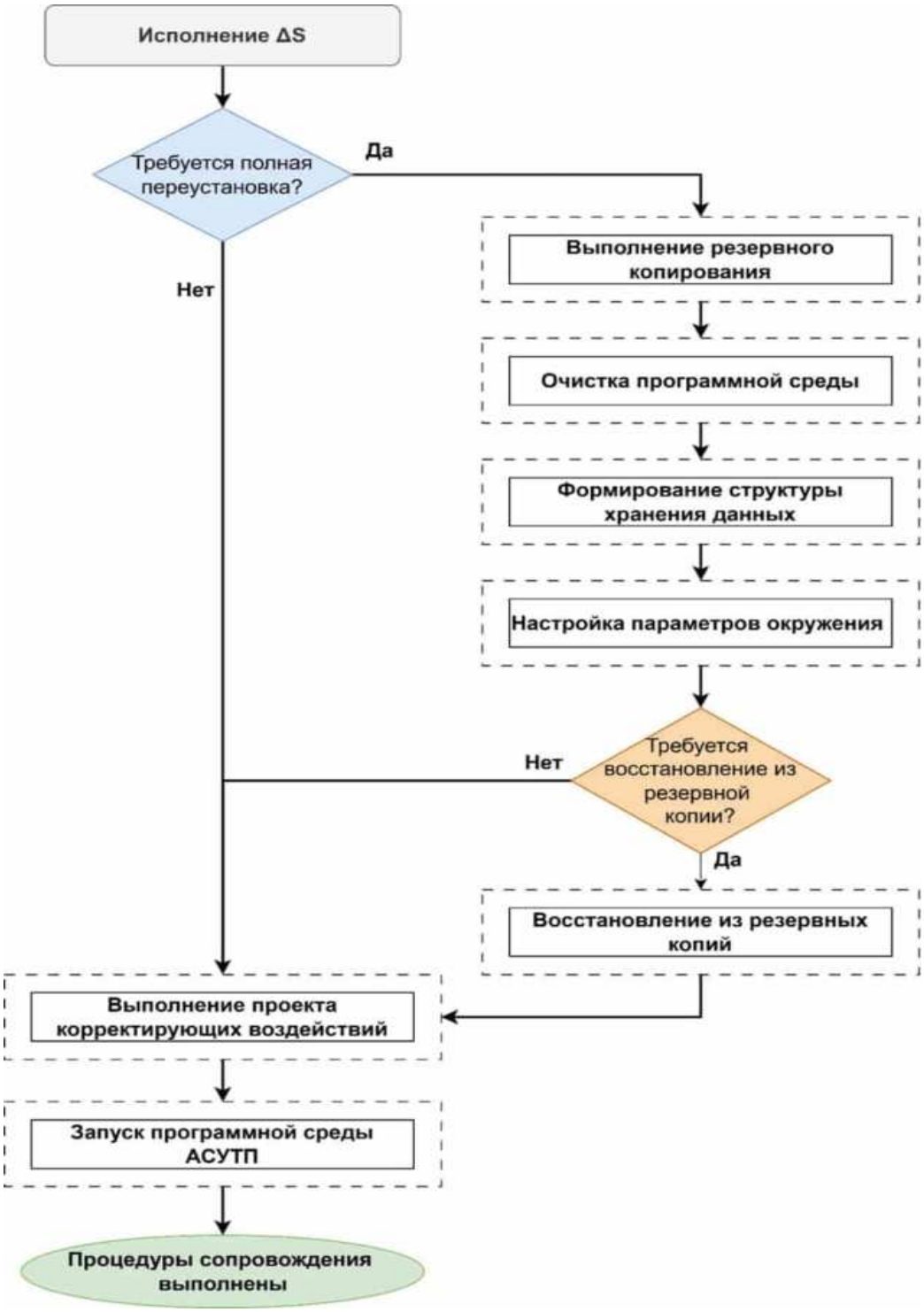

Рисунок 60 – Описание последовательности операций CIDeploy

Представленная блок-схема отражает логику выполнения процедур программного конвейера сопровождения и переходов между ними.

3.5.2 Механизмы активации процедур и управление параметрами выполнения

Настройка программного конвейера реализуется посредством механизмов планирования, интегрированных в CI/CD-интерфейс GitLab. Для каждого задания формализуется набор переменных, определяющих состав активируемых процедур, а также параметры их исполнения (рисунок 61).

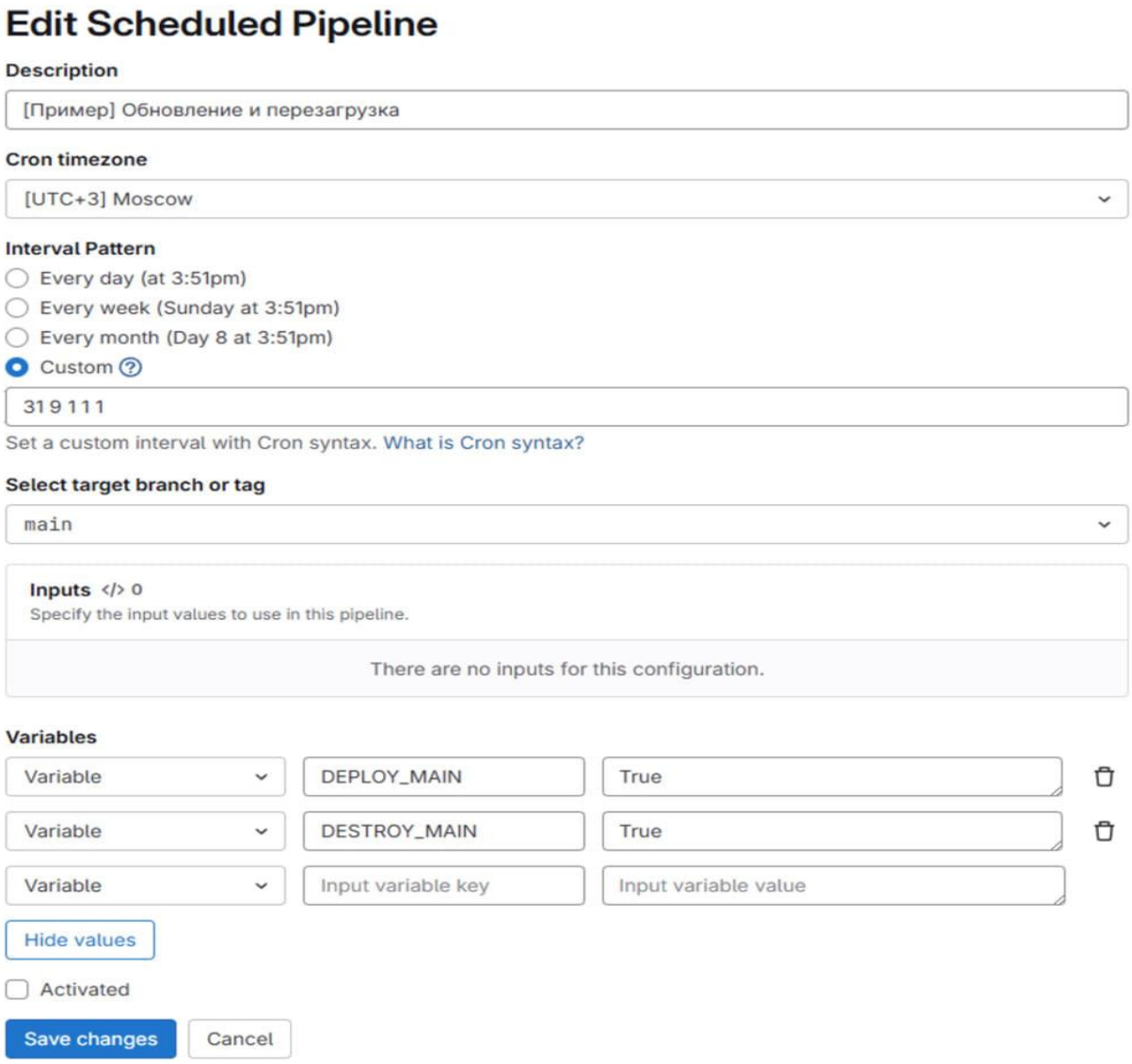

Рисунок 61– Интерфейс определения переменных

В интерфейсе настройки задачи обязательным элементом является указание расписания в формате cron [155], регламентирующего периодичность выполнения проверок процедур сопровождения. Дополнительно предусмотрена возможность задания признака автоматического запуска, при котором выполнение процедуры осуществляется строго по расписанию, либо ручного режима, когда активация производится уполномоченным лицом.

После сохранения настроек задание отображается в общем списке. Экранная форма представлена на рисунке 62.

Рисунок 62 – Панель запуска задачи

С помощью данной экранной формы обеспечивается централизованный контроль и мониторинг состояния выполнения всех процедур сопровождения.

3.5.3 Контроль и безопасность операций сопровождения

Операции сопровождения переходят под централизованный контроль, обеспечиваемый средствами CI/CD-инфраструктуры. Для повышения безопасности и минимизации риска несанкционированных изменений при самостоятельной инициализации процедур сопровождения реализован механизм подтверждения операций (рисунок 63).

Для этого используется защитный механизм SafeLock, предоставляемый средствами программной платформы GitLab, который требует явного подтверждения запуска каждого этапа ответственным лицом. Данный механизм обеспечивает дополнительный уровень защиты, блокируя автоматическое

выполнение ответственных стадий без согласия ответственного лица (удаление содержимого и полной переустановки программной среды АСУТП), тем самым предотвращает несанкционированные или ошибочные действия.

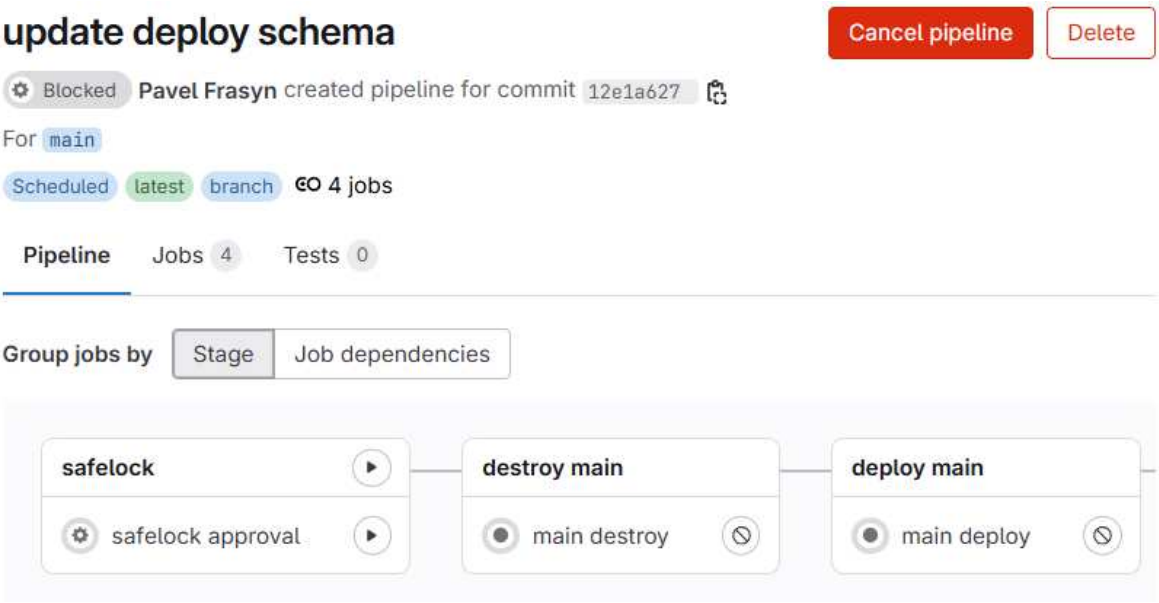

Рисунок 63 – Ожидание подтверждения операции системой сопровождения

При наличии эксплуатационные требований, допускающих полностью автоматизированное выполнение процедур, механизм SafeLock может быть декларативно отключен посредством установки переменной SAFELOCK в значение false (рисунок 64). В этом случае все операции системы сопровождения выполняется без необходимости ручного подтверждения, позволяя повысить уровень автоматизации при сохранении контроля за параметрами выполнения (рисунок 65).

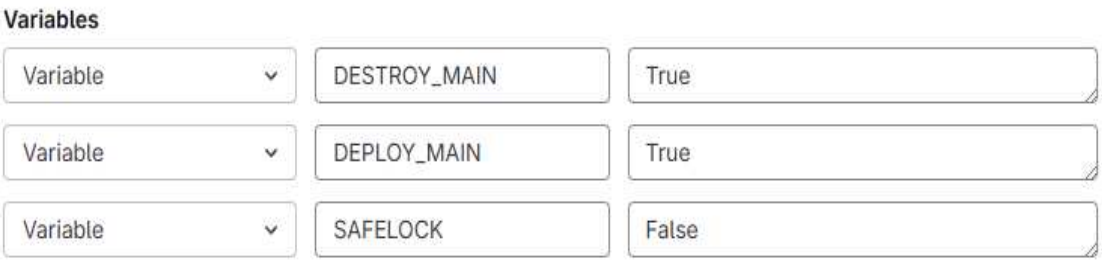

Рисунок 64 – Отключение SAFELOCK

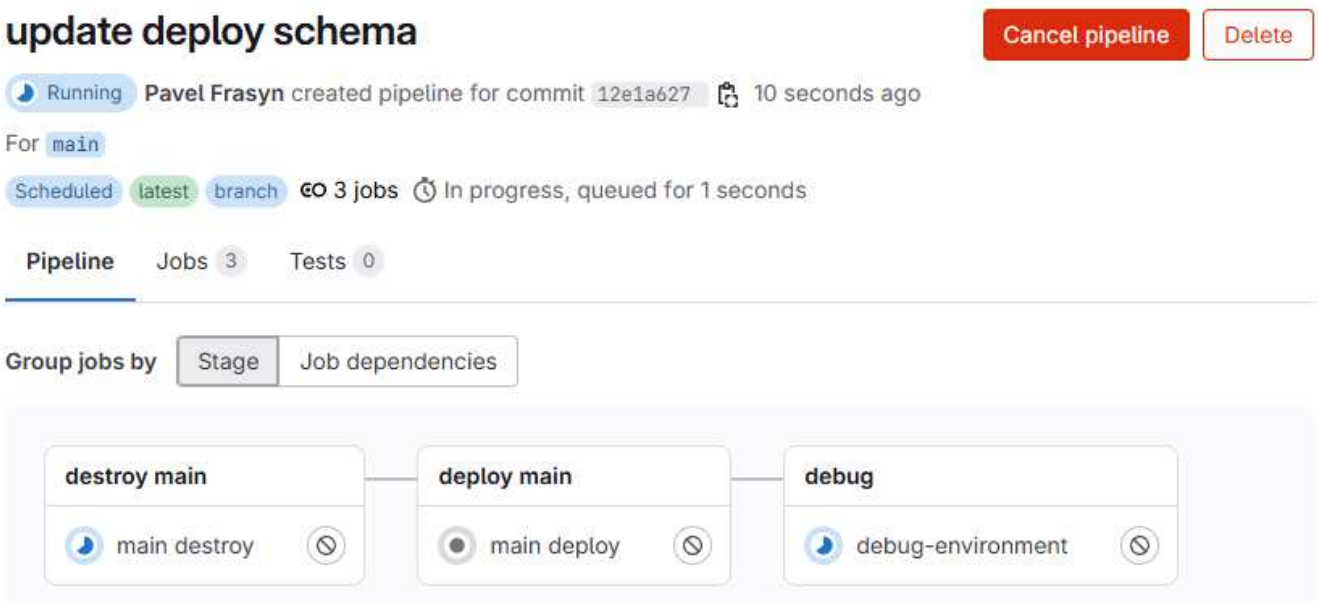

Рисунок 65 – Автоматический запуск конвейера без подтверждения

Таким образом, реализованный механизм управления программой средой АСУТП в контуре сопровождения на базе программной платформы GitLab и библиотеки CIDeploy обеспечивает прозрачность, управляемость и воспроизводимость сопровождения программных компонентов.

Выводы по 3 главе

В третьей главе выполнена практическая реализация разработанных методов сопровождения программной среды диспетчерского уровня АСУТП и сформирована экспериментальная программно-техническая база для ее экспериментальной проверки.

Разработаны и внедрены механизмы сопровождения программной среды, обеспечивающие автоматизированное приведение текущей конфигурации программных компонентов к требуемой конфигурации, заданной в формализованном описании.

Реализован механизм генерации и согласования конфигураций программных компонентов на основе параметризованных Ansible-ролей и шаблонной генерации, обеспечивающий согласованность параметров взаимодействия компонентов и воспроизводимость конфигурации программной среды.

Сформирована воспроизводимая среда исполнения программных компонентов АСУТП на основе контейнерных технологий и автоматизированного сборочного конвейера с многостадийным контролем состава, структуры и безопасности программных компонентов.

Реализована система организации процедур сопровождения программной среды АСУТП на базе CI/CD-инфраструктуры, обеспечивающая централизованное управление процессами установки, обновления и восстановления программных компонентов.

Полученные результаты подтверждают реализуемость предложенных методов сопровождения программной среды АСУТП и создают экспериментальную основу для ее последующей верификации.

ГЛАВА 4. ЭКСПЕРИМЕНТАЛЬНАЯ ВЕРИФИКАЦИЯ И АНАЛИЗ ЭФФЕКТИВНОСТИ СИСТЕМЫ СОПРОВОЖДЕНИЯ ПРОГРАММНОЙ СРЕДЫ АСУТП

4.1 Экспериментальная проверка корректности функционирования методов управления конфигурацией программной среды АСУТП

В соответствии с критериями успешной верификации методов сопровождения программной среды АСУТП, сформулированными в разделе 3.1.4, экспериментальная проверка направлена на подтверждение качественных свойств разработанных методов управления конфигурацией программной среды.

Объектом экспериментальной проверки является программная реализация разработанных методов сопровождения конфигурации, обеспечивающая выявление, формализацию и устранение отклонений фактической конфигурации от нормативного описания.

Нормативное описание конфигурации программной среды формируется на основе предварительной функциональной и эксплуатационной верификации и соответствует состоянию программной среды, допущенному к промышленной эксплуатации.

При анализе работы методов сопровождения конкретная природа эксплуатационных воздействий не учитывается при формировании управляющих решений, и независимо от их вида – планового обслуживания, развития программной среды, восстановления после отказов или переноса вычислительной инфраструктуры – результат выполнения регламентных процедур формализуется как изменение конфигурации программной среды, включая состав программных компонентов, их параметры и условия межкомпонентного взаимодействия..

Таким образом, экспериментальная проверка направлена на подтверждение инвариантных свойств методов сопровождения конфигурации, реализованных в составе системы и сохраняющихся независимо от причин возникновения конфигурационных отклонений.

4.1.1 Корректность выполнения процедур сопровождения при отсутствии конфигурационных отклонений

Целью эксперимента является проверка корректности выполнения процедур управления конфигурацией программной среды при отсутствии конфигурационных отклонений и фиксированном нормативном описании конфигурации $S_{\text{цел}}$.

Эксперимент проводился при фиксированной конфигурации $S_{\text{цел}}$ программной среды, не изменяющемся в процессе наблюдения. Нормативное описание конфигурации реализовано в виде параметризованного файла Ansible, однозначно определяющего требуемый состав и параметры программной среды.

В таблице 4 приведен фрагмент конфигурации, используемой в эксперименте.

Таблица 4 – Фрагмент нормативной конфигурации программной среды диспетчерского уровня АСУТП

Компонент	**Назначение**	**Версия и идентификатор сборки**
Modbus Reader	Опрос полевых устройств	v1.4.2 (build 2024-03-15)
MQTT	Транспорт телеметрии	Eclipse Mosquitto 2.0.18
Converter	Преобразование телеметрии	v2.1.0 (commit a3f9c7d)
PostgreSQL	Долговременное хранение	PostgreSQL 16.3
HMI	Интерфейс оператора	v3.0.1 (build 2024-04-02)

Для иллюстрации структуры конфигурации на рисунке 66 приведен фрагмент Ansible, фиксирующий состав программных компонентов, их версии, а также параметры размещения и межкомпонентного взаимодействия в рамках вычислительного узла.

```
# prepare_service_postgresql_db
- role: prepare_service_postgresql_db
  vars:
    override_prepare_service_postgresql_db:
      service_name: "tomilino_dispatcher_postgres"
      docker_image: "{{ docker_registry }}/system/postgres:16.3"
      ports:
        api:
          external: "{{ postgres_properties.port }}"
      ENVIRONMENT:
        POSTGRES_PASSWORD: "{{ credentials_db_postgres_superuser.password }}"
        POSTGRES_DB: "{{ credentials_db_postgres_superuser.login }}"
        POSTGRES_USER: "{{ credentials_db_postgres_superuser.login }}"
        #
        # using logical replicant, this postgres master
        POSTGRES_COMMAND: "postgres -c 'tcp_keepalives_idle=60' -c 'tcp_keepa
  tags: prepare_service_postgresql_db
```

Рисунок 66 – Фрагмент описания конфигурации базы данных

Для обеспечения однозначной идентификации нормативной конфигурации используется ее интегральное представление в виде хеш-суммы, вычисляемой по совокупности параметров, в том числе составом программных компонентов, версиями, идентификаторами сборок, настройки межкомпонентного и сетевого взаимодействия.

Полученное значение хеш-суммы по алгоритму SHA-256 используется системой сопровождения как компактное представление нормативной конфигурации $S_{\text{цел}}$. При каждом запуске процедур формируется хеш-представление фактической конфигурации программной среды $S(t)$, определяемое по данным о реально развернутых программных компонентах и параметрах среды исполнения (см. 3.3.1).

В протоколах выполнения процедур сопровождения нормативное представление конфигурации $S_{\text{цел}}$ обозначается как remote, а фактическое представление $S(t)$ – как current. На рисунке 67 приведен фрагмент протокола выполнения процедуры сопровождения, иллюстрирующий результат сопоставления хеш-представлений нормативной и фактической конфигураций при повторных запусках операций сопровождения. В обоих случаях хеш-сумма конфигурации равняется

c1a3bdb4a374c4a91b0038ac6ce30b567cbc4aa7e84d78bc02b37a45bef96d39.

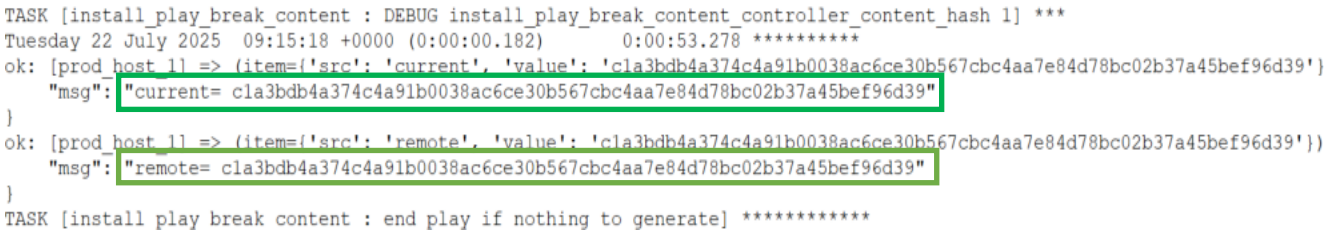

Рисунок 67 – Сопоставление хеш-представлений нормативной и фактической конфигураций программной среды

Совпадение значений хеш-сумм нормативного и фактического представлений конфигурации свидетельствует о тождественности конфигураций $S(t) = S_{\text{цел}}$. В этом случае вектор отклонений $e(t)$ принимает нулевое значение, что формализованно интерпретируется системой как отсутствие конфигурационных рассогласований и приводит к завершению процедур сопровождения без формирования корректирующих воздействий (результат «end play if nothing to generate» на рисунке 67).

Повторное выполнение процедур сопровождения при неизменной нормативной конфигурации приводит к идентичному результату, что экспериментально подтверждает корректность работы системы сопровождения при отсутствии конфигурационных отклонений и отсутствие побочных изменений конфигурации программной среды при повторном выполнении процедур.

4.1.2 Экспериментальная проверка автоматизированного выявления конфигурационных расхождений

Экспериментальная проверка диагностической способности системы сопровождения выполнялась путем формирования контролируемого расхождения между нормативным описанием и фактической конфигурацией программной среды.

Расхождение формировалось путем изменения нормативного описания при сохранении неизменной фактической конфигурации. В качестве экспериментального воздействия была изменена версия программного компонента

интерфейса оператора (HMI) в нормативном описании конфигурации, тогда как фактически развернутый экземпляр продолжал функционировать в ранее установленной версии.

Таким образом, указанное изменение формирует новую нормативную конфигурацию $S_{\text{цел}}$, в то время как фактическая используемая конфигурация программной среды $S(t)$ остается неизменной. На рисунке 68 приведен фрагмент изменения нормативной конфигурации, иллюстрирующий обновление требуемой версии программного компонента HMI.

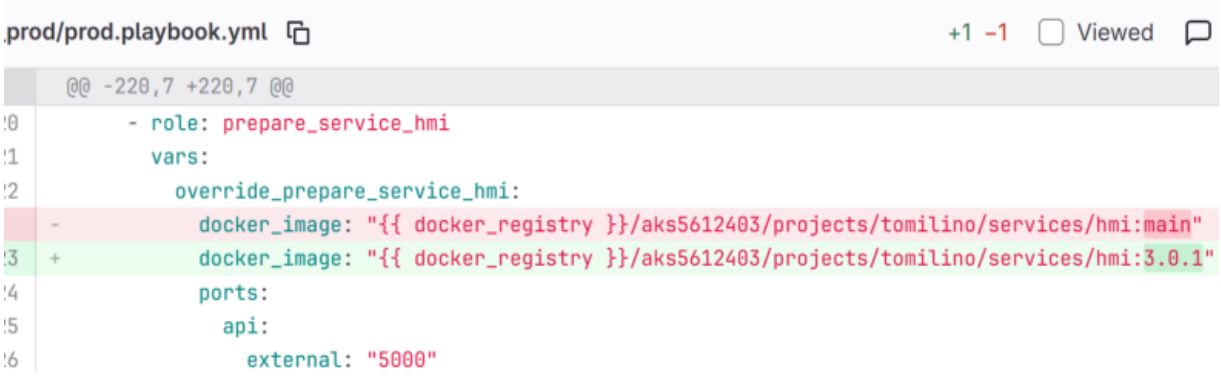

Рисунок 68 – Изменение нормативного описания конфигурации программного компонента HMI

На рисунке 69 приведен фрагмент протокола выполнения процедуры сопровождения, отражающий результат сопоставления хеш-представлений конфигурации при наличии расхождения.

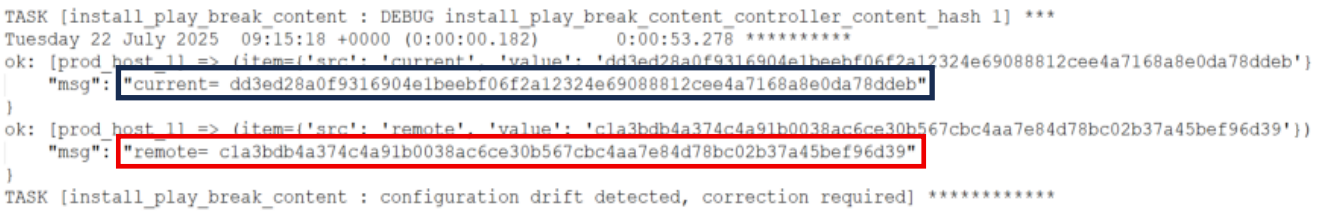

Рисунок 69 – Сопоставление хеш-представлений конфигурации

В представленном фрагменте протокола значения хеш-сумм current и remote различаются.

Хеш-сумма требуемой конфигурации равняется **dd3ed28a0f9316904e1beebf06f2a12324e69088812cee4a7168a8e0da78ddeb**,

а не фактической используемой – **c1a3bdb4a374c4a91b0038ac6ce30b567cbc4aa7e84d78bc02b37a45bef96d39**.

Различие значений хеш-сумм current и remote интерпретируется системой сопровождения как наличие конфигурационного расхождения, то есть $S(t) \neq S_{\text{цел}}$. В этом случае вектор отклонений $e(t)$ принимает ненулевое значение.

Факт выявления конфигурационного расхождения фиксируется системой сопровождения до формирования корректирующих воздействий и обозначает завершение диагностической фазы алгоритма сопровождения. В протоколе выполнения данный переход обозначается сообщением «configuration drift detected, correction required» (рисунок 69).

4.1.3 Экспериментальная проверка формирования и применения корректирующих воздействий

Формирование информации, необходимой для автоматизированного устранения конфигурационного расхождения, осуществляется в два этапа. На первом этапе выполняется интегральная оценка тождественности фактической и нормативной конфигураций программной среды на основе сопоставления их хеш-представлений, позволяя установить сам факт наличия конфигурационного расхождения. На втором этапе система сопровождения, оперируя параметризированными моделями $S(t)$ и $S_{\text{цел}}$ (см. раздел 2.1 и 2.2), выполняет поэлементное сопоставление их параметров. В результате формируется вектор отклонений $e(t)$, содержащий информацию о составе, локализации и характере выявленных расхождений.

Соответственно, после фиксации конфигурационного расхождения и формирования ненулевого вектора отклонений $e(t)$ в ходе диагностической фазы алгоритма сопровождения (раздел 4.1.2) система автоматически переходит к

корректирующей фазе. На данном этапе управление конфигурацией осуществляется в соответствии с декларативной моделью (см. раздел 2.2.2), в рамках которой выявленные отклонения используются для вычисления совокупности корректирующих воздействий, направленных на приведение конфигурации программной среды к нормативному описанию.

В ходе корректирующей фазы системой сопровождения выполняется анализ сформированного вектора отклонений $e(t)$, отражающего различия между фактической эксплуатируемой конфигурацией программной среды $S(t)$ и нормативной $S_{\text{цел}}$. Формирование корректирующих воздействий осуществляется в соответствии с моделью управления конфигурацией, описанной выражениями (2.7)-(2.11), согласно которой каждому ненулевому элементу вектора отклонений сопоставляется регламентированное управляющее действие, определяемое функцией (2.8). Фрагмент $e(t)$ представлен в таблице 5.

Таблица 5 – Фрагмент вектора отклонений $e(t)$ при экспериментальном конфигурационном расхождении

Объект конфигурации	**Контролируемый параметр**	**Фактическое значение $S(t)$**	**Нормативное значение $S_{\text{цел}}$**	**Тип отклонения**
HMI	Версия ПО	main	3.0.1	Несоответствие версии

Таким образом, состав и параметры корректирующих воздействий ΔS однозначно определяются структурой выявленных отклонений и не зависят от субъективных решений обслуживающего персонала.

В рассматриваемом эксперименте вектор отклонений $e(t)$ содержал ненулевой элемент, соответствующий параметру версии программного компонента интерфейса оператора (HMI). Данное отклонение было интерпретировано системой сопровождения как несоответствие версии программного компонента требованиям нормативного описания конфигурации. В результате применения функции формирования корректирующих воздействий был автоматически сформирован проект с управляющим воздействие на программную среду. Он

заключается в установке требуемой нормативной версии программного компонента и применении регламентированных параметров его настройки. Запуск проекта корректирующих воздействий подтверждается эксплуатационным персоналом.

На рисунке 70 приведен фрагмент протокола выполнения процедуры сопровождения, иллюстрирующий процесс формирования и применения воздействий в рамках корректирующей фазы алгоритма. Факт выполнения корректирующего воздействия отражается в протоколе наличием записей со статусом changed, что свидетельствует о выполнении операций, соответствующих рассчитанной совокупности воздействий ΔS.

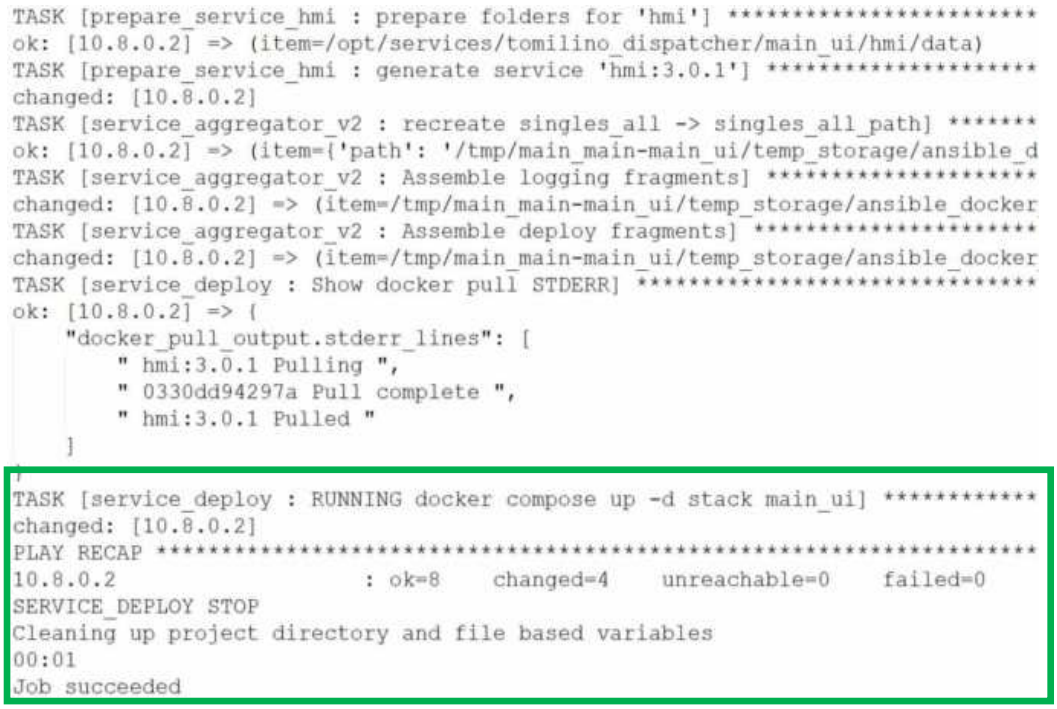

Рисунок 70 – Формирование и применение корректирующих воздействий

После выполнения корректирующих воздействий системой сопровождения осуществляется повторное сопоставление фактической конфигурации программной среды с нормативным описанием. Для этого вновь формируется хеш-представление фактической конфигурации $S(t)$, которое сопоставляется с хеш-представлением нормативной конфигурации $S_{\text{цел}}$.

На рисунке 71 приведен фрагмент протокола выполнения процедуры сопровождения после применения корректирующих воздействий.

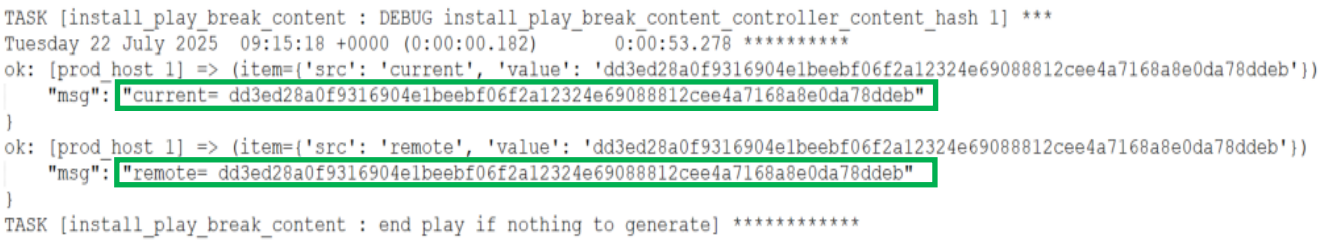

Рисунок 71 – Сопоставление нормативной и фактической конфигураций программной среды после устранения конфигурационного расхождения

В представленном фрагменте зафиксировано совпадение значений хеш-сумм фактического и нормативного представлений конфигурации, обе из которых равны **dd3ed28a0f9316904e1beebf06f2a12324e69088812cee4a7168a8e0da78ddeb**.

Совпадение значений хеш-сумм рассматривается системой как выполнение условия $S(t) = S_{\text{цел}}$, что соответствует нулевому значению вектора отклонений $e(t)$. В этом случае формирование корректирующих воздействий больше не требуется, и процедуры сопровождения завершаются без выполнения дополнительных операций, что в протоколе выполнения отражается сообщением «end play if nothing to generate» (рисунок 71).

Повторные запуски алгоритма сопровождения после устранения конфигурационного расхождения приводят к идентичному результату сопоставления конфигураций (см. раздел 4.1.1), что подтверждает корректность функционирования корректирующей фазы алгоритма сопровождения и отсутствие повторного накопления конфигурационных рассогласований при последующих запусках процедур.

Аналогичным образом были экспериментально проверены отклонения параметров взаимодействия компонентов (порт, параметры подключения), а также отклонения параметров прикладного диспетчерского проекта (состав проекта, активные элементы), для которых система сопровождения корректно формировала вектор отклонений $e(t)$ и обеспечивала восстановление нормативной конфигурации.

Полученные экспериментальные результаты также подтверждают, что формализованное представление фактической конфигурации $S(t)$, используемое в системе сопровождения, удовлетворяет критериям полноты и достоверности, сформулированным в разделах 2.1.2 и 2.1.3.

4.2. Количественная оценка эффективности системы сопровождения программной среды в промышленной эксплуатации

Количественная оценка эффективности системы сопровождения программной среды диспетчерского уровня АСУТП выполнена в условиях промышленной эксплуатации технологической площадки, характеристики которой приведены в разделе 3.1 настоящей работы. В рамках данного раздела рассматриваются условия выполнения регламентных процедур сопровождения и методика количественной оценки трудоемкости эксплуатационных операций, связанных с сопровождением программной среды.

Сопровождение диспетчерского уровня АСУТП на объекте водозаборного узла осуществляется в соответствии с утвержденными эксплуатационными и технологическими регламентами, определяющими порядок выполнения работ по администрированию, сопровождению и восстановлению программной среды при плановых и внеплановых воздействиях. Указанные регламенты задают состав, периодичность и последовательность действий эксплуатационного персонала, направленных на поддержание согласованности фактической эксплуатируемой конфигурации программной среды диспетчерского уровня АСУТП с ее установленным нормативным описанием.

Эксплуатирующей организацией технологического водозаборного узла является ООО «Самолет-Ресурс». Работы по регламентному сопровождению программной среды и аппаратной инфраструктуре диспетчерского уровня АСУТП выполняются специализированной организацией – ООО «АК-Системы», осуществляющей комплексные работы в области промышленной автоматизации и диспетчеризации технологических объектов.

4.2.1 Порядок количественной оценки трудоемкости процедур сопровождения

Количественная оценка трудоемкости процедур сопровождения программной среды диспетчерского уровня АСУТП выполнена по данным фактической промышленной эксплуатации за период с января 2022 г. по апрель 2025 г. Оценка носит ретроспективный характер и основана на анализе реально выполненных эксплуатационных операций без применения прогнозных или модельных допущений.

В качестве исходных данных использованы первичные эксплуатационные журналы сопровождения, содержащие сведения о выездах эксплуатационного персонала на объект, фактически затраченном времени и перечне выполненных работ, оформленные в установленном порядке, а также протоколы автоматизированных запусков процедур, сформированных внедренной системой сопровождения программной среды в процессе эксплуатации (рисунок 72).

Указанные источники данных используются для формирования эмпирической базы расчета показателей трудоемкости процедур сопровождения программной среды диспетчерского уровня АСУТП.

```
OBJECT:  vod_zabor_tomilino_park
EXPORT:  automation_jobs
FROM:    2024-07-01T00:00:00
TO:      2024-08-01T00:00:00
CREATED: 2024-10-15T14:32:11
```

date	time	job_id	type	description	status	duration
2024-07-01	00:00:12	job-20240701-0000	scheduled	configuration check, service health	success	00:38
2024-07-01	00:10:11	job-20240701-0010	scheduled	configuration check, service health	success	00:39
2024-07-01	00:20:13	job-20240701-0020	scheduled	configuration check, service health	success	00:37
2024-07-01	00:30:14	job-20240701-0030	scheduled	configuration check, service health	success	00:41
2024-07-01	00:40:12	job-20240701-0040	scheduled	configuration check, service health	success	00:38
2024-07-01	00:50:11	job-20240701-0050	scheduled	configuration check, service health	success	00:40
2024-07-01	01:00:09	job-20240701-0100	scheduled	configuration check, service health	success	00:39
2024-07-01	01:10:11	job-20240701-0110	scheduled	configuration check, service health	success	00:38
2024-07-01	01:20:10	job-20240701-0120	scheduled	configuration check, service health	success	00:40

Рисунок 72. Журнал выгрузки выполняемых операций системы сопровождения

В расчет включались исключительно трудозатраты, связанные с выполнением процедур эксплуатационного сопровождения программной среды

диспетчерского уровня АСУТП, направленных на поддержание нормативной конфигурации в процессе эксплуатации. Не учитывались трудозатраты, связанные с разработкой и модификацией прикладного программного обеспечения, проектированием технологических решений и обслуживанием полевого оборудования, а также организационно-административные и управленческие работы, не относящиеся непосредственно к сопровождению программной среды диспетчерского уровня АСУТП.

Все трудозатраты учитывались в форме человеко-часов. В качестве базового интервала агрегации трудоемкости выбран календарный месяц. Месячная трудоемкость сопровождения программной среды определяется как сумма фактически затраченного рабочего времени эксплуатационного персонала на выполнение всех регламентных и внеплановых процедур в соответствующем календарном месяце (4.15).

$$T_m = \sum_{i=1}^{K} \mathrm{t_{i,m}}, \tag{4.15}$$

где $\mathrm{t_{i,m}}$ – фактические затраты времени на выполнение i-ой процедуры сопровождения в месяце m, зафиксированные в эксплуатационных журналах; K – количество выполненных процедур сопровождения в рассматриваемом месяце.

Величина T_m характеризует полный объем фактических трудозатрат, необходимых для поддержания программной среды диспетчерского уровня АСУТП в нормативной конфигурации в промышленной эксплуатации. Методика расчета не предполагает нормирования или приведения процедур сопровождения к единой сложности; каждая процедура учитывается по фактически затраченному времени.

В расчет трудоемкости включались процедуры сопровождения, относящиеся к следующим функциональным группам:

- контроль и поддержание нормативной конфигурации программной среды, включая проверку состава программных компонентов, значений конфигурационных параметров и условий межкомпонентного взаимодействия в соответствии с формализованным нормативным описанием;

- обеспечение работоспособности и готовности программной среды, включая контроль факта запуска программных компонентов, их доступности и штатного состояния по регламентированным критериям, а также соблюдения установленного порядка инициализации после плановых и внеплановых воздействий;
- сопровождение изменений программной среды, связанных с развитием технологического объекта, интеграцией новых источников технологических данных и актуализацией нормативного описания конфигурации программной среды;
- процедуры проверки и восстановления программной среды при эксплуатационных воздействиях, включая приведение фактической конфигурации программной среды к заданному формализованному нормативному описанию в случае возникновения отказов, выполнения регламентных работ по обслуживанию вычислительной инфраструктуры, замены аппаратных компонентов или переноса программной среды на резервные вычислительные узлы;
- приемочная верификация конфигурации программной среды после выполнения восстановительных и регламентных операций, включая проверку готовности интерфейсов диспетчеризации, прохождение регламентированных проверок работоспособности сервисов, а также контроль полноты и актуальности технологических архивов в соответствии с установленными требованиями.

Сопоставимость показателей трудоемкости обеспечивалась сохранением состава регламентных процедур сопровождения в пределах каждого календарного года.

4.2.2 Сопоставление трудоемкости сопровождения в различные периоды эксплуатации

Сопоставление трудоемкости сопровождения программной среды диспетчерского уровня водозаборного узла выполнено по данным фактической

промышленной эксплуатации за период с января 2022 г. по апрель 2025 г. Динамика трудоемкости представлена на рисунке 73.

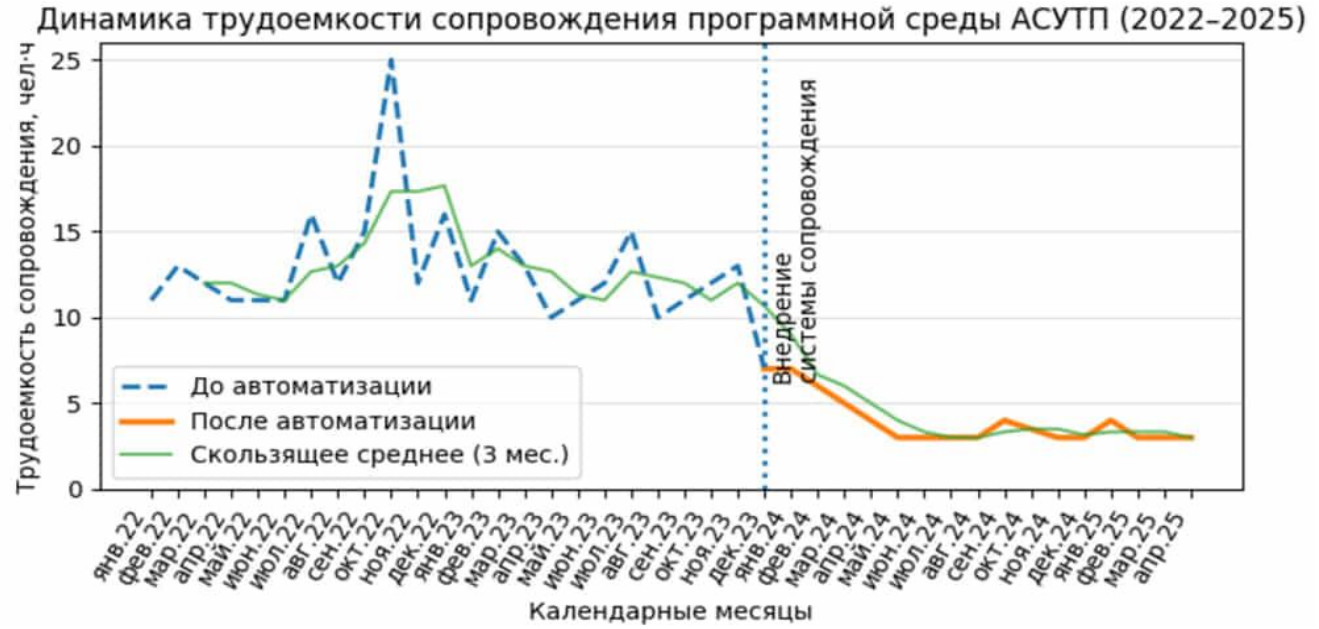

Рисунок 73. Динамика трудоемкости сопровождения программной среды 2022-2025 год

В рассматриваемый период эксплуатация программной среды диспетчерского уровня АСУТП осуществлялась в условиях различной степени формализации и автоматизации процедур сопровождения.

На начальном этапе сопровождение программной среды осуществлялось преимущественно на основе регламентных ручных операций с ограниченным применением средств автоматизации. По мере внедрения и последовательного развития системы сопровождения происходил переход к более полному и формализованному охвату процедур управления конфигурацией программной среды.

Весь период эксплуатации был разбит на календарные годы, в пределах которых условия организации процедур и набор регламентных действий сопровождения программной среды оставались неизменными. Для каждого года рассчитывались значения месячной трудоемкости сопровождения T_m, определенные согласно выражению (4.15). Полученные значения использовались

для анализа динамики трудоемкости сопровождения и выявления устойчивых тенденций изменения эксплуатационных трудозатрат.

Для наглядной интерпретации полученных результатов трудоемкость сопровождения дополнительно представлена в стоимостном выражении с использованием нормированной расчетной стоимости одного человеко-часа, принятой равной 2000 руб.

На рисунках 74-77 представлены динамики месячной трудоемкости сопровождения программной среды диспетчерского уровня АСУТП за 2022-2025 гг.

Анализ данных за 2022 год, представленных на рисунке 74, выполнен на основе показателя месячной трудоемкости сопровождения T_m, определяемого согласно выражению (4.15). В течение 2022 года значения T_m изменялись в диапазоне от 11 до 25 чел. /ч в месяц при среднем значении 13,75 чел. /ч.

Рисунок 74. Трудоемкость эксплуатации сопровождения программной среды в 2022 году

Данный уровень трудоемкости и выраженная неравномерность значений T_m характерны для начального периода эксплуатации программной среды, при

котором сопровождение осуществлялось преимущественно в форме очных регламентных и внеплановых работ с участием выездных эксплуатационных бригад. Существенная вариативность обусловлена необходимостью выполнения значительного объема очных операций с программной средой и ее инфраструктурой непосредственно на объекте, а также ограниченной возможностью дистанционного выполнения процедур сопровождения.

В стоимостном выражении затраты на сопровождение программной среды в 2022 году составляли от 22 тыс. руб. до 50 тыс. руб. в месяц, а совокупные годовые затраты – 330 тыс. руб.

В конце IV квартала 2023 года была введена в промышленную эксплуатацию система сопровождения программной среды, что на рисунке 75 отражено выделением значения для декабря 2023 года.

Рисунок 75. Трудоемкость эксплуатации сопровождения программной среды в 2023 году

После ввода системы сопровождения значение месячной трудоемкости в декабре 2023 года составило 7 чел. /ч, что более чем в 1,4 раза ниже среднего значения месячной трудоемкости за предшествующий период 2023 года (~ 11 чел. /ч).

Наблюдаемое снижение трудоемкости связано с началом практического применения формализованных и автоматизированных процедур управления конфигурацией программной среды, а также с увеличением доли операций сопровождения, выполняемых дистанционно без привлечения выездных эксплуатационных бригад.

В этом случае среднемесячные затраты на сопровождение программной среды в 2023 году составили около 23,3 тыс. руб., а в декабре 2023 года снизились до 14 тыс. руб. Совокупные затраты на сопровождение программной среды за 2023 год составили 280 тыс. руб.

Анализ данных, представленных на рисунке 76, показывает дальнейшее снижение трудоемкости сопровождения программной среды.

Рисунок 76. Трудоемкость эксплуатации сопровождения программной среды в 2024 году

В процессе донастройки системы сопровождения и расширения нормативного описания конфигурации программной среды за счет формализации состава программных компонентов и их параметров в феврале-марте 2024 года значения месячной трудоемкости сопровождения снизились до уровня порядка 3-4 чел. /ч в месяц. Дальнейшая динамика значений T_m характеризуется стабилизацией трудоемкости сопровождения при воспроизводимом выполнении формализованных и автоматизированных процедур управления конфигурацией программной среды. В этом случае среднемесячные затраты на сопровождение программной среды в 2024 году составили около 7,9 тыс. руб., а совокупные годовые затраты – 95 тыс. руб.

Анализ данных, представленных на рисунке 77, показывает, что в I квартале 2025 года значения месячной трудоемкости сопровождения находились в диапазоне от 3 до 7 чел. /час в месяц, при среднем значении около 4 чел. /час.

Рисунок 77. Трудоемкость эксплуатации сопровождения программной среды в I квартале 2025 года

Сохраняющийся низкий уровень и малая вариативность значений T_m свидетельствуют о переходе системы сопровождения программной среды в режим

устойчивой эксплуатации с воспроизводимым выполнением формализованных и автоматизированных процедур управления конфигурацией.

В стоимостном выражении среднемесячные затраты на сопровождение программной среды за первые 4 месяца 2025 года составили около 6,5 тыс. руб., а суммарные затраты за рассматриваемый период – 26 тыс. руб.

Обобщенные значения показателей трудоемкости сопровождения программной среды и соответствующие им стоимостные оценки для рассматриваемых периодов эксплуатации приведены в таблице 6.

Таблица 6 – Сводные показатели трудоемкости сопровождения программной среды диспетчерского уровня АСУТП

Период эксплуатации	**Среднее T_m, чел. /ч/месяц**	**Среднемесячные затраты, тыс. руб.**	**Совокупные затраты за период, тыс. руб.**
2022 год	13,75	27,5	330
2023 год	11,67	23,3	280
2024 год	3,96	7,9	95
январь – апрель 2025	3,25	6,5	26

Сохранение низкого уровня затрат подтверждает устойчивый характер достигнутого снижения эксплуатационных трудозатрат, полученного по итогам внедрения системы сопровождения в конце 2023 года.

Выводы по 4 главе

В главе 4 выполнена экспериментальная верификация разработанной системы сопровождения программной среды диспетчерского уровня АСУТП в условиях промышленной эксплуатации.

Экспериментально подтверждено, что разработанные методы обеспечивают достижение состояния технической определенности программной среды. Это реализуется за счет процедур автоматизированного контроля и верифицируемого устранения рассогласований фактических параметров с нормативными, позволяющих проводить регламентные изменения в рамках установленных технологических ограничений.

Показана работоспособность автоматизированного механизма выявления конфигурационных расхождений между фактической и нормативной конфигурациями на основе формализованного сопоставления их представлений.

Установлено, что формирование и применение корректирующих воздействий на конфигурацию программной среды осуществляется системой сопровождения автоматически на основе выявленного вектора отклонений.

На основе данных фактической промышленной эксплуатации за период 2022-2025 гг. выполнена количественная оценка трудоемкости процедур сопровождения, направленных на поддержание согласованности фактической конфигурации программной среды с ее нормативным описанием.

Показано, что внедрение системы сопровождения обеспечивает переход от преимущественно очных регламентных и внеплановых операций к воспроизводимым автоматизированным процедурам управления конфигурацией, сопровождающийся устойчивым снижением трудоемкости эксплуатационных работ и уменьшением вариативности трудозатрат.

Представление трудоемкости процедур сопровождения в стоимостном выражении показало существенное снижение эксплуатационных затрат, связанных с сопровождением конфигурации программной среды, что подтверждает

практическую эффективность внедрения системы сопровождения в условиях промышленной эксплуатации.

Таким образом, результаты экспериментальной верификации подтверждают выполнение критериев успешности методов управления конфигурацией программной среды АСУТП, сформулированных в подразделе 3.1.4, а также демонстрируют практическую реализуемость и эксплуатационную эффективность разработанной системы сопровождения.

ЗАКЛЮЧЕНИЕ

В ходе выполнения диссертационного исследования получены следующие основные результаты:

1. Обоснованы эксплуатационные и архитектурные предпосылки необходимости автоматизированного сопровождения программной среды диспетчерского уровня АСУТП.

2. Разработана модель формализованного представления конфигурации программной среды диспетчерского уровня АСУТП, обеспечивающая параметрическое описание состава программных компонентов, их характеристик и сопоставимость фактической и нормативной конфигураций.

3. Предложена архитектурная организация автоматизированного сопровождения программной среды диспетчерского уровня АСУТП, основанная на выделении специализированной системы сопровождения, независимой от реализации прикладных программных компонентов.

4. Разработаны императивный и декларативный методы управления конфигурацией программной среды АСУТП, формализующие способы приведения ее к требуемым параметрам на основе регламентированных операций и автоматизированного сопоставления фактических и нормативных значений.

5. Предложен метод автоматизированного сопровождения программной среды диспетчерского уровня АСУТП, обеспечивающий автоматизированное выявление отклонений и формирование проекта корректирующих воздействий, применение которого осуществляется уполномоченным эксплуатационным персоналом в регламентированные временные интервалы.

6. Разработан программный комплекс системы сопровождения программной среды диспетчерского уровня АСУТП, реализующий предложенные архитектурные решения и методы.

7. Проведена экспериментальная верификация системы сопровождения на программной среде диспетчерского уровня действующего технологического объекта водозаборного узла, подтвердившая возможность автоматизированного

выявления конфигурационных отклонений и формирования проекта корректирующих воздействий, обеспечивающих приведение фактической конфигурации к нормативному описанию в процессе эксплуатации, а также восстановление программной среды до нормативных параметров и снижение трудоемкости эксплуатационных операций.

Таким образом, в диссертационной работе решена научно-техническая задача разработки методов автоматизированного сопровождения конфигурации программной среды диспетчерского уровня АСУТП, заключающаяся в формализованной организации процедур сопровождения, обеспечении соответствия конфигурации программных компонентов нормативному описанию и восстановлении программной среды в условиях промышленной эксплуатации.

Рекомендации и перспективы дальнейших исследований связаны с развитием предложенных методов автоматизированного сопровождения программной среды диспетчерского уровня АСУТП в направлении интеллектуализации процедур анализа и управления конфигурацией на основе формализованных конфигурационных данных.

Существенный интерес представляет разработка методов прогнозной оценки изменений конфигурации программной среды и вероятности возникновения конфигурационных расхождений с использованием накопленных данных о составе и параметрах программных компонентов, а также истории эксплуатационных воздействий.

Дополнительного внимания требует развитие методов формального анализа эффективности процедур сопровождения и адаптивных стратегий выбора корректирующих воздействий, ориентированных на оценку последствий конфигурационных изменений и степени достижения нормативного описания $S_{\text{цел}}$ в различных условиях эксплуатации.

СПИСОК ТЕРМИНОВ

Контур управления – совокупность средств и процедур, обеспечивающих централизованное управление параметрами и конфигурацией программной среды.

Площадка – отдельная вычислительная машина, сервер или кластер, на которых развернута программная среда АСУТП и осуществляется ее управление и сопровождение.

Процесс развертывания – совокупность процедур, обеспечивающих установку, настройку и ввод в эксплуатацию программной среды АСУТП на выбранной площадке.

Конвейер (pipeline, пайплайн) – последовательность программных автоматизированных процедур, обеспечивающих реализацию процессов развертывания, обновления и контроля среды исполнения.

Среда исполнения – совокупность аппаратных и программных ресурсов, в рамках которых функционирует программное программная среда АСУТП.

Корректирующее воздействие – действие или процедура, направленные на устранение выявленных конфигурационных отклонений среды исполнения от нормативных параметров.

Прикладные компоненты – программные элементы, реализующие конкретные функции мониторинга, диспетчеризации и управления в АСУТП.

Системные компоненты – программные средства, обеспечивающие функционирование среды исполнения, но не реализующие прикладную логику.

СПИСОК ЛИТЕРАТУРЫ

1. Егоров А.Ф. Интегрированные автоматизированные системы управления химическими производствами и предприятиями. – М.: Юрайт, 2025.

2. Егоров А.Ф. Интегрированные системы управления химическими производствами: учеб. Пособие / А. Ф. Егоров. – М.: РХТУ им. Д. И. Менделеева, 2020. – 200 с.

3. Кафаров В. В. Методы кибернетики в химии и химической технологии: учебник для вузов по специальности "Основные процессы химических производств и химическая кибернетика" / В. В. Кафаров. - 4-е изд., перераб. и доп. – М: Химия, 1985. - 448 с.

4. Мешалкин В.П. Основы интенсификации и ресурсоэнерго эффективности химико-технологических систем / В.П. Мешалкин - Смоленск: Универсум, 2021. – 999 с.

5. Пьявченко Т.А. Управление технологическими процессами на основе SCADA // Политематический сетевой электронный научный журнал Кубанского государственного аграрного университета. – 2017. – № 128. – С. 167-187.

6. Максимова Е.А., Грицюк С.Н. Использование SCADA-технологий в современных автоматизированных системах управления // Молодой ученый. – 2015. – Т. 5. – № 22 (102). – С. 45-48.

7. Шерстобитов Я.Е., Воробьев В.В., Лукьянов А.Д. Анализ и обзор современных решений в сфере диспетчеризации и мониторинга промышленного оборудования // European Journal of Natural History. – 2021. – № 2. – С. 97-101.

8. Сверчков Д.С. Разработка человеко-машинного интерфейса и его применение в системах управления // Труды Крыловского государственного научного центра. – 2018. – Спец. вып. 1. – С. 184–190.

9. Вейбер В.В., Кудинов А.В., Марков Н.Г. Задача сбора и передачи технологической информации распределенного промышленного предприятия // Известия Томского политехнического университета. Инжиниринг георесурсов. – 2011. – Т. 319. – № 5. – С. 69-74.

10. Бабаев Д.И., Полетыкин А.Г., Промыслов В.Г., Тимофеев М.Ю. Управление архитектурой кибербезопасности АСУТП атомных электростанций // Проблемы управления. – 2018. – № 3. – С. 47-55.

11. Дагаев Д.В. О разработке оберон-системы с заданными свойствами эргодичности // Труды Института системного программирования РАН. – 2020. – Т. 32. – № 6. – С. 67-78.

12. Вериго А.А., Цапко Г.П., Каташев А.С. Оценка уязвимостей автоматизированных систем управления технологическими процессами // Международный научно-исследовательский журнал. – 2016. – № 11-4 (53). – С. 47-49.

13. Привалов А.Н., Ларкин Е.В., Шаров В.А. Модель отказов цифровых систем управления // Известия Тульского государственного университета. Технические науки. – 2021. – № 9. – С. 346-352.

14. Лучкин Н.А. Информационная безопасность систем оперативно-диспетчерского управления технологическими процессами // Наука и современность. – 2011. – № 10-2. – С. 67-70.

15. Абдалов А.В. Методы и модели обеспечения процесса модернизации автоматизированных систем управления предприятий: дис. ... канд. техн. наук: 2.3.3. - М., 2024. - 159 с.

16. Marques P., Correia F.F. Foundational DevOps Patterns // arXiv.org. – 2023. – URL: arxiv.org/abs/2302.01053 (дата обращения: 15.09.2023).

17. Безпятый М.В. Автоматизация и оптимизация процессов разработки и развертывания в Devops: применение современных методов и инструментов // Инновации и инвестиции. – 2023. – № 7. – С. 458–464.

18. Максимов В.Ю. Преодоление трудностей наблюдаемости в микросервисной архитектуре // Инновационная наука. – 2024. – № 2-1. – С. 30-35.

19. Волович К.И. Методы и алгоритмы организации вычислительного процесса в гибридном высокопроизводительном комплексе на основе виртуальной среды исполнения: дис. ... канд. техн. наук: 05.13.15. - М., 2019. - 114 с.

20. Спицын А.А. Управление процессами миграции виртуальных машин в облачных средах на основе реализации иерархической стратегии балансировки нагрузки: дис. ... канд. техн. наук: 2.3.5. - Воронеж, 2023. - 153 с.

21. Пья Сон Ко Ко Построение и оптимизация распределенных виртуальных вычислительных систем: дис. ... канд. техн. наук: 05.13.11. - СПб., 2020. - 247 с.

22. Сивов В.В. Модели и методы обеспечения отказоустойчивости компьютерных систем бизнес-аналитики: дис. ... канд. техн. наук: 2.3.2. - СПб., 2023. - 240 с.

23. Котенок А.В. Мультиверсионная среда исполнения для отказоустойчивых программных комплексов систем управления: дис. ... канд. техн. наук: 05.13.01, 05.13.11. - Красноярск, 2009. - 119 с.

24. Бушуев С.А. Сокращение операционных издержек бизнеса с помощью Infrastructure as Code при управлении высоконагруженными системами // Вестник науки. – 2024. – Т. 1. – № 8 (77). – С. 173-184.

25. Morris K. Infrastructure as Code: Managing Servers in the Cloud. – 1st ed. – Sebastopol, CA, USA: O'Reilly Media, 2016. – 360 p. – ISBN 978-1-4919-2435-8.

26. Kabarukhin A. Methodology «Infrastructure as Code» in the operation of it infrastructure // Восточно-европейский научный журнал. – 2022. – № 6 (82). – С. 57-61.

27. Овчинников М.А., Овчинникова Е.В. Инфраструктура управляемая кодом: методология и инструменты // Вестник науки и образования Северо-Запада России. – 2015. – Т. 1. – № 3. – С. 63-67.

28. Князева Е.А. Конфигурационное управление проектами разработки программного обеспечения // Управление проектами и развитие производства. – 2005. – № 4 (16). – С. 87-93.

29. Мадорская Ю.М. Формирование оценки изменений программного обеспечения АСУП // Информатика, телекоммуникации и управление. – 2011. – № 1 (115). – С. 65-71.

30. Амбарцумян А.А., Браништов С.А. Модель технологического регламента в АСУТП // Проблемы управления. – 2008. – № 3. – С. 73-79.

31. Барчан К.А. Разработка метода имитационного компьютерного моделирования эталонного состояния и поведения SCADA-систем на основе модели акторов // Вестник Новосибирского государственного университета. Серия: Информационные технологии. – 2014. – Т. 12. – № 1. – С. 11-18.

32. Журавлев С.С., Окольнишников В.В., Рудометов С.В. Инструментальные средства отладки и тестирования программ управления АСУТП // Актуальные проблемы гуманитарных и естественных наук. – 2016. – № 1-2. – С. 49-54.

33. Артюшенко В.М., Аббасова Т.С. Особенности резервирования источников бесперебойного питания компьютерного и телекоммуникационного оборудования // Электротехнические и информационные комплексы и системы. – 2007. – Т. 3. – № 3. – С. 3.

34. Арустамов С.А., Генин М.Г. Методы снижения рисков потерь доступности программного обеспечения критических информационных систем // Научно-технический вестник информационных технологий, механики и оптики. – 2012. – № 4 (80). – С. 131-136.

35. Чипижко Д.С., Самбор М.Е. Изучение работы ЦОД на физическом уровне: с примерами оборудований и безопасности // Вестник науки и образования. – 2025. – Изучение работы цод на физическом уровне. – № 1 (156). – С. 37-44.

36. Ягьяева Л.Т., Перухин М.Ю., Обади А. Распределенная система управления // Вестник Казанского технологического университета. – 2013. – Т. 16. – № 9. – С. 291-293.

37. Анжело Н., Вирджиния А. Умная автоматизация в интересах кибербезопасности // Форсайт. – 2023. – Т. 17. – № 1. – С. 89-97.

38. Яровая Е.В. Принципы построения архитектуры программного обеспечения // E-Scio. – 2022. – № 8 (71). – С. 20-24.

39. Гудков М.С. Анализ архитектур информационных систем: монолитная и микросервисная // Вестник науки. – 2021. – Т. 2. – Анализ архитектур информационных систем. – № 1 (34). – С. 48-51.

40. Радостев Д.К., Никитина Е.Ю. Стратегия миграции программного кода из монолитной архитектуры в микросервисы // Вестник Пермского университета. Серия: Математика. Механика. Информатика. – 2021. – № 2 (53). – С. 65-68.

41. Кабарухин А.П. Выгоды перехода от монолитной к микросервисной архитектуре приложения // Проблемы современной науки и образования. – 2022. – № 1 (170). – С. 18-23.

42. Альфара Амир Юсеф Али, Королев Д.В., Зайцев К.С., Дунаев М.Е. Разработка системы мониторинга для серверного приложения // International Journal of Open Information Technologies. – 2023. – Т. 11. – № 8. – С. 24-31.

43. Царев Р.Ю. Среда исполнения мультиверсионного программного обеспечения // Программные продукты и системы. – 2007. – № 2. – С. 29-30.

44. Кугушева Д.С. Проектирование сложного программного обеспечения с использованием микросервисной архитектуры // Инновации и инвестиции. – 2020. – № 5. – С. 188-190.

45. Балес А.И. Унифицированная модель данных и ее применение в микросервисной архитектуре // Современные информационные технологии и ИТ-образование. – 2020. – Т. 16. – № 2. – С. 416-425.

46. Ткаченко Н.И., Спирин Н.А. Применение сервис-ориентированной архитектуры при интеграции систем управления технологическими процессами // Известия Томского политехнического университета. – 2010. – Т. 317. – № 5. – С. 61–67.

47. Taylor R.N., Medvidovic N., Dashofy E.M. Software Architecture: Foundations, Theory, and Practice. – Hoboken, NJ : Wiley, 2009. – 736 p. – ISBN 978-0-470-16774-8.

48. Schmidt D., Stal M., Rohnert H., Buschmann F. Pattern-Oriented Software Architecture. Vol. 2: Patterns for Concurrent and Networked Objects. – Chichester: John Wiley & Sons, 2000. – 666 p. – ISBN 978-0-471-60695-6.

49. Нагорный Н.Н. Основные аспекты разработки микросервисного веб-приложения // Международный научно-исследовательский журнал. – 2023. – № 7 (133). – С. 62.

50. Артамонов И.В. Показатели производительности микросервисных систем // Вестник НГИЭИ. – 2018. – № 8 (87). – С. 24-33.

51. Мясников И.В. Мониторинг компонентов Istio для обеспечения надежности и наблюдаемости Service Mesh // Вестник науки. – 2025. – Т. 1. – № 5 (86). – С. 734-744.

52. Маркова В.Д. Цифровизация управления: от АСУ к микросервисам // Всероссийский экономический журнал ЭКО. – 2022. – Цифровизация управления. – № 9 (579). – С. 113-129.

53. Ольхов Д.А. Анализ подходов к управлению прикладным программным интерфейсом микросервисов в облачной среде // Символ науки. – 2021. – № 3. – С. 26-32.

54. Джалалов М.Э. Стратегии управления версионностью API в микросервисной архитектуре // Экономика и качество систем связи. – 2024. – № 1 (31). – С. 136-143.

55. Жаринов В.Н. К понятию связывания структур данных и управления в моделях императивных знаний // Объектные системы. – 2015. – № 11. – С. 26-35.

56. Жаринов В.Н. Объектное описание систем на императивно-семантическом языке // Объектные системы. – 2015. – № 11. – С. 35-43.

57. Фрасын П.Г. Математическая модель управления конфигурацией программных средств в автоматизированных системах управления технологическими процессами // Инженерный вестник Дона. – 2025. – № 3. URL: ivdon.ru/ru/magazine/archive/n3y2025/9924.

58. Бабанов А.М., Петров А.В. Декларативное определение условий непротиворечивости в ERM-метасхеме // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. – 2020. – № 51. – С. 94-101.

59. Грибова В.В., Москаленко Ф.М., Тимченко В.А., Шалфеева Е.А. Создание жизнеспособных интеллектуальных систем с управляемыми декларативными компонентами // Информационные и математические технологии в науке и управлении. – 2018. – № 3 (11). – С. 6-17.

60. Кузнецов К.В., Лысенкова С.А. Краткое аннотирование новостей из области информационных технологий // Вестник кибернетики. – 2024. – Т. 23. – № 3. – С. 31-39.

61. Ильин В.Г., Цынгалев П.С., Боронников А.С. Применение CEPH в современных облачных инсталляциях // International Journal of Open Information Technologies. – 2023. – Т. 11. – № 2. – С. 44-50.

62. Логинова Л.Н., Николаев А.М., Савицкий Д.Д. Анализ и сравнение популярных гипервизоров // Молодой ученый. – 2023. – № 471. – С. 23-26.

63. Кулешов С.В., Шальнев И.О. Особенности реализации распределенной виртуальной машины при построении коммуникационной инфраструктуры общего назначения // Известия высших учебных заведений. Приборостроение. – 2023. – Т. 66. – № 2. – С. 112-117.

64. Мамонтов Д.В., Селезнев С.В. Обзор технологий виртуализации // Молодой ученый. – 2013. – № 55. – С. 60-62.

65. Гордеев А.В., Горелик Д.В. Сравнительное тестирование контейнерной и гипервизорной виртуализации // Информационно-управляющие системы. – 2018. – № 2 (93). – С. 60-66.

66. Меньшов Н.А. Использование технологии контейнеризации как компонента обеспечения информационной безопасности // Актуальные проблемы авиации и космонавтики. – 2022. – Т. 2. – С. 281-283.

67. Убеев В.Г. Анализ существующих механизмов контейнеризации в операционных системах // E-Scio. – 2022. – № 10 (73). – С. 27-33.

68. Баранов А.В., Николаев Д.С. Использование контейнерной виртуализации в организации высокопроизводительных вычислений // Программные системы: теория и приложения. – 2016. – Т. 7. – № 1 (28). – С. 117-134.

69. Холодков Д.В., Зинкин С.А. Влияния технологий контейнеризации приложений на скорость выполнения // Вестник Пензенского государственного университета. – 2024. – № 4 (48). – С. 134-136.

70. Назаренко Э.Г. Лучше поздно, чем никогда, или эффект санкций // Инновационные проекты и программы в образовании. – 2022. – № 2 (80). – С. 65-68.

71. Морозова Н.В., Мамчуев А.К., Хатуаев Т.А. Импортозамещение программного продукта в россии // Естественно-гуманитарные исследования. – 2022. – № 6 (44). – С. 367-370.

72. Кипкеева А.М., Урусов А.А. Современные проблемы развития IT-технологий в банковском секторе России // Вестник Академии знаний. – 2022. – № 6 (53). – С. 336-338.

73. Житнюк П.П. Не было бы счастья // Россия в глобальной политике. – 2024. – Т. 22. – № 1. – С. 229-245.

74. Ямгуров Р.Р. Особенности Российской защиты информации // Инновационная наука. – 2022. – № 4-2. – С. 61-63.

75. Kononova N., Grobova T., Azarova E., Kononov M., Grobova S. Features of the Astra Linux operating system // The Scientific Heritage. – 2021. – № 75-1. – С. 3-5.

76. Бобылев Е.Д. Нагрузочное тестирование решений Docker на базе отечественной ОС // Вестник науки. – 2025. – Т. 3. – № 1 (82). – С. 959-969.

77. Шейнман В., Стариков Д.Д., Тюменцев Д.В., Вавилов Г.Д. Повышение эффективности процессов разработки программного обеспечения: контейнерные технологии // Программные системы и вычислительные методы. – 2024. – Повышение эффективности процессов разработки программного обеспечения. – № 4. – С. 151-161.

78. Бондаренко А.С., Зайцев К.С. Управление контейнерами при построении распределенных систем с микросервисной архитектурой // International Journal of Open Information Technologies. – 2023. – Т. 11. – № 8. – С. 17-23.

79. Зайнабидинов Р.М. Обзор ядра Linux и его роль в современных информационных системах // Universum: технические науки. – 2024. – Т. 1. – № 3 (120). – С. 34-37.

80. Han J., Huang C., Liu J., Zhang T. An Effective Docker Image Slimming Approach Based on Source Code Data Dependency Analysis // arXiv.org. – 2025. – URL: arxiv.org/abs/2501.03736 (дата обращения: 16.02.2025).

81. Брыжинская А.В., Чернова С.В. Кратко про Docker // Теория и практика современной науки. – 2019. – № 10 (52). – С. 33-35.

82. Muzumdar P., Bhosale A., Basyal G.P., Kurian G. Navigating the Docker Ecosystem: A Comprehensive Taxonomy and Survey // Asian Journal of Research in Computer Science. – 2024. – Vol. 17, No. 1. – P. 42–61. – DOI: 10.9734/AJRCOS/2024/v17i1411.

83. Ksontini E., Mastouri M., Khalsi R., Kessentini W. Refactoring for Dockerfile Quality: A Dive into Developer Practices and Automation Potential // arXiv.org. – 2025. – URL: arxiv.org/abs/2501.14131 (дата обращения: 18.02.2025).

84. Eng K., Hindle A. Revisiting Dockerfiles in Open Source Software Over Time // arXiv.org. – 2021. – URL: arxiv.org/abs/2103.12298 (дата обращения: 20.10.2023).

85. Understanding the image layers // Docker docs URL: docs.docker.com/get-started/docker-concepts/building-images/understanding-image-layers/ (дата обращения: 25.01.2024).

86. Baresi L., Quattrocchi G., Rasi N. A Qualitative and Quantitative Analysis of Container Engines // arXiv.org. – 2023. – URL: arxiv.org/abs/2303.04080 (дата обращения: 20.10.2023).

87. Aqasizade H., Ataie E., Bastam M. Experimental Assessment of Containers Running on Top of Virtual Machines // arXiv.org. – 2024. – URL: arxiv.org/abs/2401.07539 (дата обращения: 22.04.2025).

88. Priedhorsky R., Canon R.S., Randles T., Younge A.J. Minimizing privilege for building HPC containers // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC ’21). – New York, NY: Association for Computing Machinery, 2021. – Article No. 32. – P. 1–14. – DOI: 10.1145/3458817.3476187.

89. Installing Podman and Related Utilities // Oracle Help Center URL: docs.oracle.com/en/operating-systems/oracle-linux/podman/podman-InstallingPodmanandRelatedUtilities.html (дата обращения: 15.11.2023).

90. Arango C., Dernat R., Sanabria J. Performance Evaluation of Container-based Virtualization for High Performance Computing Environments // arXiv.org. – 2017. – URL: arxiv.org/abs/1709.10140 (дата обращения: 17.08.2024).

91. Reile C., Chadha M., Hauner V., Jindal A., Hofmann B., Gerndt M. Bunk8s: Enabling Easy Integration Testing of Microservices in Kubernetes // arXiv.org. – 2022. – URL: arxiv.org/abs/2207.06811 (дата обращения: 22.10.2023).

92. Jindal A., Chadha M., Benedict S., Gerndt M. Estimating the Capacities of Function-as-a-Service Functions // arXiv.org. – 2022. – URL: arxiv.org/abs/2201.11454 (дата обращения: 23.10.2023).

93. What is Buildah? // Red Hat URL: www.redhat.com/en/topics/containers/what-is-buildah (дата обращения: 20.10.2024).

94. What is Skopeo? // Red Hat – URL: www.redhat.com/en/topics/containers/what-is-skopeo (дата обращения: 20.10.2024).

95. Haque M.U., Babar M.A. Well Begun is Half Done: An Empirical Study of Exploitability & Impact of Base-Image Vulnerabilities // arXiv.org. – 2021. – URL: arxiv.org/abs/2112.12597 (дата обращения: 22.10.2023).

96. Wang Y., Bao Q. A Code Injection Method for Rapid Docker Image Building // arXiv.org. – 2019. – URL: arxiv.org/abs/1911.07444 (дата обращения: 22.10.2023).

97. Multi-stage builds // Docker docs URL: docs.docker.com/build/building/multi-stage/ (дата обращения: 28.09.2024).

98. Колясников П.В., Силаков И.Н., Ильин Д.Ю., Гусев А.А., Никульчев Е.В. Повышение эффективности виртуального рабочего окружения распределенной разработки программ // Современные информационные технологии и ИТ-образование. – 2019. – Т. 15. – № 1. – С. 72-80.

99. Грушин Д.А., Лазарев Д.О., Фомин С.А. Кэширование данных в мультиконтейнерных системах // Труды Института системного программирования РАН. – 2019. – Т. 31. – № 6. – С. 125-144.

100. Deutschbein C., Stassinopoulos A. «Test, Build, Deploy» -- A CI/CD Framework for Open-Source Hardware Designs // arXiv.org. – 2025. – URL: arxiv.org/abs/2503.19180v2 (дата обращения: 01.05.2025).

101. Sun S., Friberg D., Staron M. «Good» and «Bad» Failures in Industrial CI/CD -- Balancing Cost and Quality Assurance // arXiv.org. – 2025. – URL: arxiv.org/abs/2504.11839 (дата обращения: 30.05.2025).

102. Фрасын П.Г. Автоматизация процесса развертывания программных технических средств систем мониторинга состояния оборудования текстильного отделочного производства и его диспетчеризации / П.Г. Фрасын, С.Л. Власов, Е.А. Рыжкова // Известия высших учебных заведений. Технология текстильной промышленности. – 2024. – № 5 (413). – С. 191-197. (Scopus)

103. Фрасын П.Г. Методология развертки CI/CD систем на примере Gitlab / П.Г. Фрасын, Е.А. Рыжкова // Материалы докладов всероссийской научно-практической конференции им. Я.В. Мильмана: Материалы докладов, Москва, 19 декабря 2023 года. – Москва: Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство), 2024. – С. 126-133.

104. Лазаренко Е.Н., Дорохина Г.В., Гаркуша Д.А. Анализ систем контроля версий // Проблемы искусственного интеллекта. – 2023. – № 2 (29). – С. 40-48.

105. Шурыгин В.Н., Сиваченко Д.А. Системы контроля версий // Вестник Московского государственного университета печати. – 2015. – № 5. – С. 103-104.

106. Wang H., Tang H., Jiang L., Shi S., Naeem M.F., Li H., Schiele B., Wang L. GiT: Towards Generalist Vision Transformer through Universal Language Interface // arXiv.org. – 2024. – URL: arxiv.org/abs/2403.09394 (дата обращения: 27.12.2024).

107. Грузин Н.А., Голубничий А. Обзор и сравнение хостингов для git-репозиториев: Bitbucket, Github и Gitlab // E-Scio. – 2021. – Обзор И Сравнение Хостингов Для Git-Репозиториев. – № 1 (52). – С. 588-596.

108. Fairbanks J., Tharigonda A., Eisty N.U. Analyzing the Effects of CI/CD on Open Source Repositories in GitHub and GitLab // arXiv.org. – 2023. – URL: arxiv.org/abs/2303.16393 (дата обращения: 20.09.2024).

109. Ríos J.C.C., Kopec-Harding K., Eraslan S., Page C., Haines R., Jay C., Embury S.M. A Methodology for Using GitLab for Software Engineering Learning Analytics // arXiv.org. – 2019. – URL: arxiv.org/abs/1903.06772 (дата обращения: 17.11.2023).

110. Фрасын П.Г., Никитин Н.В. Методологические основания выбора средств автоматизации сопровождения программной среды верхнего уровня автоматизированных систем управления технологическими объектами // Инженерный вестник Дона. – 2025. – № 7. URL: ivdon.ru/ru/magazine/archive/n7y2025/10190.

111. Фрасын П.Г. Автоконфигурирование систем опроса технических средств в области промышленной автоматизации / П.Г. Фрасын, Н.В. Никитин // Второй Международный молодежный конкурс научных проектов «Стираем границы»: сборник материалов, Москва, 20 – 22 февраля 2024 года. – Москва: Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство), 2024. – С. 254-257.

112. Фрасын П.Г. Мониторинг и диспетчеризация водозаборного узла на базе современных цифровых технологий / П.Г. Фрасын // Цифровая трансформация социальных и экономических систем – DIGITAL2025: Материалы IV международной научно-практической конференции, Москва, 31 января 2025 года. – Москва: Московский университет им. С.Ю. Витте, 2025. – С. 618-626.

113. Фрасын П.Г. Концепция клиент-серверной архитектуры при разработке современной системы диспетчеризации и мониторинга параметров технологического процесса / П.Г. Фрасын, Ю.С. Комбаров // Инновационные технологии: теория, инструменты, практика. – 2024. – Т. 1. – С. 290-296.

114. Баталин Р.Ю. Проблемы адаптации информационных систем и программного обеспечения под ОС Astra Linux // Известия Тульского государственного университета. Технические науки. – 2024. – № 7. – С. 140-142.

115. Фрасын П.Г. Анализ данных в реальном времени и обнаружение аномалий для промышленной автоматизации / П.Г. Фрасын // Студенческая научно-исследовательская лаборатория: современное состояние и перспективы: сборник

научных статей III Международной студенческой междисциплинарной научно-практической конференции, Краснодар, 17 – 18 декабря 2024 года. – Краснодар: Академия маркетинга и социально-информационных технологий – ИМСИТ, 2024. – С. 907-911.

116. Фрасын П.Г. Применение MQTT сервера в качестве агрегатора данных между компонентами системы мониторинга и диспетчеризации / П.Г. Фрасын, С.Л. Власов, Н.В. Никитин [и др.] // Сборник научных трудов кафедры автоматики и промышленной электроники Российского государственного университета им. А.Н. Косыгина: Сборник научных трудов. – Москва: Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство), 2024. – С. 80-84.

117. Бочкарева А.В., Минкин А.В. Анализ языков программирования для первого изучения // Форум молодых ученых. – 2018. – № 10 (26). – С. 193-195.

118. Кирдяев М.М. Обзор языка программирования Python для решения задач математического моделирования // Труды Международного симпозиума «Надежность и качество». – 2016. – Т. 1. – С. 305-307.

119. Мадиярбекова А. Python как высокоуровневый язык программирования // Вестник науки. – 2024. – Т. 4. – № 11 (80). – С. 1114-1126.

120. Свидетельство о государственной регистрации программы для ЭВМ № 2024615327 Российская Федерация. Рсарiра: № 2023683619 : заявл. 09.11.2023 : опубл. 05.03.2024 / П.Г. Фрасын, Н.В. Никитин, Е.А. Рыжкова, Д.В. Масанов ; заявитель федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина».

121. Фрасын П.Г., Никитин Н.В., Масанов Д.В., Рыжкова Е.А. Методологические основы работы с протоколом Modbus TCP с примером на высокоуровневом языке программирования Python // Инженерный вестник Дона. – 2023. – № 11. URL: ivdon.ru/ru/magazine/archive/n11y2023/8785.

122. Свидетельство о государственной регистрации программы для ЭВМ № 2025614578 Российская Федерация. Modus Reader : заявл. 12.02.2025 : опубл. 24.02.2025 / П.Г. Фрасын, Н.В. Никитин, Е.А. Рыжкова, С.Л. Власов ; заявитель

федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина».

123. Шульман В.Д., Шабанов В.В., Сухов П.А., Чунихин А.О. Сравнительный анализ форматов сериализации и передачи данных JSON, XML, CBOR и GPB // StudNet. – 2021. – Т. 4. – № 7. – С. 1686-1696.

124. Городничев М.Г., Кочупалов А.Е. Исследование методов межпроцессного взаимодействия в информационной системе с горизонтальным взаимодействием // Вестник евразийской науки. – 2018. – Т. 10. – № 4. – С. 39.

125. Свидетельство о государственной регистрации программы для ЭВМ № 2025614580 Российская Федерация. ТопикЭкспортер : заявл. 12.02.2025 : опубл. 24.02.2025 / П.Г. Фрасын, Н.В. Никитин, Е.А. Рыжкова, С.Л. Власов ; заявитель федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина».

126. Свидетельство о государственной регистрации программы для ЭВМ № 2022663490 Российская Федерация. Система централизованного сбора данных в сложном технологическом процессе : № 2022662651 : заявл. 06.07.2022 : опубл. 15.07.2022 / П.Г. Фрасын, Д.В. Масанов ; заявитель федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина».

127. Hellerstein J.M. Looking Back at Postgres // arXiv.org. – 2019. – URL: arxiv.org/abs/1901.01973 (дата обращения: 20.09.2023).

128. Рогов Е.В. PostgreSQL 17 изнутри. – М.: ДМК Пресс, 2025. – 668 с.

129. Никитин Н.В., Фрасын П.Г., Масанов Д.В. Анализ методов интеграции моделей машинного обучения в SCADA-системы // Инженерный вестник Дона. – 2025. – № 6. URL: ivdon.ru/ru/magazine/archive/n6y2025/10143.

130. Суханов В.И. Программные платформы для разработки горной ГИС // Проблемы недропользования. – 2018. – № 4 (19). – С. 51-60.

131. Свидетельство о государственной регистрации программы для ЭВМ № 2025614303 Российская Федерация. ОператорХаб : заявл. 12.02.2025 : опубл. 20.02.2025 / П.Г. Фрасын, Н.В. Никитин, Е.А. Рыжкова, С.Л. Власов ; заявитель

федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина».

132. Фрасын П.Г. Исследование и применение недокументированных возможностей устройств умного дома / П.Г. Фрасын, С.Л. Власов, Е.А. Рыжкова // Сборник научных трудов кафедры автоматики и промышленной электроники Российского государственного университета им. А.Н. Косыгина: Сборник статей. – Москва: Федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство)», 2023. – С. 59-62.

133. Фрасын П.Г. Проксирование запросов к Rcon-консоли через http / П.Г. Фрасын, Н.В. Никитин, Е.А. Рыжкова // Сборник научных трудов кафедры автоматики и промышленной электроники Российского государственного университета им. А.Н. Косыгина: Сборник статей. – Москва: Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство), 2025. – С. 29-32.

134. Свидетельство о государственной регистрации программы для ЭВМ № 2025614299 Российская Федерация. ГеоРепорт : заявл. 12.02.2025 : опубл. 20.02.2025 / П.Г. Фрасын, Н.В. Никитин, Е.А. Рыжкова, С.Л. Власов ; заявитель федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина».

135. Eng K., Hindle A., Stroulia E. Patterns of Multi-Container Composition for Service Orchestration with Docker Compose // Empirical Software Engineering. – 2024. – Vol. 29, No. 3. – P. 65.

136. Ansible playbooks – Ansible Community Documentation. – URL: docs.ansible.com/ansible/latest/playbook_guide/playbooks_intro.html (дата обращения: 22.09.2024).

137. Kocsis Z.A., Swan J. Dependency Injection for Programming by Optimization // arXiv.org. – 2017. – URL: arxiv.org/abs/1707.04016 (дата обращения: 22.09.2024).

138. Kandpal N., Lester B., Muqeeth M., Mascarenhas A., Evans M., Baskaran V., Huang T., Liu H., Raffel C. Git-Theta: A Git Extension for Collaborative Development

of Machine Learning Models // arXiv.org. – 2023. – URL: arxiv.org/abs/2306.04529 (дата обращения: 12.09.2023).

139. Rosa G., Scalabrino S., Oliveto R. Fixing Dockerfile Smells: An Empirical Study // arXiv.org. – 2022. – URL: arxiv.org/abs/2208.09097 (дата обращения: 11.04.2023).

140. Priedhorsky R., Jennings M., Phinney M. Zero-consistency root emulation for unprivileged container image build // arXiv.org. – 2024. – URL: arxiv.org/abs/2405.06085 (дата обращения: 12.12.2024).

141. Matsumoto N., Suda A. bypass4netns: Accelerating TCP/IP Communications in Rootless Containers // arXiv.org. – 2024. – URL: arxiv.org/abs/2402.00365 (дата обращения: 16.04.2024).

142. Шляпников В.М. Ускорение непрерывной интеграции и развертывания Python-приложений // Инновационные аспекты развития науки и техники. – 2021. – № 2. – С. 71-79.

143. Doan T.-P., Jung S. DAVS: Dockerfile Analysis for Container Image Vulnerability Scanning // Computers, Materials & Continua. – 2022. – Vol. 72. – DAVS. – No. 1. – P. 1699-1711.

144. Израилов К.Е. Моделирование программы с уязвимостями с позиции эволюции ее представлений. Часть 1. Схема жизненного цикла // Труды учебных заведений связи. – 2023. – Т. 9. – № 1. – С. 75-93.

145. Belov R. Modern methodologies for organizing DevOps processes in web development // Вестник науки. – 2025. – Т. 2. – № 5 (86). – С. 722-735.

146. Dahlmanns M., Sander C., Decker R., Wehrle K. Secrets Revealed in Container Images: An Internet-wide Study on Occurrence and Impact // Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security (AsiaCCS '23). – New York, NY: Association for Computing Machinery, 2023. – P. 797–811. – DOI: 10.1145/3579856.3590329.

147. Аканов Т.Т. Безопасность сайтов: SQL - инъекция // Вестник науки. – 2019. – Т. 4. – Безопасность Сайтов. – № 6 (15). – С. 265-268.

148. Zerouali A., Mens T., Robles G., Gonzalez-Barahona J. On The Relation Between Outdated Docker Containers, Severity Vulnerabilities and Bugs // arXiv.org. – 2018. – URL: arxiv.org/abs/1811.12874 (дата обращения: 10.09.2024).

149. Nousias A., Katsaros E., Syrmos E., Radoglou-Grammatikis P., Lagkas T., Argyriou V., Moscholios I., Markakis E., Goudos S., Sarigiannidis P. Malware Detection in Docker Containers: An Image is Worth a Thousand Logs // arXiv.org. – 2025. – URL: arxiv.org/abs/2504.03238 (дата обращения: 25.05.2025).

150. Zhou Y., Zhan W., Li Z., Han T., Chen T., Gall H. DRIVE: Dockerfile Rule Mining and Violation Detection // arXiv.org. – 2023. – URL: arxiv.org/abs/2212.05648v3 (дата обращения: 11.01.2024).

151. Shabani T., Nashid N., Alian P., Mesbah A. Dockerfile Flakiness: Characterization and Repair // arXiv.org. – 2025. – URL: arxiv.org/abs/2408.05379 (дата обращения: 12.03.2025).

152. CVE-2025-29087 Detail // National Vulnerability Database URL: nvd.nist.gov/vuln/detail/CVE-2025-29087 (дата обращения: 20.09.2024).

153. Pan Z., Shen W., Wang X., Yang Y., Chang R., Liu Y., Liu C., Liu Y., Ren K. Ambush from All Sides: Understanding Security Threats in Open-Source Software CI/CD Pipelines // IEEE Transactions on Dependable and Secure Computing. – 2024. – Vol. 21, No. 1. – P. 403–418. – DOI: 10.1109/TDSC.2023.3253572.

154. Gião H. da, Flores A., Pereira R., Cunha J. Chronicles of CI/CD: A Deep Dive into its Usage Over Time // arXiv.org. – 2024. – URL: arxiv.org/abs/2402.17588 (дата обращения: 15.04.2024).

155. Трубачева С.И. Почему Linux и системы реального времени? // Вестник Волжского университета им. В. Н. Татищева. – 2015. – № 2 (24). – С. 99-105.

156. Никитин Н.В. Исследование эффективности работы систем обезжелезивания воды: анализ данных и выявление закономерностей / Н.В. Никитин, П.Г. Фрасын, Д.В. Масанов // Сборник научных трудов кафедры автоматики и промышленной электроники Российского государственного университета им. А.Н. Косыгина: Сборник статей. – Москва: Российский

государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство), 2025. – С. 164-172.

157. Постолаки Е.С. Анализ применения различных типов электродвигателей в многодвигательных системах / Е.С. Постолаки, Д.В. Масанов, П.Г. Фрасын // Сборник научных трудов кафедры автоматики и промышленной электроники Российского государственного университета им. А.Н. Косыгина: Сборник научных трудов. – Москва: Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство), 2024. – С. 116-120.

158. Иванов М.С. Технические решения оптимального управления сложными многомерными динамическими объектами технологического оборудования / М.С. Иванов, П.Г. Фрасын, Ю.С. Комбаров // Известия высших учебных заведений. Технология текстильной промышленности. – 2024. – № 4 (412). – С. 185-189. (Scopus)

ПРИЛОЖЕНИЯ

А. Акт о внедрении результатов диссертационной работы в ООО «АК-Системы»

Б. Акты о внедрении результатов диссертационной работы в ООО «Самолет-Ресурс»

В. Свидетельство о государственной регистрации программы для ЭВМ 2024615327 «PCAPIPA»

Г. Свидетельство о государственной регистрации программы для ЭВМ 2022663490 «Системы централизованного сбора данных в сложном технологическом процессе»

Д. Свидетельство о государственной регистрации программы для ЭВМ 2025614303 «ОператорХаб»

Е. Свидетельство о государственной регистрации программы для ЭВМ 2025614299 «Георепорт»

Ж. Свидетельство о государственной регистрации программы для ЭВМ 2025614578 «Modbus Reader»

З. Свидетельство о государственной регистрации программы для ЭВМ 2025614580 «ТопикЭкспортер»

Приложение А. Акт о внедрении результатов диссертационной работы в ООО «АК-Системы»

«УТВЕРЖДАЮ»
Генеральный директор
ООО «АК Системы»
_____ С. Н. Новченко
«03» января 2025г.

АКТ

о внедрении результатов диссертационной работы Фрасын П.Г.
«Разработка методов управления программной средой автоматизированных систем управления технологическими процессами»

ООО «АК Системы» подтверждает применение результатов указанной диссертационной работы при эксплуатации и модернизации программной среды верхнего уровня систем мониторинга и диспетчеризации на водозаборных узлах «Томилино Парк», «Ям» и «Коробово». Разработанные методические материалы и процедуры по управлению жизненным циклом программной среды верхнего уровня включены в корпоративные регламенты и применяются на практике.

По данным журналов сопровождения и контрольных отработок за период внедрения, плановые работы по обновлению и перенастройке выполняются в регламентированном порядке без нарушения наблюдаемости технологического процесса и без вывода из эксплуатации операторских интерфейсов. Типовые технологические окна при обновлениях укладываются в 3–5 минут; первичное развертывание стандартной конфигурации — 9–10 минут; операции изменения структуры и миграции данных — 10–12 минут; проверка готовности и восстановления из резервной копии сопоставимых объемов — 20–25 минут.

В сравнении с ранее применявшимися практиками на сопоставимых объектах отмечено сокращение длительности плановых окон обновлений в среднем на 45 %, первичного развертывания — на 40 %, работ по структурным изменениям и миграциям — на 35 %; время регламентных процедур восстановления по резервной копии уменьшено на 30–40 %.

Доля операций, выполняемых дистанционно в рамках утвержденных регламентов, возросла, что сопровождалось снижением числа внеплановых выездов эксплуатационного персонала на 25–30 %.

По итогам периода эксплуатации зафиксировано повышение фактической доступности интерфейсов диспетчеризации на ~0,3–0,5 п.п. при сохранении целостности архивов технологических данных в ходе выполняемых работ.

Отмечаем улучшение предсказуемости и управляемости операций сопровождения, упорядочение учета версий и повышение наблюдаемости выполняемых действий. Внедренные решения применяются подразделениями ООО «АК Системы» в текущей эксплуатации указанных объектов и учитываются при проектировании и вводе в работу аналогичных систем.

Генеральный директор
ООО «АК Системы» Новченко С.Н.

Приложение Б. Акты о внедрении результатов диссертационной работы в ООО «Самолет-Ресурс»

«УТВЕРЖДАЮ»
Главный инженер
ООО «Самолет-Ресурс»
_____ П. М. Драница
«08» апреля 2025.

АКТ

о внедрении результатов диссертационной работы Фрасын П.Г.
«Разработка методов управления программной средой автоматизированных систем управления технологическими процессами»

ООО «Самолет-Ресурс» подтверждает применение результатов указанной работы на объекте «Томилино Парк» в части диспетчерского наблюдения и ведения технологических архивов. Внедрение выполнялось совместно с ООО «АК Системы» и отражено в действующих эксплуатационных документах.

По наблюдениям технологической службы, в период плановых работ экраны и тренды остаются доступными, тревожная индикация и звук не отключаются; по сменным журналам пропусков трендов в окна работ не отмечено. Переход объекта в ручной режим не требовался: короткие регламентные окна, как правило, до 5 минут, укладываются в штатные смены без перестройки режимов.

После перехода на принятый порядок сопровождения снизились обращения по сменам по тематике «не обновляется экран / пропал тренд / не подгружаются уставки»: за первый квартал — 8 обращений вместо 13 за сопоставимый период прошлого года. Время подготовки ежемесячных технологических сводок (дебиты, давления, уровни) сократилось примерно на 20 %: ориентировочно с ~5 часов до ~4 часов за счет стабильных выгрузок без возвратов.

Стандартизированы единицы измерения и шкалы по ключевым параметрам (уровень, давление, расход), что устранило ручные пересчеты и расхождения после обновлений; доля неинформативных тревог после плановых работ снизилась примерно на 15 %, операторы меньше отвлекаются на некритичные сигналы. Выборочные контрольные выгрузки показали полноту архивов ≥ 99,8 % по проверяемым периодам; корректировки «вручную» не потребовались.

Дополнительно налажен ввод результатов лабораторного контроля (хлор, железо) с привязкой времени в общий архив — сверка с технологическими событиями стала наглядней, замечания от смен по «несходу данных» не поступали.

Считаем принятый порядок сопровождения полезным для качества технологического управления и поддерживаем его применение на аналогичных объектах компании.

Главный специалист – технолог
ООО «Самолет-Ресурс» Драница П.М.

«УТВЕРЖДАЮ»
Главный инженер
ООО «Самолет-Ресурс»
_____ Д. В. Русаков
«08» _____ 2025г.

АКТ

о внедрении результатов диссертационной работы Фрасын П.Г.
«Разработка методов управления программной средой автоматизированных систем управления технологическими процессами»

ООО «Самолет-Ресурс» подтверждает применение результатов указанной работы на объекте водоснабжения «Томилино Парк» при сопровождении программной части диспетчеризации. Внедрение выполнено совместно с ООО «АК Системы» и закреплено в наших эксплуатационных регламентах.

По факту эксплуатации: плановые обновления и перенастройка программ выполняются в рабочем порядке без перевода объекта в ручной режим. Окна работ короткие: по журналам площадки укладываются не более чем в 6 минут. Первичная подготовка стандартной конфигурации на резервной машине заняла до 12 минут. Контрольное восстановление из резервной копии на тестовом узле проводили — по времени около получаса с проверкой экранов и архивов.

После кратковременных перебоев электропитания и сетевых сбоев система диспетчеризации восстанавливалась автоматически, вмешательство дежурного инженера не требовалось. По итогам квартала уменьшилось число вызовов ИТР на объект по программной части — примерно на четверть относительно прошлого года; внезапных простоев интерфейсов в период регламентов не зафиксировано.

Отмечаем удобство согласования работ: заранее получаем окно, объем и ответственных; незапланированных вмешательств не наблюдалось. Отчеты по проведенным операциям и контрольные листы приложены к сменным журналам.

Считаем результаты значимыми для устойчивой эксплуатации и рекомендуем подобный порядок сопровождения применять на прочих площадках компании.

Главный инженер
ООО «Самолет-Ресурс» Д. В. Русаков

Приложение В. Свидетельство о государственной регистрации программы для ЭВМ 2024615327 «PCAPIPA»

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2024615327

PCAPIPA

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство)» (RU)***

Авторы: ***Фрасын Павел Геннадьевич (RU), Никитин Никита Валерьевич (RU), Рыжкова Елена Александровна (RU), Масанов Дмитрий Викторович (RU)***

Заявка № **2023683619**
Дата поступления **09 ноября 2023 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***05 марта 2024 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Ю.С. Зубов

Приложение Г. Свидетельство о государственной регистрации программы для ЭВМ 2022663490 «Системы централизованного сбора данных в сложном технологическом процессе»

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2022663490

Система централизованного сбора данных в сложном технологическом процессе

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство)» (RU)***

Авторы: ***Фрасын Павел Геннадьевич (RU), Масанов Дмитрий Викторович (RU)***

Заявка № **2022662651**
Дата поступления **06 июля 2022 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***15 июля 2022 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Ю.С. Зубов

Приложение Д. Свидетельство о государственной регистрации программы для ЭВМ 2025614303 «ОператорХаб»

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2025614303

ОператорХаб

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство)» (RU)***

Авторы: ***Фрасын Павел Геннадьевич (RU), Никитин Никита Валерьевич (RU), Рыжкова Елена Александровна (RU), Власов Святослав Львович (RU)***

Заявка № **2025612510**
Дата поступления **12 февраля 2025 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***20 февраля 2025 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Ю.С. Зубов

Приложение Е. Свидетельство о государственной регистрации программы для ЭВМ 2025614299 «Георепорт»

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2025614299

ГеоРепорт

Правообладатель: ***федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство) (RU)***

Авторы: ***Фрасын Павел Геннадьевич (RU), Никитин Никита Валерьевич (RU), Рыжкова Елена Александровна (RU), Власов Святослав Львович (RU)***

Заявка № **2025612513**
Дата поступления **12 февраля 2025 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***20 февраля 2025 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Ю.С. Зубов

Приложение Ж. Свидетельство о государственной регистрации программы для ЭВМ 2025614578 «Modbus Reader»

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2025614578

Modbus Reader

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство)» (RU)***

Авторы: ***Фрасын Павел Геннадьевич (RU), Никитин Никита Валерьевич (RU), Рыжкова Елена Александровна (RU), Власов Святослав Львович (RU)***

Заявка № **2025612542**
Дата поступления **12 февраля 2025 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***24 февраля 2025 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Ю.С. Зубов

Приложение З. Свидетельство о государственной регистрации программы для ЭВМ 2025614580 «ТопикЭкспортер»

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2025614580

ТопикЭкспортер

Правообладатель: ***Федеральное государственное бюджетное образовательное учреждение высшего образования «Российский государственный университет им. А.Н. Косыгина (Технологии. Дизайн. Искусство)» (RU)***

Авторы: ***Фрасын Павел Геннадьевич (RU), Никитин Никита Валерьевич (RU), Рыжкова Елена Александровна (RU), Власов Святослав Львович (RU)***

Заявка № **2025612544**
Дата поступления **12 февраля 2025 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***24 февраля 2025 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Ю.С. Зубов