ПОДСИСТЕМА ВИЗУАЛИЗАЦИИ И НАСТРОЙКИ ИНТЕГРАЦИОННЫХ КОМПОНЕНТОВ ДЛЯ АСУ ТП: АРХИТЕКТУРНЫЕ РЕШЕНИЯ И ПРОТОТИП

Р. Д. Гимранов[1,а], **В. А. Галкин**[2,3,б], **Т. В. Гавриленко**[2,3,в], **Д. А. Моргун**[2,г], **М. Т. Гавриленко**[2,3,д], **М. Е. Амелин**[2,3,е], **С. А. Батуро**[2,3,ж], **А. И. Грюнталь**[4,з]

[1] *ПАО «Сургутнефтегаз», г. Сургут, Российская Федерация*

[2] *Сургутский филиал федерального государственного бюджетного учреждения «Федеральный научный центр Научно-исследовательский институт системных исследований Национального исследовательского центра «Курчатовский институт», г. Сургут, Российская Федерация*

[3] *Сургутский государственный университет, г. Сургут, Российская Федерация*

[4] *Федеральное государственное бюджетное учреждение «Федеральный научный центр Научно-исследовательский институт системных исследований Национального исследовательского центра «Курчатовский институт», г. Москва, Российская Федерация*

[а] *ORCID: https://orcid.org/0000-0002-9928-8416, gimranov_rd@mail.ru*

[б] *ORCID: https://orcid.org/0000-0002-9721-4026, val-gal@yandex.ru*

[в] *ORCID: https://orcid.org/0000-0002-3243-2751, taras_gavrilenko@office.niisi.tech*

[г] *ORCID: https://orcid.org/0000-0003-0692-1583,* ✍ *morgun_da@office.niisi.tech*

[д] *ORCID: https://orcid.org/0009-0008-4789-7376, gavrilenko.michail.t@office.niisi.tech*

[е] *ORCID: https://orcid.org/0009-0008-9831-2112, amelin_me@office.niisi.tech*

[ж] *ORCID: https://orcid.org/0009-0008-9270-9207, baturo_sa@office.niisi.tech*

[з] *grntl@niisi.ras.ru*

Аннотация: в статье представлены результаты разработки подсистемы визуализации и настройки интеграционных компонентов для автоматизированных систем управления технологическими процессами (АСУ ТП). Проведен сравнительный анализ архитектурных подходов к построению настольных приложений: слоистой архитектуры, чистой и гексагональной архитектур, модульного монолита и плагинно-ориентированной архитектуры. Обоснован выбор модульного монолита как оптимального решения для данного класса задач. Описан разработанный прототип – «Конфигуратор интеграционных компонентов», реализующий визуальное проектирование конфигураций интеграционных компонентов с поддержкой семи типов компонентов, трех протоколов взаимодействия и нескольких форматов экспорта. Рассмотрены вопросы хранения данных в PostgreSQL, организации API на базе FastAPI и обеспечения межкомпонентной интеграции.

Ключевые слова: АСУ ТП, интеграционные компоненты, модульный монолит, подсистема визуализации, GTK, FastAPI, PostgreSQL, UML, BPMN, OWL, DFD, архитектура программного обеспечения, технологический суверенитет.

Благодарности: работа выполнена в рамках НИОКР «Разработка прототипа программной платформы генерации прикладных систем управления» № 1814 от 28.12.2024.

Для цитирования: Гимранов Р. Д., Галкин В. А., Гавриленко Т. В., Моргун Д. А., Гавриленко М. Т., Амелин М. Е., Батуро С. А., Грюнталь А. И. Подсистема визуализации и настройки интеграционных компонентов для АСУ ТП: архитектурные решения и прототип. *Успехи кибернетики.* 2026;7(3):8–16.

Поступила в редакцию: 16.08.2026. *В окончательном варианте: 23.09.2026.*

VISUALIZATION AND CONFIGURATION SUBSYSTEM FOR INTEGRATION COMPONENTS IN PROCESS CONTROL SYSTEMS: ARCHITECTURE AND PROTOTYPE

R. D. Gimranov[1,a], **V. A. Galkin**[2,3,b], **T. V. Gavrilenko**[2,3,c], **D. A. Morgun**[2,d], **M. T. Gavrilenko**[2,3,e], **M. E. Amelin**[2,3,f], **S. A. Baturo**[2,3,g], **A. I. Gryuntal**[4,h]

[1] *"Surgutneftegas" Public Joint Stock Company, Surgut, Russian Federation*

[2] *Surgut Branch of Scientific Research Institute for System Analysis of the National Research Centre "Kurchatov Institute", Surgut, Russian Federation*

[3] *Surgut State University, Surgut, Russian Federation*

[4] *Scientific Research Institute for System Analysis of the National Research Centre "Kurchatov Institute", Moscow, Russian Federation*

[a] ORCID: https://orcid.org/0000-0002-9928-8416, gimranov_rd@mail.ru
[b] ORCID: https://orcid.org/0000-0002-9721-4026, val-gal@yandex.ru
[c] ORCID: https://orcid.org/0000-0002-3243-2751, taras_gavrilenko@office.niisi.tech
[d] ORCID: https://orcid.org/0000-0003-0692-1583, ✍ morgun_da@office.niisi.tech
[e] ORCID: https://orcid.org/0009-0008-4789-7376, gavrilenko.michail.t@office.niisi.tech
[f] ORCID: https://orcid.org/0009-0008-9831-2112, amelin_me@office.niisi.tech
[g] ORCID: https://orcid.org/0009-0008-9270-9207, baturo_sa@office.niisi.tech
[h] grntl@niisi.ras.ru

Abstract: we developed a subsystem for visualizing and configuring integration components in automated process control systems (PCS). We compared several architectural approaches to desktop application development, including layered, clean, and hexagonal architectures, as well as the modular monolith and plug-in architectures. Based on this comparison, we selected the modular monolith architecture for the subsystem. We developed a prototype, the Integration Component Configurator, which supports visual configuration of integration components, seven component types, three interaction protocols, and several export formats. We also developed a data storage system based on PostgreSQL and an API based on FastAPI, and implemented mechanisms for inter-component integration. The results demonstrate the feasibility of the proposed architecture and prototype for configuring and integrating components in PCS.

Keywords: process control system, integration components, modular monolith, visualization subsystem, GTK, FastAPI, PostgreSQL, UML, BPMN, OWL, DFD, software architecture, technological sovereignty.

Acknowledgements: the work was performed as part of the R&D “Development of a prototype software platform for generating applied control systems” No. 1814 dated 12/28/2024.

Cite this article: Gimranov R. D., Galkin V. A., Gavrilenko T. V., Morgun D. A., Gavrilenko M. T., Amelin M. E., Baturo S. A., Gryuntal A. I. Visualization and Configuration Subsystem for Integration Components in Process Control Systems: Architecture and Prototype. *Russian Journal of Cybernetics*. 2026;7(3):8–16.

Original article submitted: 16.08.2026. *Revision submitted: 23.09.2026.*

Введение

Современные автоматизированные системы управления технологическими процессами нефтегазовой отрасли представляют собой сложные комплексы, объединяющие разнородное оборудование, протоколы передачи данных и прикладное программное обеспечение. Ключевым элементом таких систем являются интеграционные компоненты — программные элементы, реализующие преобразование форматов и протоколов на границах взаимодействия подсистем [1, 2].

В условиях обеспечения технологического суверенитета задача разработки инструментальных средств проектирования и конфигурирования интеграционных компонентов приобретает особую актуальность. Специалист по АСУ ТП должен иметь возможность описать структуру интеграционной конфигурации декларативно, то есть без погружения в детали программной реализации каждого компонента. Подходы к онтологиям были определены в первой статье «Концепция платформы генерации прикладных систем управления» [3].

Настоящая статья посвящена разработке прототипа подсистемы визуализации и настройки интеграционных компонентов: анализу архитектурных подходов, реализации настольного приложения «Конфигуратор интеграционных компонентов» на Python/GTK, а также API для его интеграции с внешними подсистемами обработки.

Анализ архитектурных подходов

Выбор архитектуры программного обеспечения является одним из наиболее критичных решений на этапе проектирования системы. Архитектурные решения напрямую влияют на поддерживаемость, тестируемость, расширяемость и понятность программного кода [4, 5]. При выборе архитектуры разрабатываемого конфигуратора учитывались следующие аспекты: необходимость четкого разделения бизнес-логики и представления, а также потенциал для будущего расширения — добавления новых типов интеграционных компонентов, форматов экспорта и интеграций с внешними системами.

Слоистая архитектура

Слоистая архитектура предполагает организацию компонентов в горизонтальные слои представления, бизнес-логики и доступа к данным [4]. Каждый слой взаимодействует только с соседними. Достоинство подхода — простота, однако жесткая связанность слоев порождает проблему при разработке визуального редактора: часть бизнес-логики неизбежно «перетекает» в слой представления и соблюсти принцип однонаправленности практически невозможно.

Чистая и гексагональная архитектуры

Чистая архитектура Мартина [6] и гексагональная архитектура Коберна [7] выносят доменную логику в независимое ядро. Внешние слои реализуют абстрактные интерфейсы (порты), что обеспечивает тестируемость и гибкость замены технологий. Гексагональная архитектура рассматривает систему как ядро бизнес-логики, окруженное реализациями портов взаимодействия с внешним миром. Интеграционный компонент рассматривается как настраиваемый элемент визуальной модели, задающий тип, параметры, протокол взаимодействия и связи с другими элементами схемы. Основной недостаток: необходимость проектировать множество интерфейсов и реализаций портов избыточна при фиксированном наборе функций.

Модульный монолит

Модульный монолит – единое развертываемое приложение, внутренне организованное в слабосвязанные модули с четкими публичными интерфейсами [8]. Для конфигуратора выделены четыре модуля:

- доменный модуль — сущности предметной области (интеграционный компонент, соединение, проект) и бизнес-правила, независимые от технических деталей;
- модуль инфраструктуры — работа с файловой системой, сериализация JSON, взаимодействие с PostgreSQL;
- модуль пользовательского интерфейса — компоненты GTK, отрисовка холста;
- модуль прикладных сценариев — координация взаимодействия остальных модулей, реализация use-cases.

Зависимости направлены от внешних модулей к внутренним: доменный модуль ни от чего не зависит. Это гарантирует изоляцию бизнес-логики и упрощает ее тестирование.

Плагинно-ориентированная архитектура

Плагинно-ориентированная (микроядерная) архитектура [9] обеспечивает максимальную расширяемость через динамически подключаемые плагины. Для прототипа с фиксированным набором типов интеграционных компонентов такой подход избыточен. При дальнейшем развитии и необходимости поддержки расширений от сторонних разработчиков этот подход может быть рассмотрен.

Сравнительный анализ и выбор

Результаты сравнения подходов по ключевым критериям представлены в таблице 1.

Таблица 1

Сравнение архитектурных подходов

Архитектура	Простота	Тестируемость	Расширяемость	Применимость
Слоистая	Высокая	Средняя	Средняя	Средняя
Чистая/ гексагональная	Низкая	Высокая	Высокая	Низкая
Модульный монолит	Средняя	Высокая	Высокая	Высокая
Плагинно-ориентированная	Низкая	Средняя	Максимальная	Низкая

По совокупности критериев выбран **модульный монолит**. Он обеспечивает оптимальный баланс структурированности и простоты реализации.

Прототип подсистемы визуализации и настройки

Разработанный прототип — «Конфигуратор интеграционных компонентов» — настольное приложение на Python с использованием библиотеки GTK 4. Приложение реализует принципы визуального программирования [10].

Архитектура и состав модулей

Структура приложения строго следует принципам модульного монолита. Точка входа (*main.py*) инициализирует приложение и систему аутентификации. Компоненты интерфейса сосредоточены в пакете *ui/*, модели данных — в *models/*, утилиты — в *utils/*, слой БД — в *database/*.

Ключевые модули:

- ui/windows/main_window.py — главное рабочее окно, трехпанельная компоновка;
- ui/widgets/canvas.py — масштабируемая рабочая область с поддержкой drag-and-drop;
- models/integration_component.py, models/connection.py, models/project.py — доменные сущности;
- utils/uml_exporter.py — экспорт в PlantUML / текстовый UML / XMI;
- database/database.py — слой доступа к PostgreSQL.

Главное окно и рабочая область

Главное окно реализует трехпанельную компоновку (рисунок 1). Левая панель разделена на каталог типов интеграционных компонентов (рисунок 2) и список созданных интеграционных компонентов текущего проекта. Центральная область — масштабируемый холст с привязкой к сетке. Правая панель отображает детали выбранного интеграционного компонента.

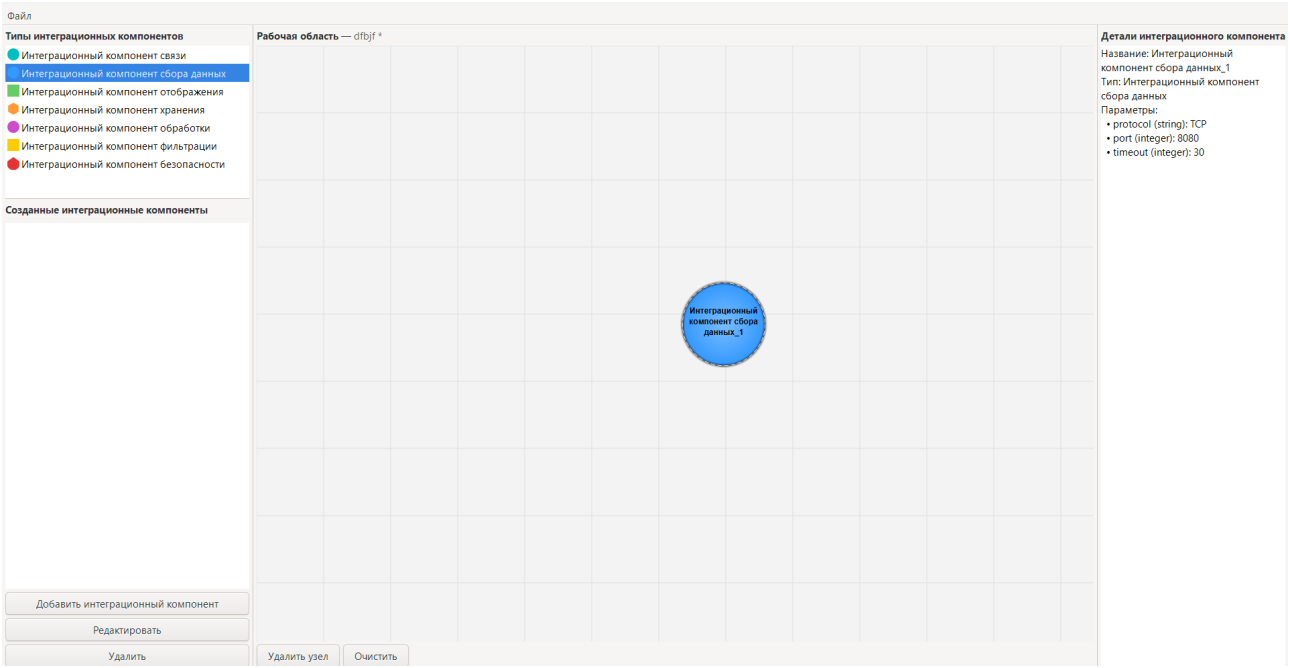


Рис. 1. *Главное окно «Конфигуратора интеграционных компонентов»: каталог типов (слева), холст (центр), детали компонента (справа)*

Система связей

Реализованы три типа направленных связей между интеграционными компонентами. Каждая связь визуально отображается стрелкой с подписью на холсте (рисунок 3).

Параметры каждой связи настраиваются в специализированном диалоге (рисунок 4). Поддерживается добавление пользовательских параметров без ограничений.

Управление проектами и хранение данных

Приложение реализует: создание, открытие, сохранение. Состояние несохраненных изменений отображается символом «*» в заголовке окна.

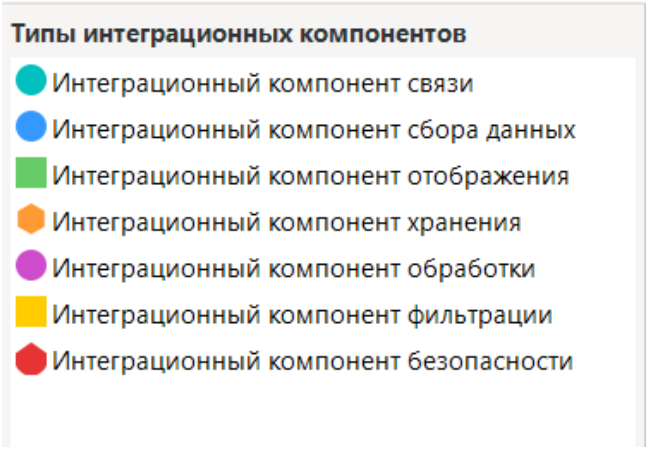


Рис. 2. *Цветовая кодировка типов интеграционных компонентов*

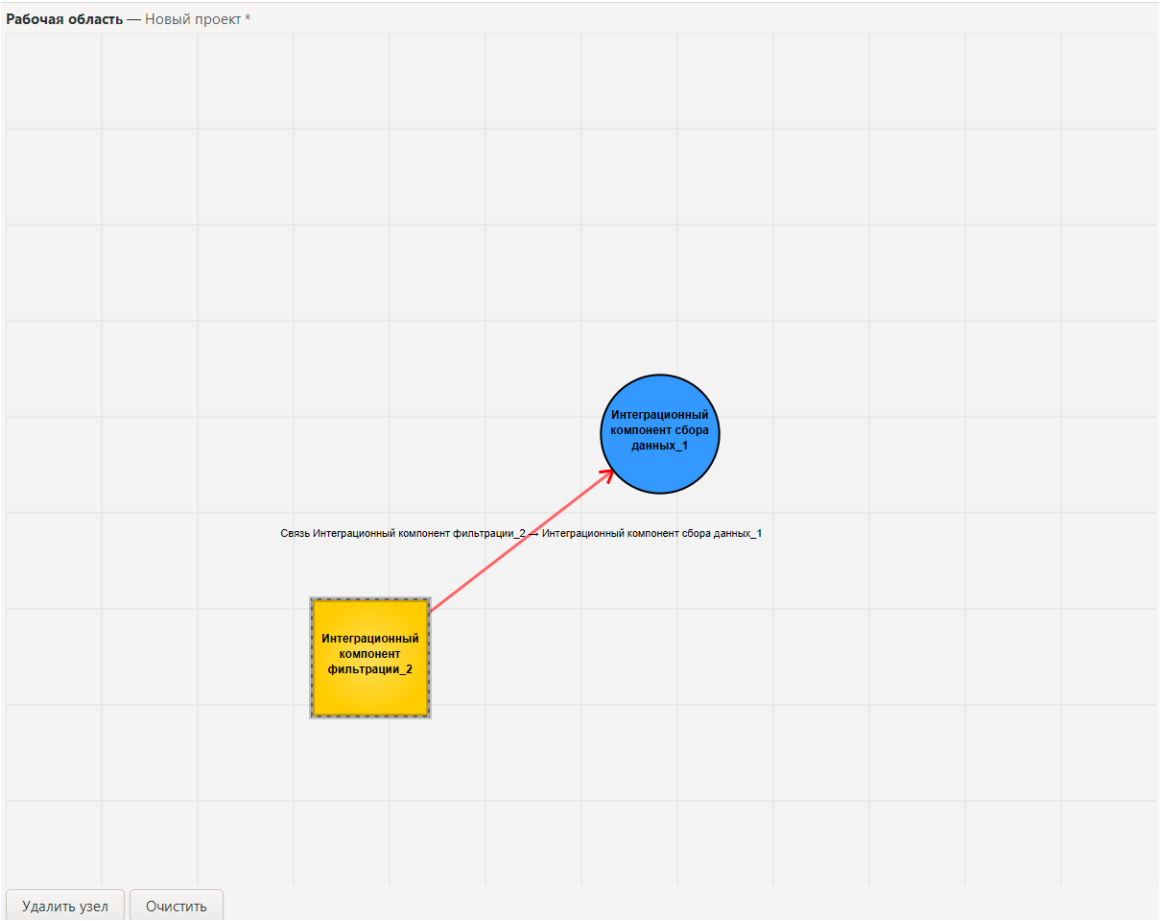


Рис. 3. *Пример связи «Поток данных» между двумя интеграционными компонентами на рабочей области*

Таблица 2

Типы связей и их параметры

Тип связи	Протокол
Поток данных	TCP
Управляющая связь	gRPC
Событийная шина	Kafka

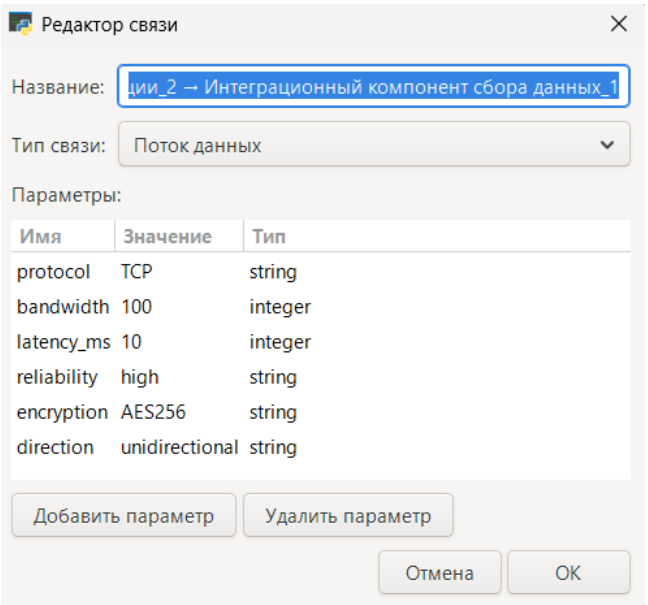


Рис. 4. *Диалог редактирования связи: тип, протокол и параметры взаимодействия*

Каждый проект хранится в виде директории со следующей структурой:

- project.json — метаданные;
- components/ — интеграционные компоненты;
- canvas.json — полная конфигурация каждого интеграционного компонента;
- connections.json — массив объектов со связями.

Применение формата JSON обеспечивает читаемость конфигурации человеком, упрощает отладку и совместимость с системами контроля версий.

Экспорт и импорт

Функция экспорта в UML (рисунок 5) поддерживает три формата:

- PlantUML (.puml) — текстовые описания диаграмм для систем контроля версий и code-review;
- текстовый UML (.txt) — читаемые описания для технической документации и спецификаций;
- XMI (.xmi) — бинарный XML-формат для профессиональных CASE-инструментов.

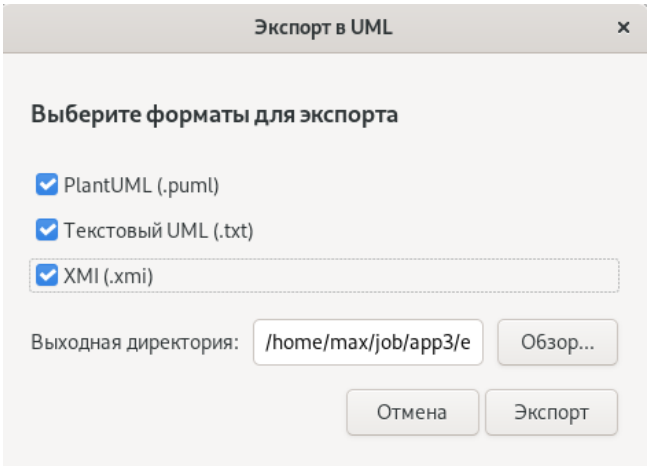


Рис. 5. *Диалог выбора форматов и директории для экспорта в UML*

После завершения экспорта пользователь получает подтверждение с указанием результата по каждому формату (рисунок 6).

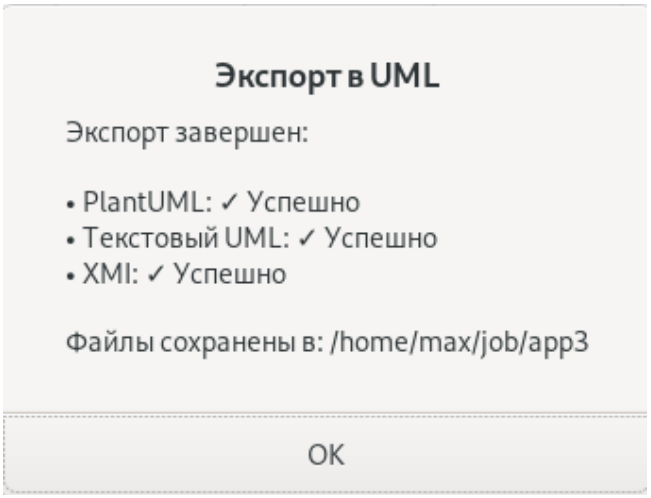


Рис. 6. *Сообщение об успешном экспорте в форматы PlantUML, текстовый UML и XMI*

Экспорт в SVG сохраняет схему в масштабируемом векторном формате с поддержкой прозрачности. Полученный SVG может быть применен в качестве подложки рабочей области.

Функция импорта иерархии интеграционных компонентов позволяет загружать JSON-файлы с иерархической структурой компонентов. Приложение автоматически разбирает структуру.

Интеграция с СУБД PostgreSQL

Для хранения справочных данных используется PostgreSQL [11] — выбор обусловлен надежностью, производительностью и поддержкой формата JSONB для хранения гибких наборов параметров. Физическая модель базы данных включает три таблицы (рисунок 7).

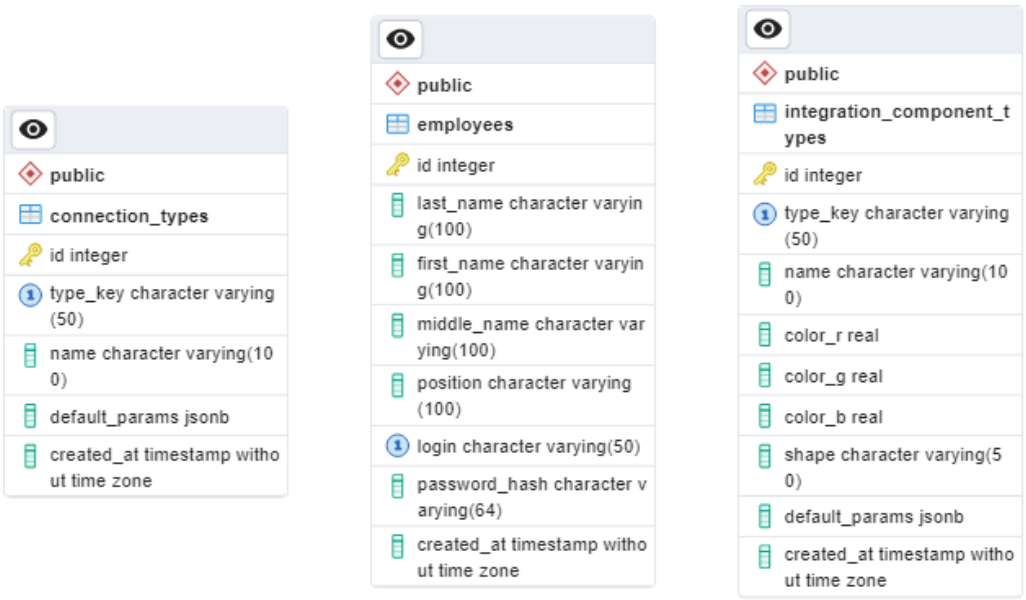


Рис. 7. *Физическая схема базы данных*

Таблица *integration_component_types* содержит тип интеграционного компонента (символьный ключ, имя), RGB-компоненты цвета (тип REAL, диапазон 0–1), форму визуального представления и параметры по умолчанию в JSONB. Использование JSONB позволяет расширять набор параметров без изменения схемы.

Таблица *connection_types* по аналогичной схеме хранит типы связей.

Таблица *employees* обеспечивает аутентификацию: логин, хэш пароля (SHA-256), Ф. И. О. и должность. Все таблицы содержат метку *created_at* для аудита изменений.

API для интеграции подсистем

API для интеграции подсистем обеспечивает взаимодействие конфигуратора с подсистемами обработки онтологий (OWL), бизнес-процессами (BPMN) и диаграммами потоков данных (DFD). Подходы к онтологическому моделированию и использованию языка OWL были определены в первой статье «Концепция платформы генерации прикладных систем управления» [3].

Технологический стек

В качестве основного фреймворка выбран **FastAPI** (лицензия MIT) — современное ASGI-решение со встроенной поддержкой асинхронности, автогенерацией OpenAPI-документации и строгой типизацией через аннотации Python [12]. Для валидации данных применяется **Pydantic v2** [13] (MIT) с Rust-ядром, обеспечивающим значительный прирост скорости валидации по сравнению с первой версией. ASGI-сервер **Uvicorn** (BSD) построен на *uvloop* — быстрой реализации цикла событий asyncio. Весь стек распространяется под разрешающими open-source лицензиями, что соответствует требованиям технологического суверенитета.

Структура и маршруты

API версионируется через префикс */api/v1*. Маршруты сгруппированы по функциональным доменам: *Conversion* (конвертация форматов), *Validation* (валидация моделей), *Analysis* (анализ структуры), *Health* (мониторинг состояния).

Таблица 3

Основные конечные точки API

Метод	Путь	Описание
POST	`/api/v1/convert/visual-to-owl`	Конвертация VISUAL → OWL
POST	`/api/v1/convert/owl-to-bpmn`	Конвертация OWL → BPMN
POST	`/api/v1/convert/bpmn-to-dfd`	Конвертация BPMN → DFD
POST	`/api/v1/convert/chain`	Полная цепочка VISUAL → DFD
POST	`/api/v1/validate/visual`	Валидация визуальной модели
POST	`/api/v1/analyze/visual`	Анализ структуры схемы
GET	`/api/v1/health`	Статус подсистем
GET	`/api/v1/capabilities`	Возможности API

Цепочка конвертации VISUAL→OWL→BPMN→DFD обеспечивает сквозное преобразование. Конечная точка */convert/chain* возвращает все промежуточные представления.

Заключение

В статье описаны архитектурные решения и прототип подсистемы визуализации и настройки интеграционных компонентов для АСУ ТП. По результатам сравнительного анализа четырех архитектурных подходов выбран модульный монолит. Разработанный «Конфигуратор интеграционных компонентов» реализует полный цикл работы с проектом. API на базе FastAPI предоставляет структурированный каркас интеграции с подсистемами OWL, BPMN и DFD с заранее определенными контрактами данных. Полученные результаты создают методологическую и программную основу для последующей реализации процессоров обработки онтологий и бизнес-процессов.

ЛИТЕРАТУРА

1. Гамма Э. и др. *Паттерны объектно-ориентированного проектирования.* Санкт-Петербург: Питер; 2023.
2. Hohpe G., Woolf B. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* Boston: Addison-Wesley; 2003. 659 с.
3. Гимранов Р. Д., Галкин В. А., Гавриленко Т. В., Моргун Д. А., Гавриленко М. Т., Амелин М. Е., Батуро С. А., Грюнталь А. И. Концепция платформы генерации прикладных систем управления. *Успехи кибернетики.* 2026;7(2):41–51.

4. Fowler M. *Patterns of Enterprise Application Architecture*. Boston: Addison-Wesley Professional; 2002. 560 с.
5. Bass L., Clements P., Kazman R. *Software Architecture in Practice*. 3rd ed. Upper Saddle River: Addison-Wesley; 2012. 640 с.
6. Martin R. C. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Upper Saddle River: Prentice Hall; 2017. 432 с.
7. Cockburn A. *Hexagonal Architecture*. 2005. Режим доступа: https://alistair.cockburn.us/hexagonal-architecture/.
8. Newman S. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. Sebastopol: O'Reilly Media; 2019. 270 с.
9. Richards M. *Software Architecture Patterns*. Sebastopol: O'Reilly Media; 2015. 46 с.
10. Burnett M. M. Visual Programming. *Encyclopedia of Electrical and Electronics Engineering*. New York: John Wiley & Sons Inc.; 1999.
11. The PostgreSQL Global Development Group. *PostgreSQL 16 Documentation*. 2023. Режим доступа: https://www.postgresql.org/docs/16/.
12. Ramírez S. *FastAPI Documentation*. 2023. Режим доступа: https://fastapi.tiangolo.com.
13. *Pydantic v2 Documentation*. 2023. Режим доступа: https://docs.pydantic.dev/latest/.