1. Введение

В настоящий момент существуют некоторые проблемы организации структурированного хранения экспериментальных данных в небольших научных группах, ведущих исследования в области естественных науки (физика, химия, биология, астрономия и т. д.). Проблемы можно условно разбить на три категории:

1. **Локальность.** Данные хранятся локально, без возможности какого-либо внешнего доступа к ним. Пример: результаты эксперимента хранятся на носителе одного из членов научной группы, в то время как другие члены группы работают рассредоточено, в результате чего передача данных может быть реализована только через личную встречу или через пересылку данных, непрозрачную для остальных членов группы.
2. **Неструктурированность.** Зачастую даже в пределах одной научной группы не существует жестко заданного формата хранения метаданных об эксперименте (протокола эксперимента), в результате чего возможна неправильная трактовка результатов эксперимента, а также невозможно его воспроизведение. Кроме того, такой подход к хранению метаданных затрудняет подключение новых участников к научной работе.
3. **Потеря акутальности.** В результате первых двух пунктов появляется проблема потери актуальности данных, так как при рассредоточенной работе нет возможности получать последние данные об эксперименте с малой задержкой.

2. Постановка задачи

Требуется разработать систему, предназначенную для хранения и первичной обработки данных экспериментальных исследований в области естественных наук.

Система должна предоставлять следующие функции:

- Возможность накопления и сортировки экспериментальных данных. Под экспериментальным данными подразумеваются данные, полученные с экспериментальной установки (далее сырые данные), а также исчерпывающие метаданные, необходимые для их интерпретации (далее метаданные).
- Возможность управления форматами экспериментальных данных. Сырые данные могут представлять собой как файлы определенного расширения, так и бинарные файлы. Метаданные могут состоять из произвольной информации, то есть необходимо исключить строгие (например, табличное представление) ограничения на формат метаданных.
- Возможность написания и "горячей" установки отдельных модулей, позволяющих проводить парсинг сырых данных, расчет необходимых метрик и их графическое представление (далее парсеров). Модули должны быть полностью обособлены от самой системы.
- Возможность генерации простейших отчетов.
- Возможность экспорта данных для их автономного перемещения и хранения.

Требования к системе:

- Должна быть реализована клиент-серверная архитектура, причем клиент должен предоставлять GUI.
- Должна быть обеспечена неизменяемость сохраненных данных.
- Должна быть реализована система пользователей и прав доступа.
- Механизм разработки парсеров может быть реализован как с использованием разработанного DSL, так и с использованием уже существующего языка программирования.
- Должен быть реализован API, позволяющий производить накопление сырых данных непосредственно в хранилище в процессе эксперимента (путем вызова методов API из некоторого связующего ПО для экспериментальной установки).

3. UML модель системы

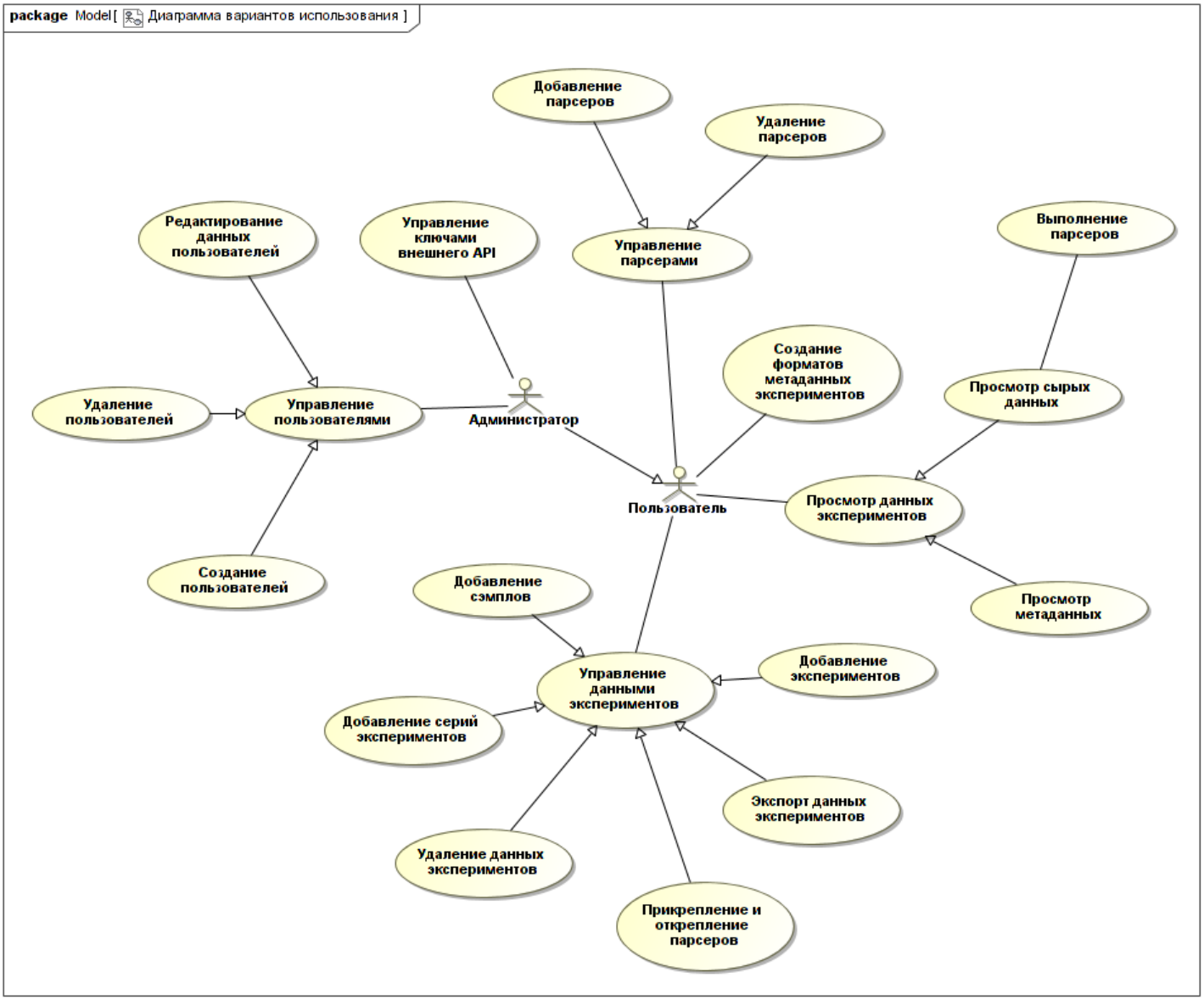

Рис. 1 Диаграмма вариантов использования

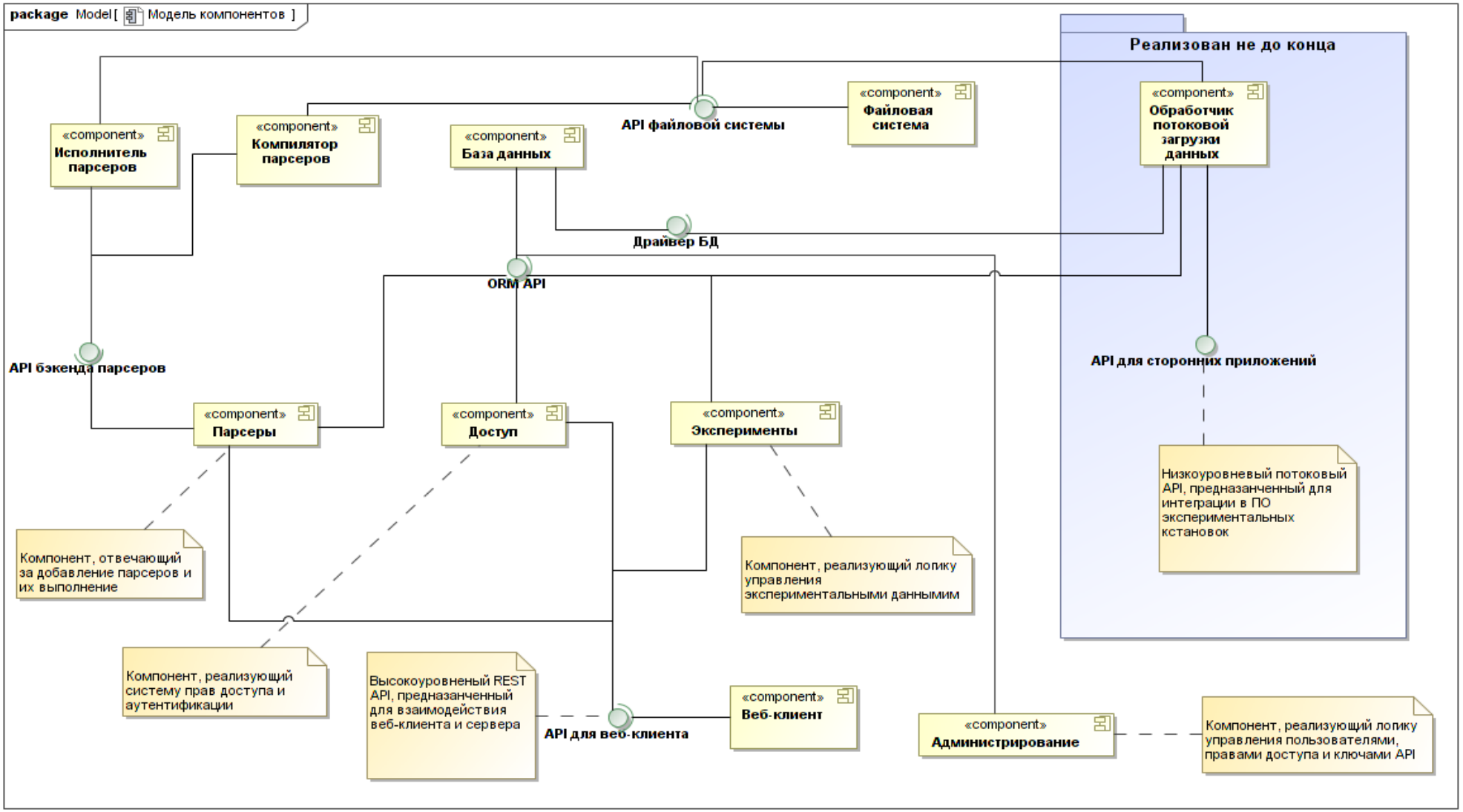

Рис. 2 Диаграмма компонентов

Пояснение к рис.2 (диаграмме компонентов):

Описание компонентов:

- **Эксперименты.** Отвечает за управление форматами метаданных; записями экспериментов и серий в БД; отображение и управление сериями, экспериментами и семплами; экспорт данных.
- **Доступ.** Отвечает за аутентификацию пользователей, проверку прав доступа пользователей.
- **Парсеры.** Отвечает за создание и отображение парсеров.
- **Исполнитель парсеров.** Отвечает за валидацию парсеров и их выполнение.
- **Компилятор парсеров.** Отвечает за валидацию парсеров и их компиляцию.
- **База данных.** Отвечает за хранение данных, предоставляет обертку в виде ORM.
- **Файловая система.** Отвечает за хранение «сырых» данных, выходных данных парсеров и их выдачу.
- **Администрирование.** Отвечает за создание и редактирование пользователей; назначение прав доступа; управление ключами потокового API.
- **Обработчик потоковой загрузки данных.** Отвечает за прием потоковых данных от экспериментальных установок через TCP сокеты, обеспечение их целостности и валидности.

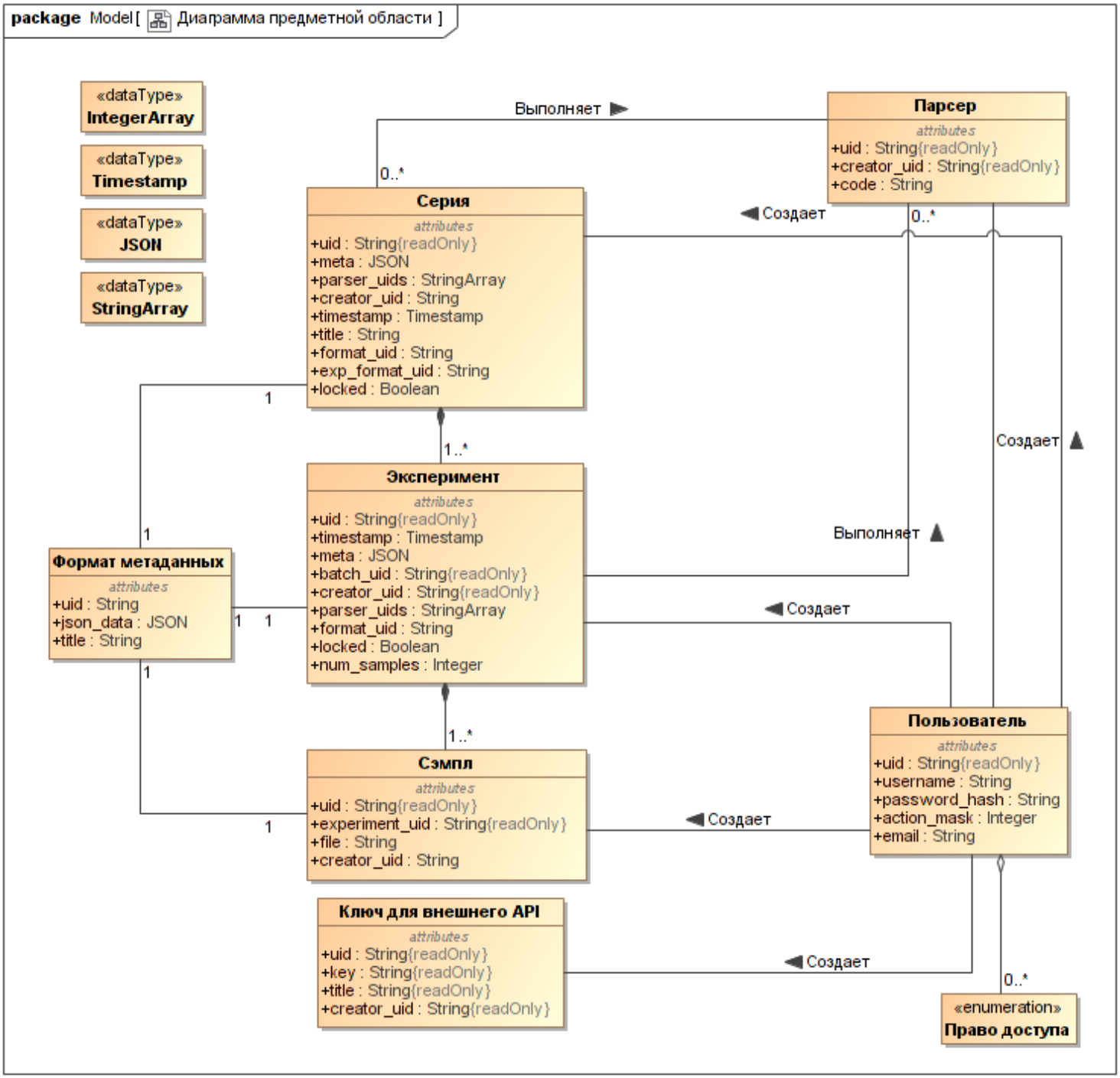

Рис. 3 Модель предметной области

Пояснение к рис. 3 (модели предметной области):

Сущности:

- **Формат метаданных.** Формат протокола эксперимента в виде JSON.
- **Серия.** Набор экспериментов, объединенных общими свойствами. Имеет свои метаданные, а также определяет формат метадднных экспериментов, в нее входящих.

- **Семпл.** Единица «сырых» данных эксперимента. В системе представляется прозвольным файлом. В отрыве от эксперимента и его метаданных смысла не имеет.
- **Ключ для внешнего API.** Ключ-токен для авторизации клиентских приложений во вненем потоковом API.
- **Парсер.** Программа на языке программирования Python, реализующая опредленный интерфейс.
- **Право доступа.** Определнное действие, к которому пользователь может иметь или не иметь доступ. В системе представлено значением сдвига битовой маски всех прав доступа пользователя.
- **Пользователь.** Профиль пользователя. Используется для аутентификации и идентификации в системе.

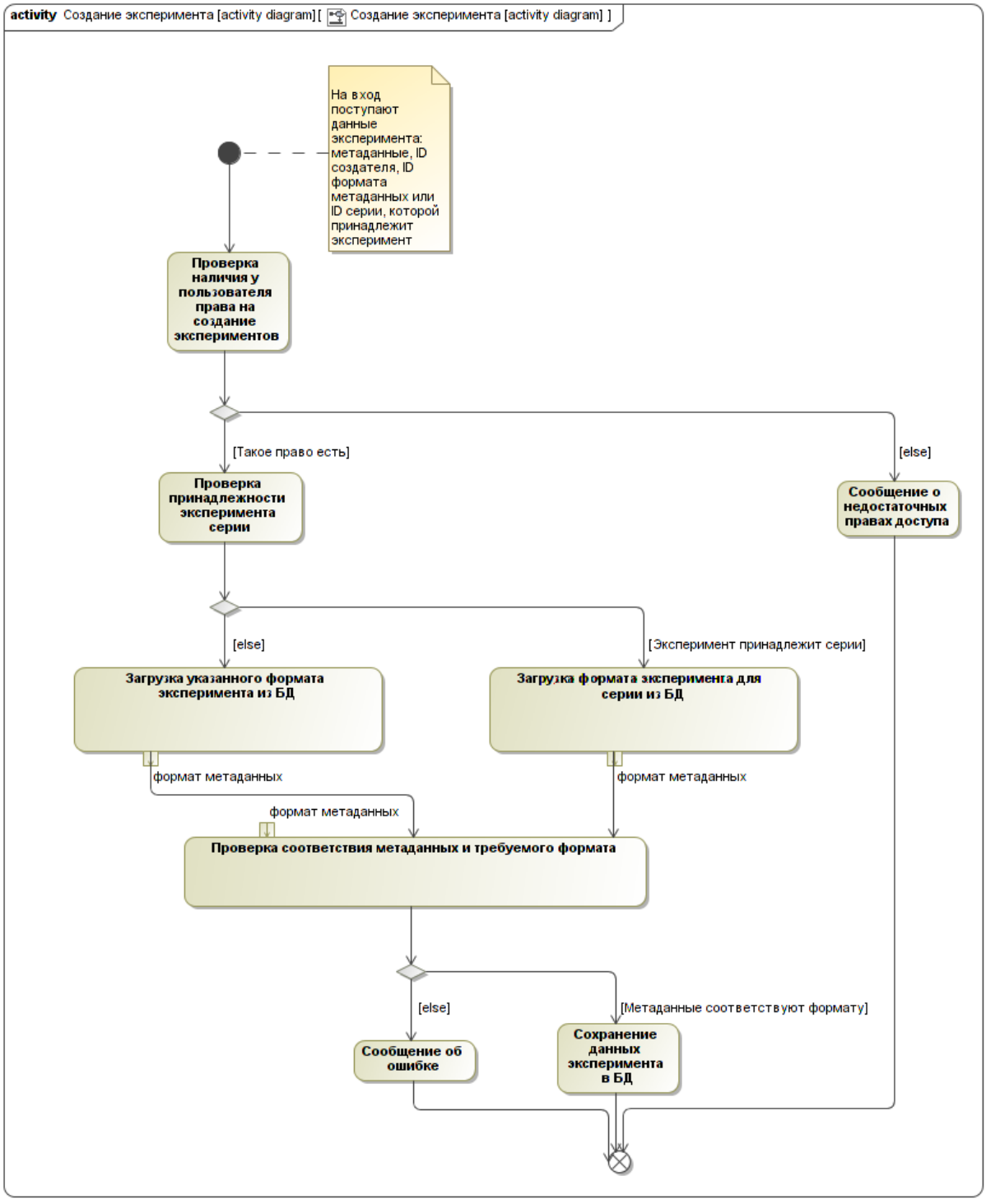

Рис. 4 Диаграмма деятельности процесса создания эксперимента

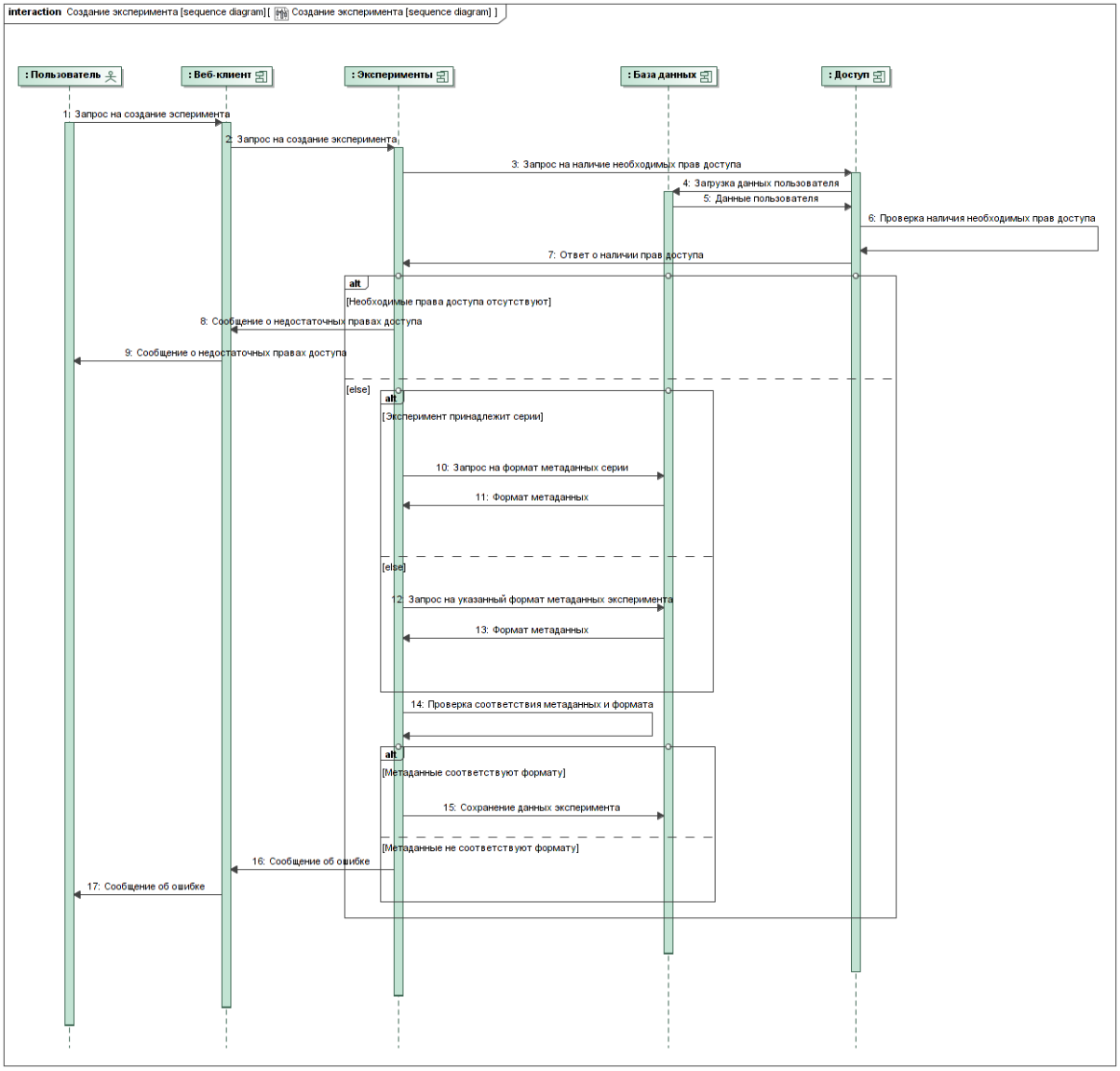

Рис. 5 Диаграмма последовательности процесса создания эксперимента

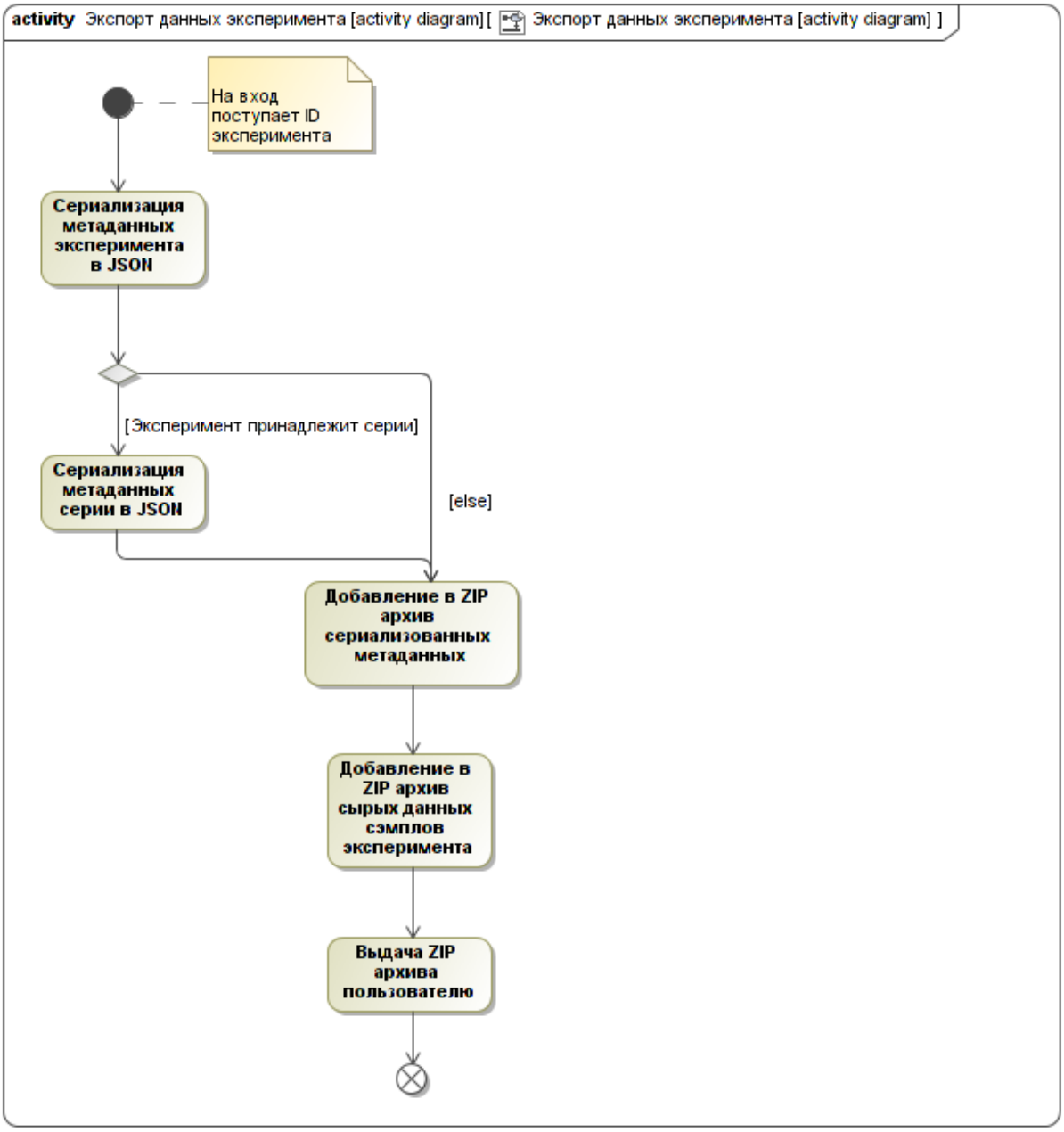

Рис. 6 Диаграмма деятельности процесса экспорта данных эксперимента

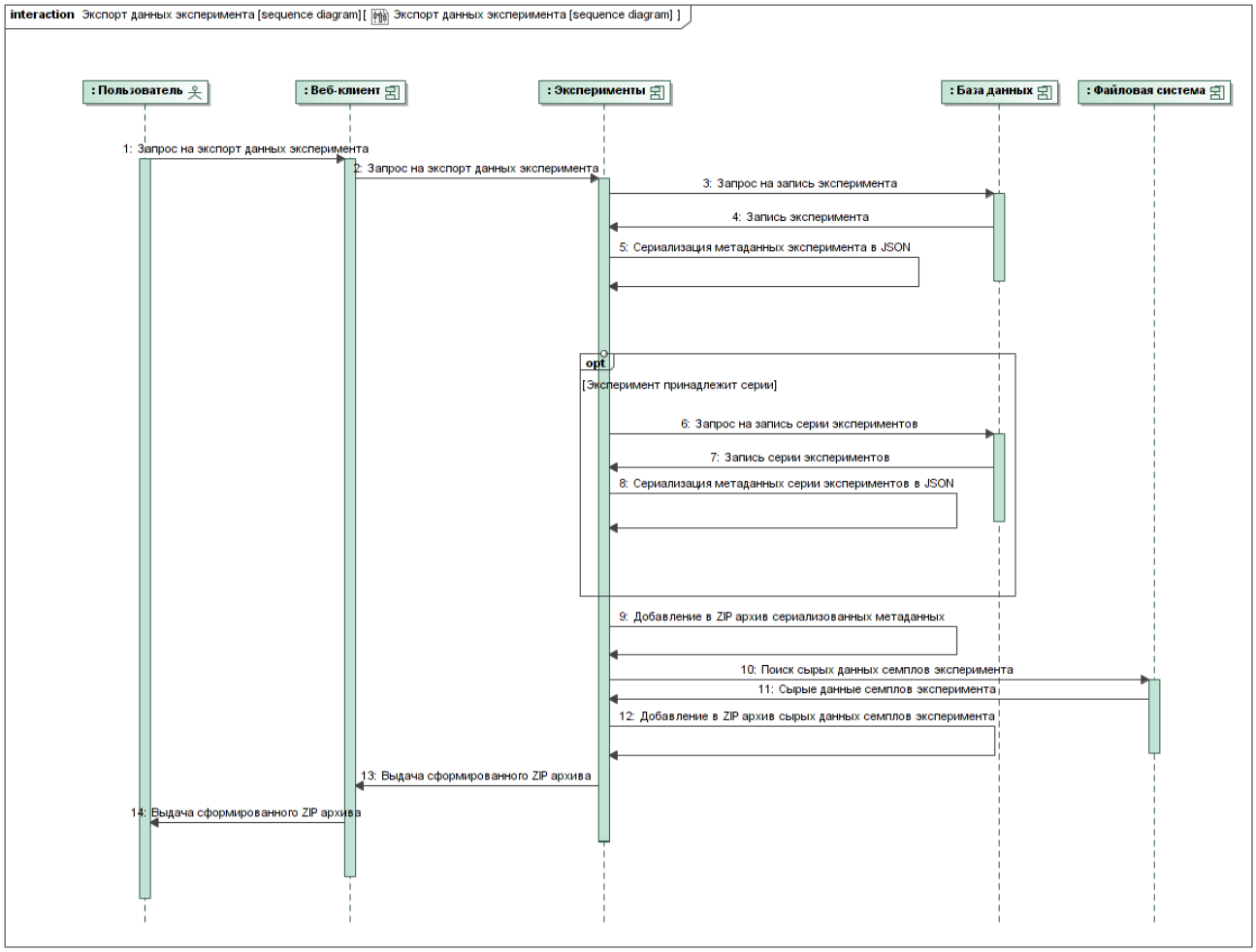

Рис. 7 Диаграмма последовательности процесса экспорта данных эксперимента

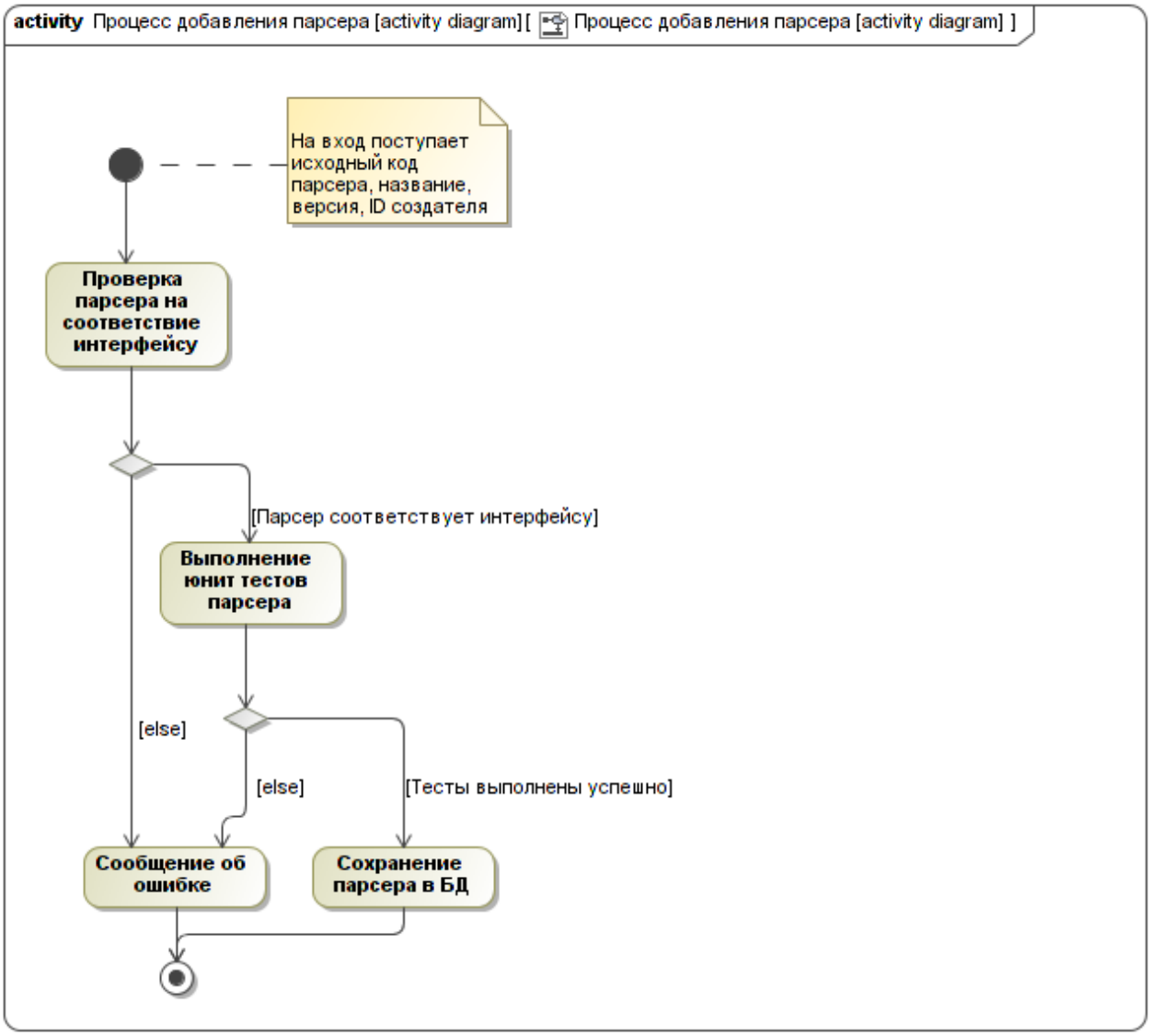

Рис. 8 Диаграмма деятельности процесса создания парсера

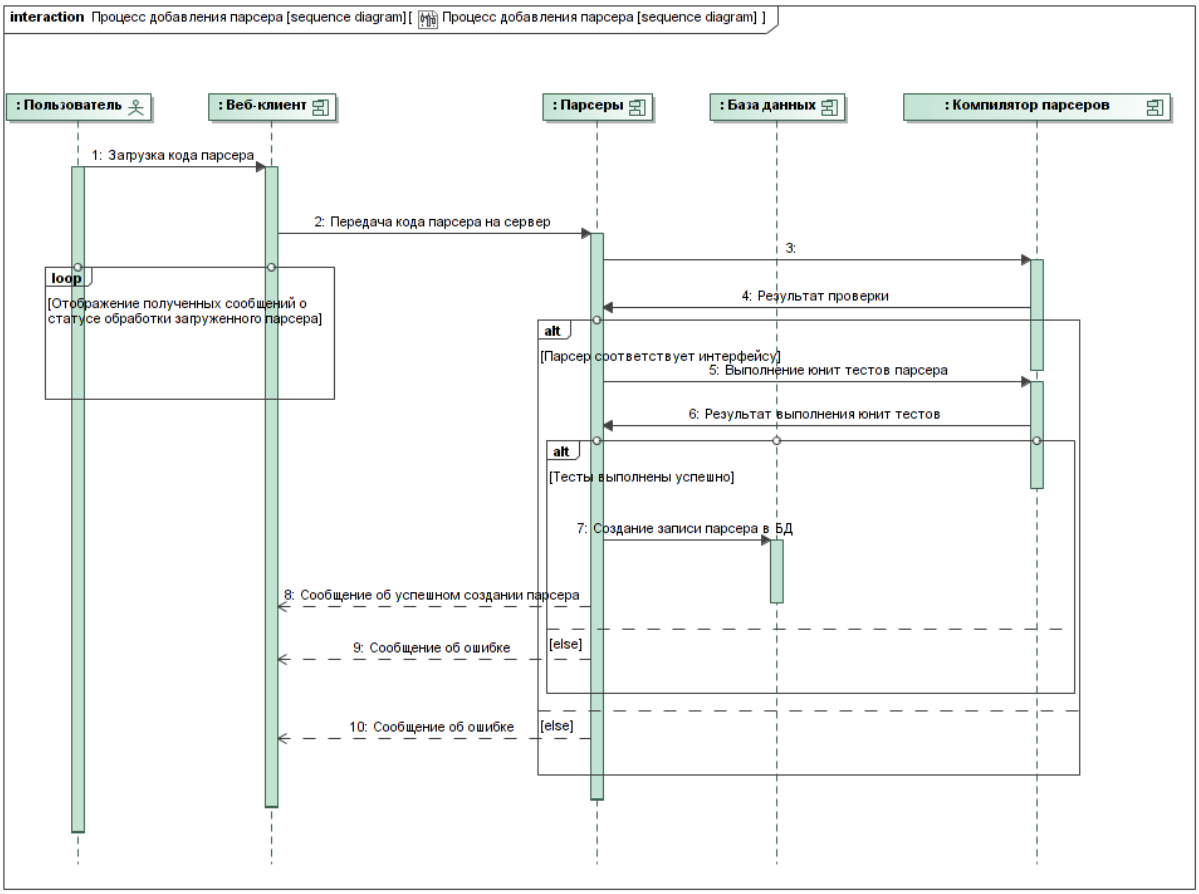

Рис. 9 Диаграмма последовательности процесса создания парсера

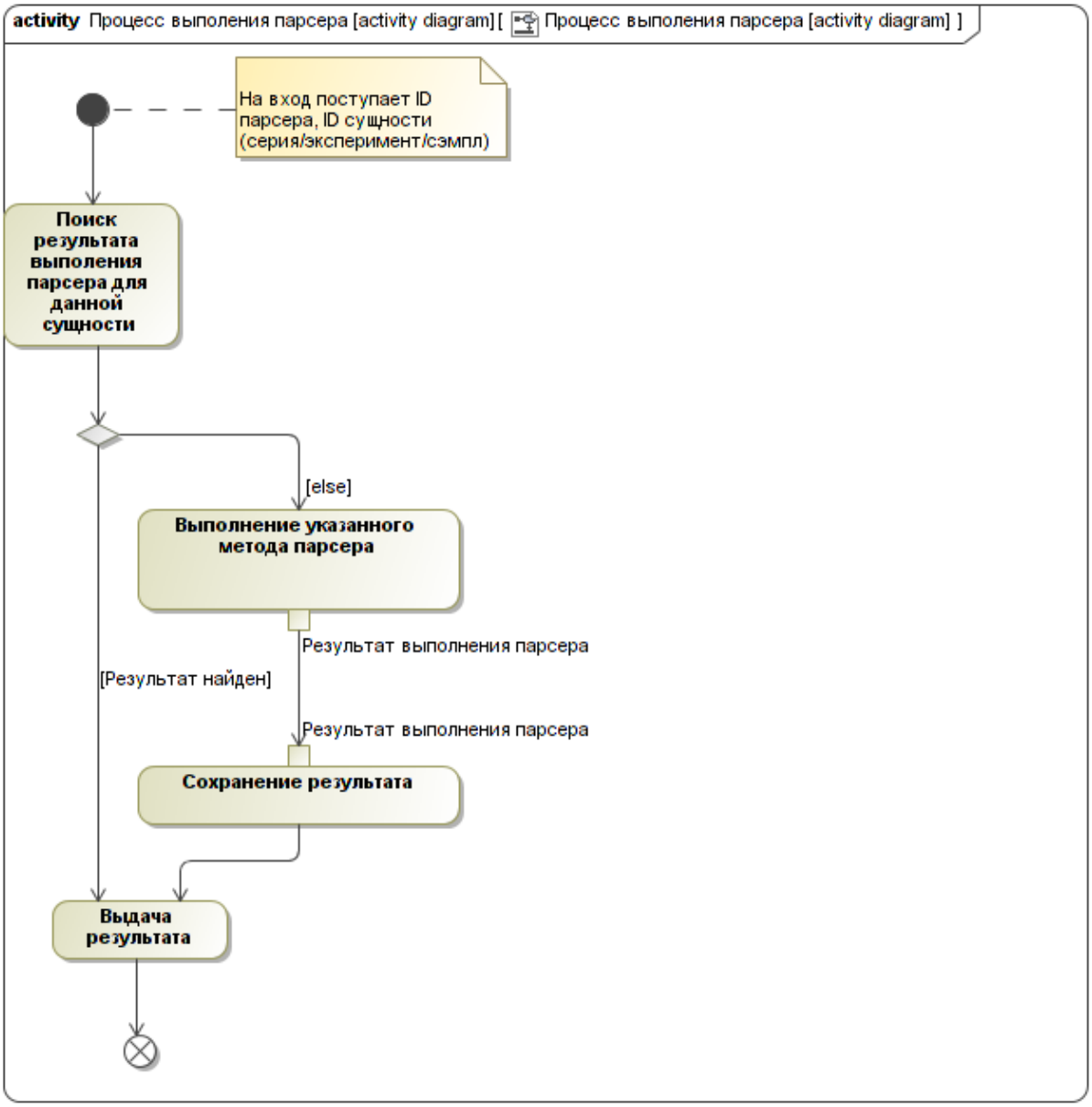

Рис. 10 Диаграмма деятельности процесса выполнения парсера

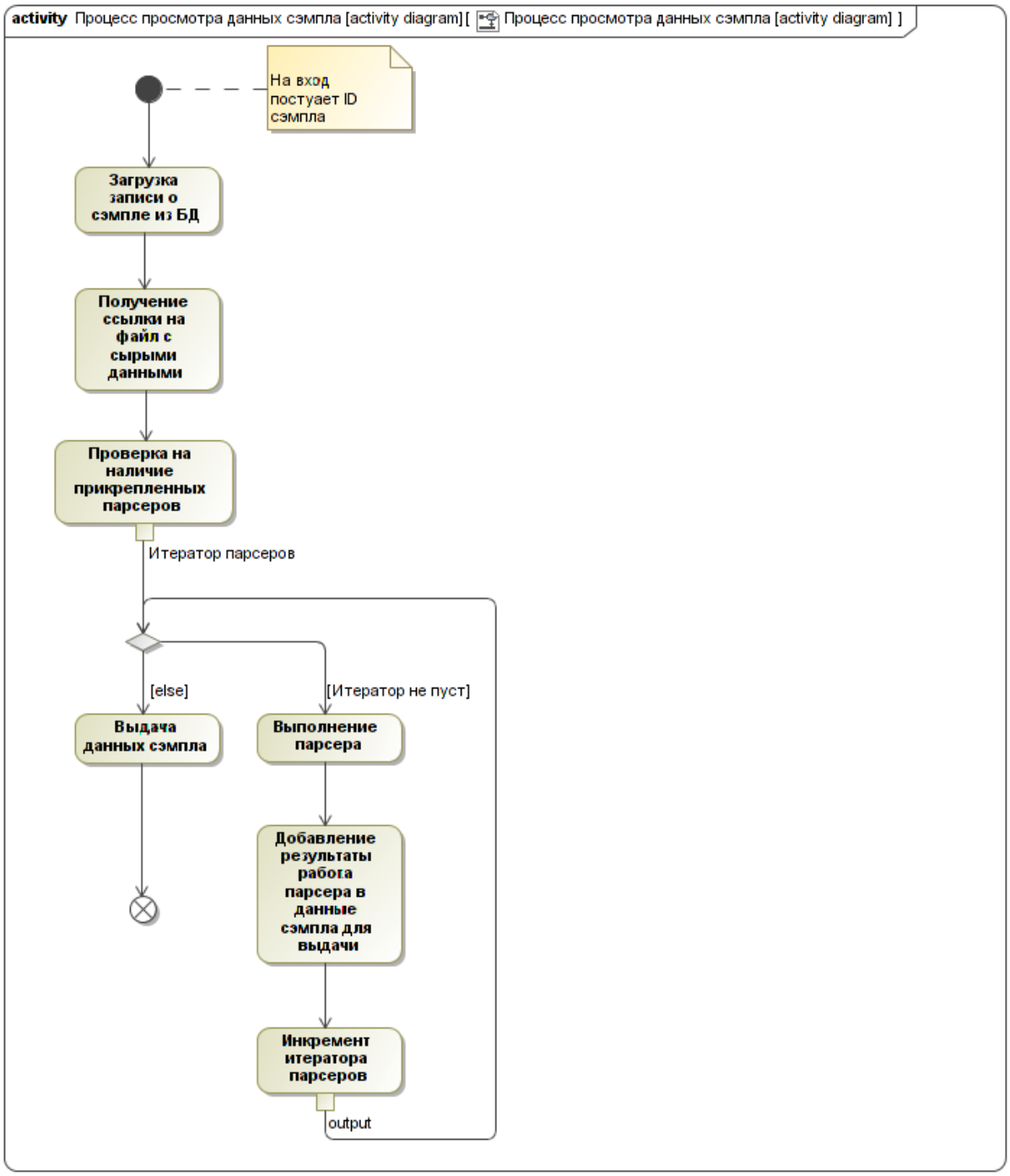

Рис. 11 Диаграмма деятельности процесса просмотра данных семпла

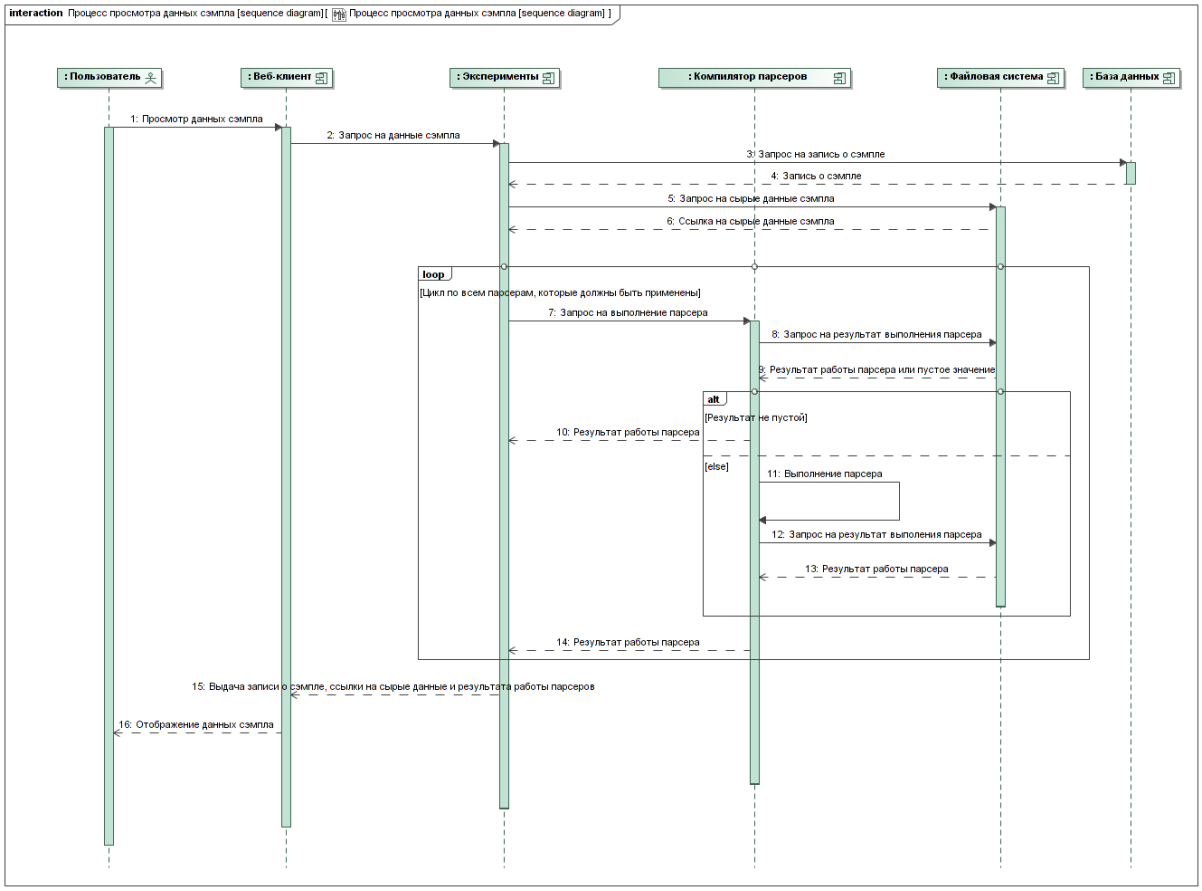

Рис. 12 Диаграмма последовательности процесса просмотра данных семпла

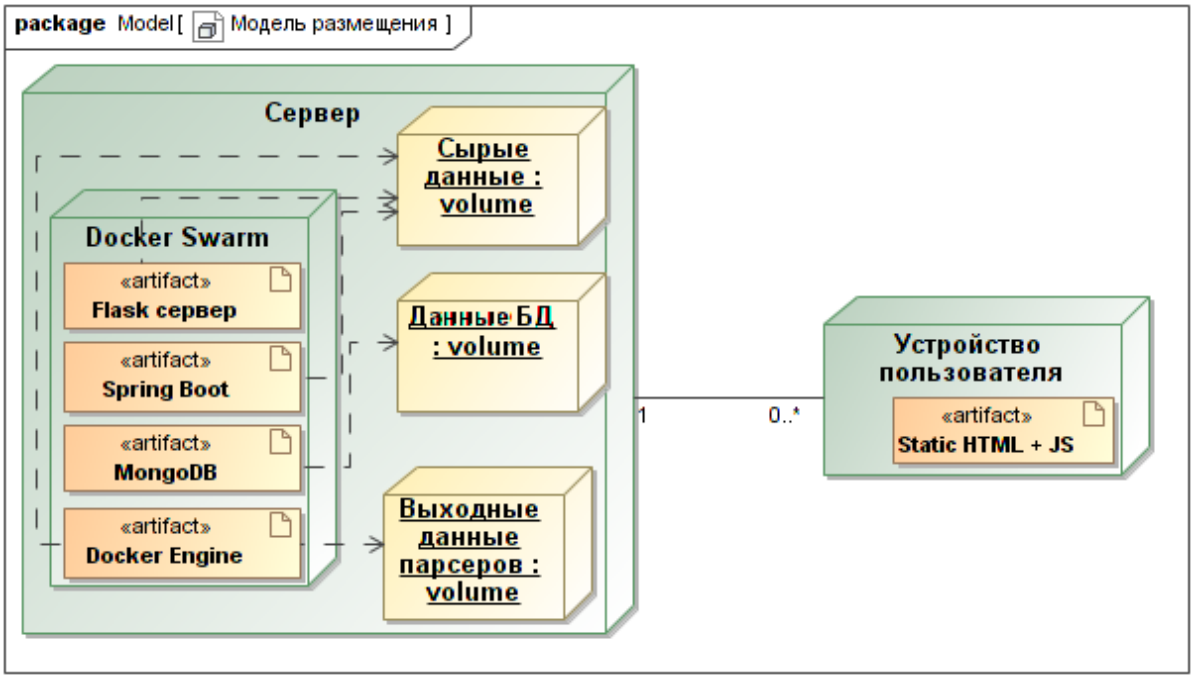

Рис. 13 Модель размещения

4. Использованный стек технологий

Для реализации системы, удовлетворяющей представленным требованиям, был использован следующий стек технологий:

- WEB интерфейс:
  - Фронтенд: Jinja2 + Twitter Bootstrap + JavaScript
  - Бэкенд: Python + Flask
- База данных: MongoDB
- Конвейер выполнения парсеров: Docker Engine + Python
- Компонент потоковой обработки “сырых” данных: Java
- Инфраструктура для развертывания приложения: Docker Engine, Docker Swarm, Bash скрипты

Исходный код системы доступен в Git репозитории:
https://github.com/dev0x13/euclid

5. Результаты

Разработанная система в целом удовлетворяет поставленной задаче: не был реализован только компонент генерации отчетов; компонент потокового API был реализован на уровне работающего прототипа.

Вся разработка выполнялась в рамках модели «Инфраструктура как код», то есть автоматизировано и отражено в коде скриптов все процессы, которые автоматизировать. Кроме того, средствами Docker Swarm все зависимости системы сведены к Docker Engine: достаточно поставить его на чистую операционную систему (Linux) и запустить скрипт, который развернет систему. На момент написания система развернута по адресу: http://3.120.98.247:5000

Реализована система аутентификации (средствами Flask-Login), а также система прав доступа и механизм управления ими. Пример конфигурации профиля пользователя при его создании показан на рис. 14.

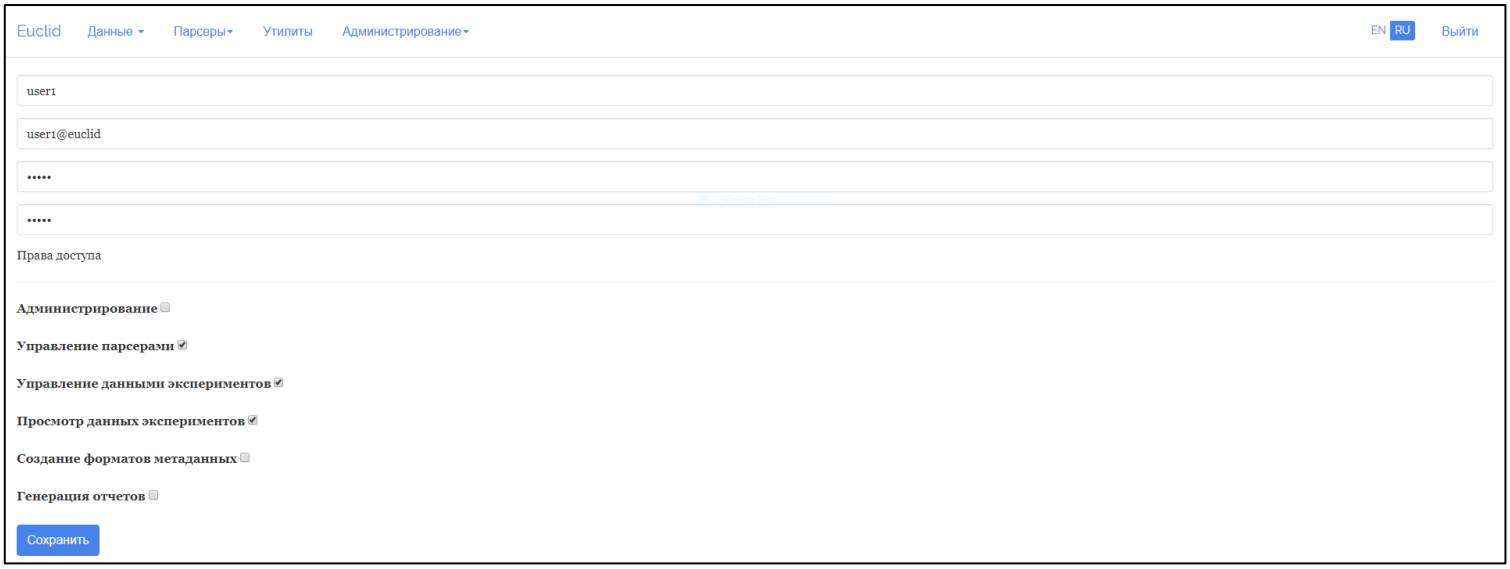

Рис. 14 Окно создания и редактирования профиля пользователя

Реализован механизм управления доступом внешних приложений к потоковому API (рис. 15.). Сам потоковый API находится в стадии работающего прототипа (разработан на языке программирования Java).

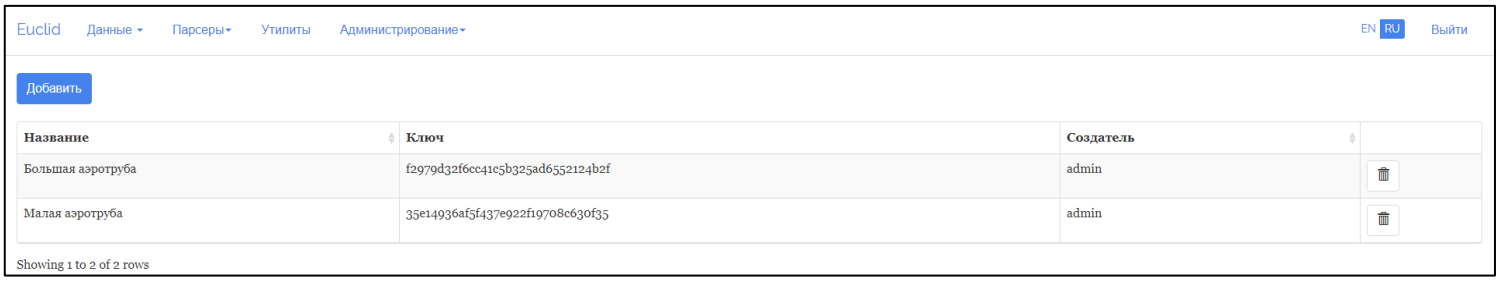

Рис. 15 Окно управления ключами потокового API

Реализована система управления форматами метаданных экспериментов и серий экспериментов, и их валидации. В качестве метаформата был выбран JSON. Валидация производится по структуре метаданных (рекурсивно), именам ключей, а также по типу значений. Таким образом, гарантируется наличие корректных метаданных для каждого набора данных. Пример созданных форматов метаданных показан на рис. 16. Пример создания формата в интерактивном редакторе и проверкой грамматики показан на рис. 17.

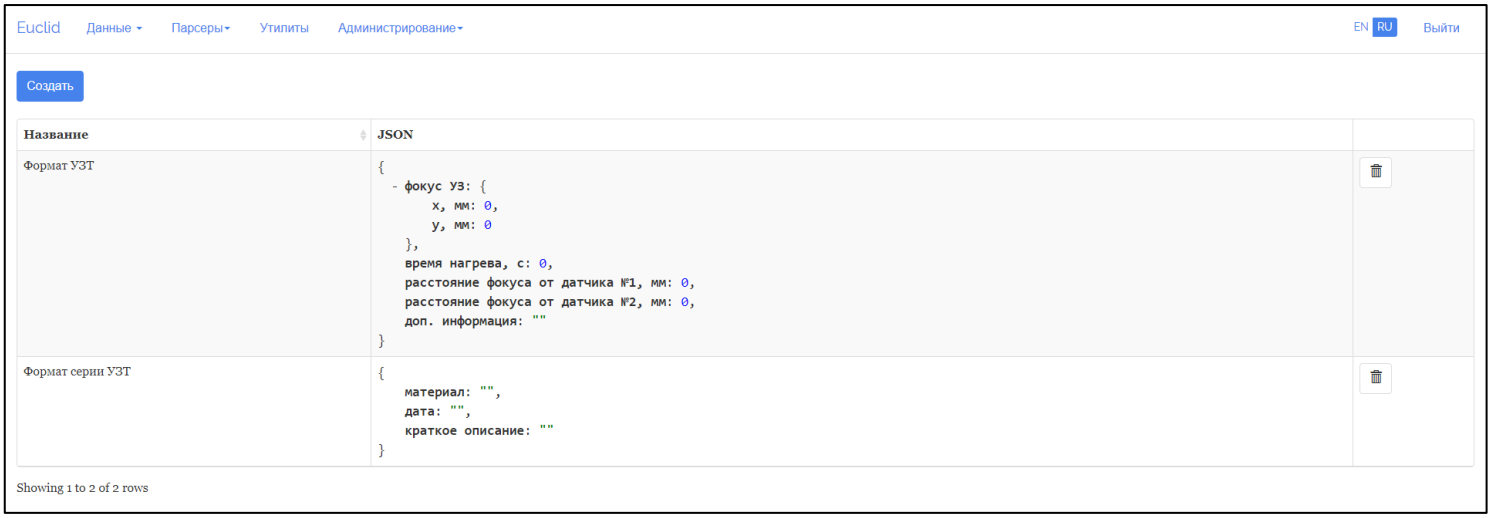

Рис. 16 Окно управления форматами метаданных

Рис. 17 Окно создания формата метаданных

Реализована система управления данными экспериментов. Для серий и экспериментов реализован механизм блокировки, который обеспечивает неизменяемость данных после их добавления. На рис. 18 показан внешний вид серии экспериментов с заполненными метаданными и добавленным экспериментом, а на рис. 19 – внешний вид эксперимента с добавленными семплами. Кроме того, реализована возможность экспорта данных на любом из уровней (серия, эксперимент, семпл). Запрошенные на экспорт данные упаковываются в ZIP архив совместно с метаданными и выдаются запросившему пользователю.

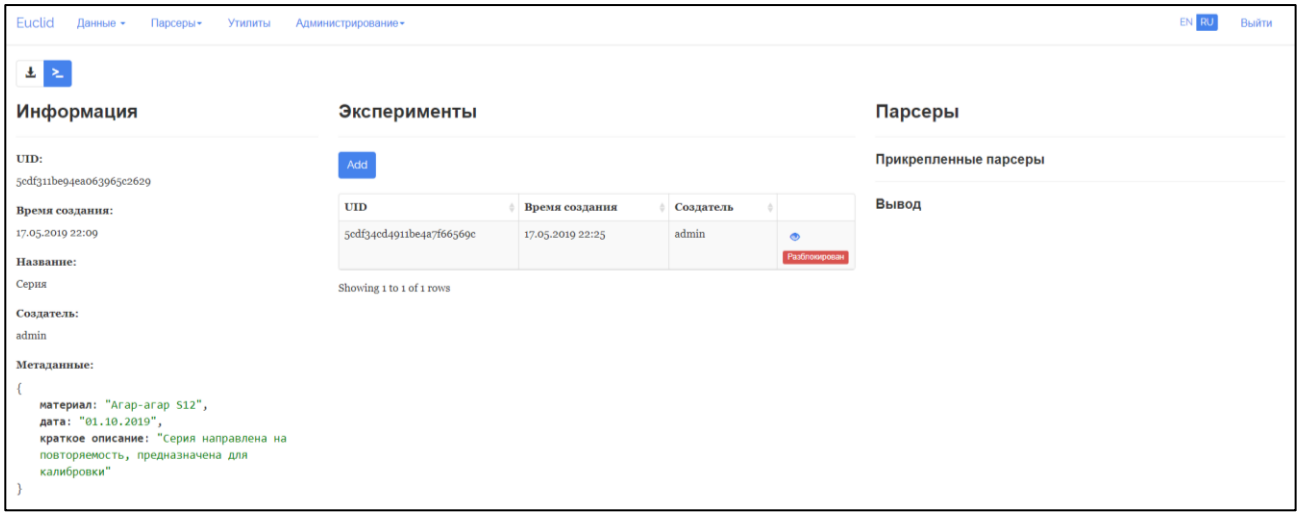

Рис. 18 Окно просмотра данных серии экспериментов

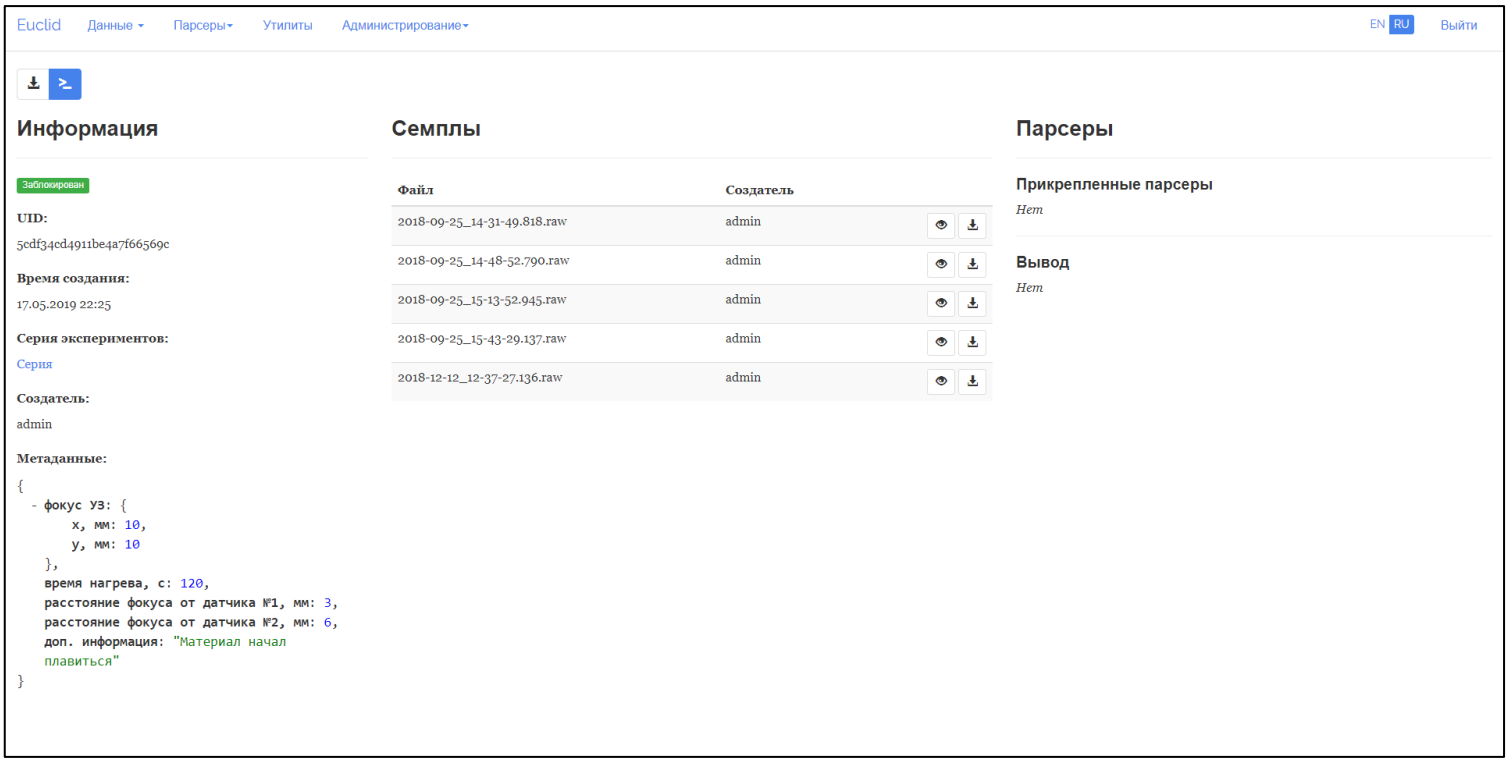

Рис. 19 Окно просмотра данных эксперимента

В соответствии с техническим заданием в качестве возможности первичного анализа данных экспериментов была разработана подсистема парсеров. Разработанная подсистема позволяет отлаживать и выполнять пользовательский код специального вида на языке программирования Python изолированно (и безопасно) от серверного окружения. Для разработки была создана интегрированная в браузер среда (рис. 20). Разработчику парсера предоставлены функции для вывода изображений и текста в форму веб-интерфейса сущности, к которой прикреплен парсер. Кроме того, для облегчения обработки и анализа данных в подсистеме установлены следующие пакеты Python: NumPy, SciPy, Pillow, Matplotlib.

Пример работы парсера «Спектрограмма WAV файла» на тестовых экспериментальных данных показана на рис. 21. Неизменяемость данных как главный принцип системы позволяет выполнять парсеры всего один раз для каждой единицы данных: результат кэшируется и его актуальность в любой момент времени гарантирована.

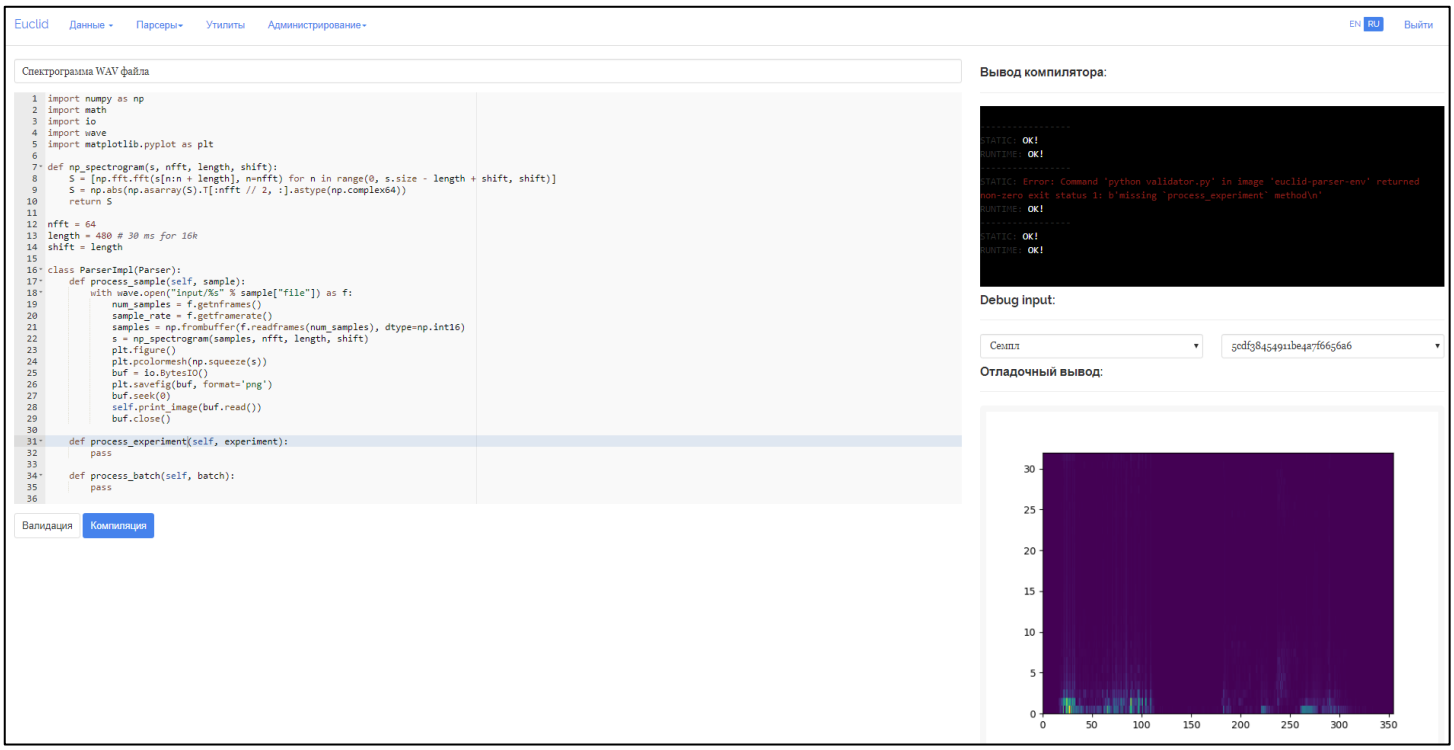

Рис. 2019 Интегрированная среда разработки парсеров

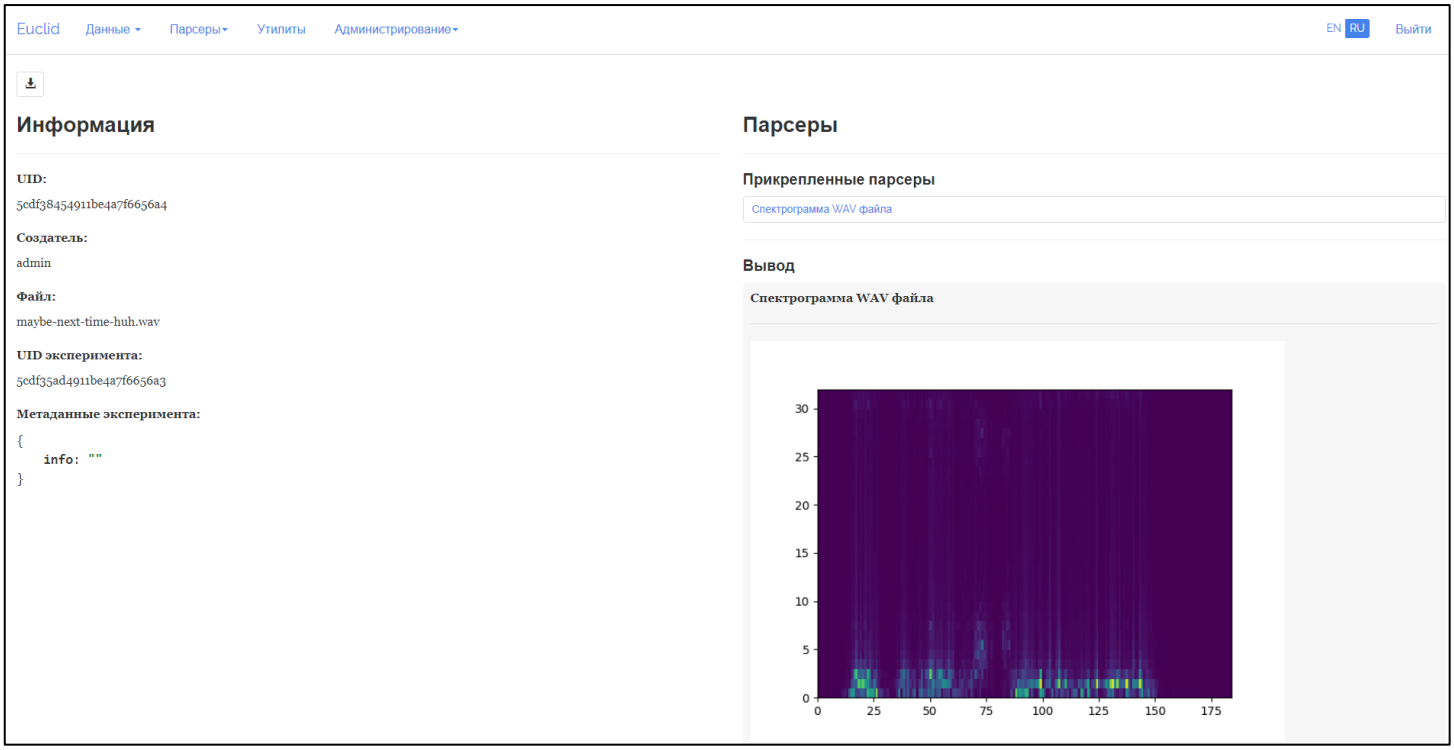

Рис. 201 Окно просмотра данных семпла с прикрепленным парсером

Кроме того, реализована поддержка английского и русского языков интерфейса с возможностью переключения пользователем. Языковая локализация реализована средствами Babel и имеет возможность легкого расширения для любого набора языков.

6. Дальнейшее развитие проекта

Дальнейшие шаги по развитию проекта:

- Доработка потокового API
- Модернизация подсистемы парсеров: добавление возможности установки целых пользовательских модулей (по аналогии с AWS Lambda)
- Написание подробной документации
- Разработка CLI (Command Line Interface) для возможности автоматизации работы с данными экспериментов