2. Диаграммы

2.1. Диаграммы прецедентов

Комплекс при его работе предоставляет пользователю следующие варианты использования:

- регистрация пользователя;
- авторизация пользователя;
- постановка задачи на исполнение;
- просмотр статуса задачи;
- отмена задачи.

Диаграмма этих и дополнительных служебных прецедентов приведена на рис. 1.

С учётом требований к разделению внутреннего функционала комплекса, диаграмма прецедентов на рис. 1 расщепляется на набор диаграмм, соответствующих каждой из выделенных подсистем. Соответствующие диаграммы приведены на рисунках 2,3,4.

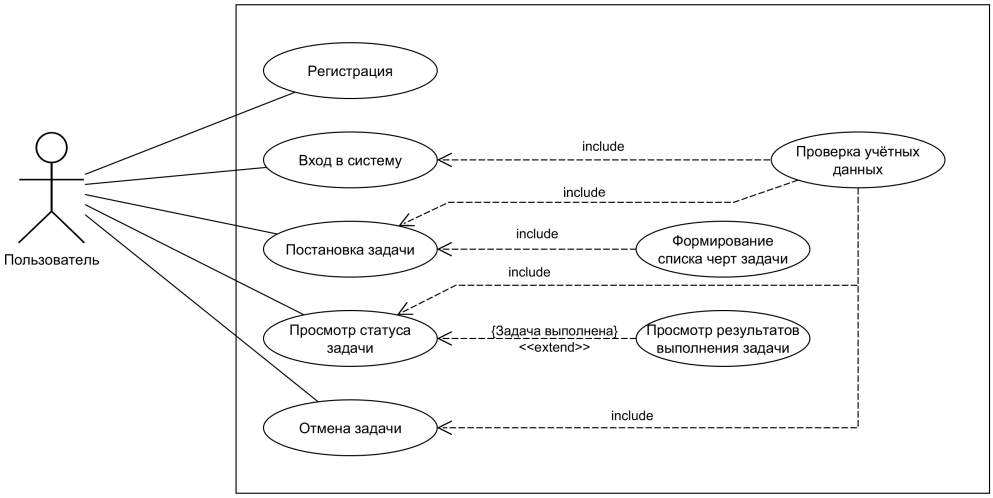

Рис. 1: Диаграмма прецедентов всего комплекса в целом

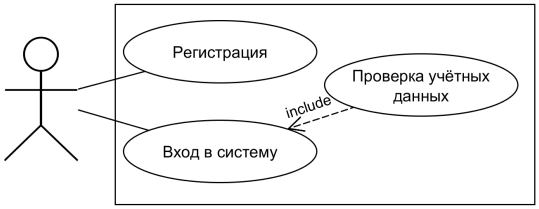

Рис. 2: Диаграмма прецедентов СУС

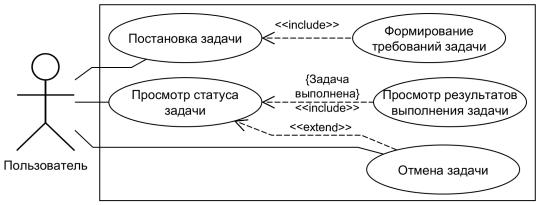

Рис. 3: Диаграмма прецедентов СУ

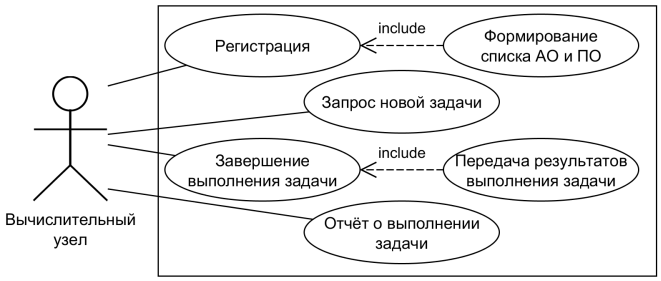

Рис. 4: Диаграмма прецедентов СБН

2.2. Диаграммы действий

Прецеденты, описанные в предыдущем пункте, отвечают определённой деятельности. Диаграмма деятельности на рис. 5 описывает полный процесс взаимодействия пользователя с комплексом.

С учётом требований к разделению внутреннего функционала комплекса, диаграмма деятельности на рис. 5 расщепляется на набор диаграмм, соответствующих определённым подсистемам из выделенных.

Диаграммы действий прецедентов подсистемы управления сессией “регистрация” и “вход в систему” приведены на рисунках 6 и 7 соответственно.

Диаграммы действий прецедентов системы балансировки нагрузки “регистрация”, “запрос новой задачи” и “завершение выполнения задачи” приведены на рисунках 8, 9 и 10 соответственно.

Диаграммы действий прецедентов системы управления “постановка задачи” и “просмотр статсуа задачи” приведены на рисунках 11 и 12 соответственно.

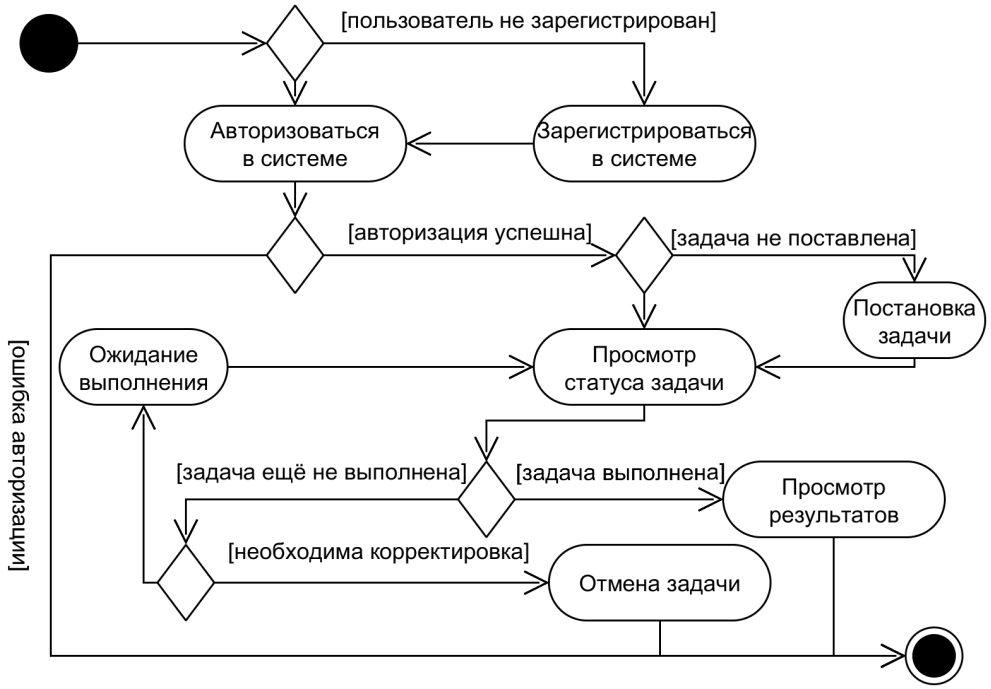

Рис. 5: Диаграмма действий прецедента “общая деятельность” для системы в целом

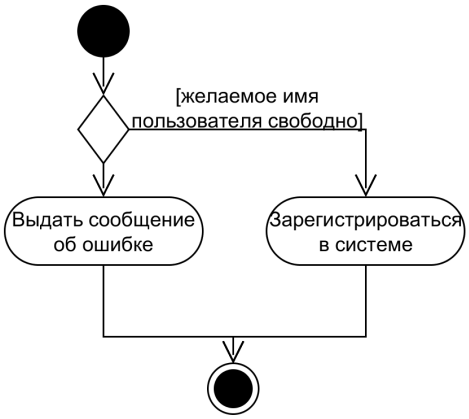

Рис. 6: Диаграмма действий прецедента “регистрация” СУС

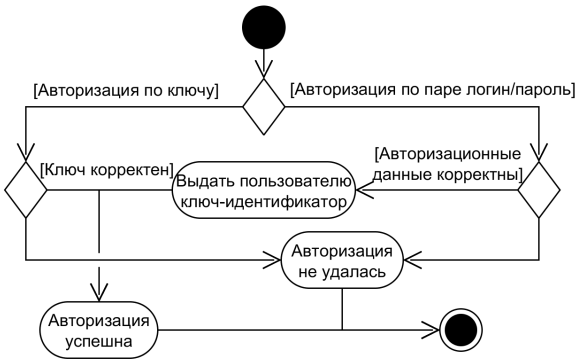

Рис. 7: Диаграмма действий прецедента “вход в систему” СУС

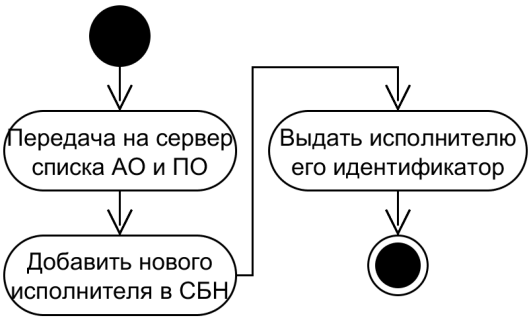

Рис. 8: Диаграмма действий прецедента “регистрация” СБН

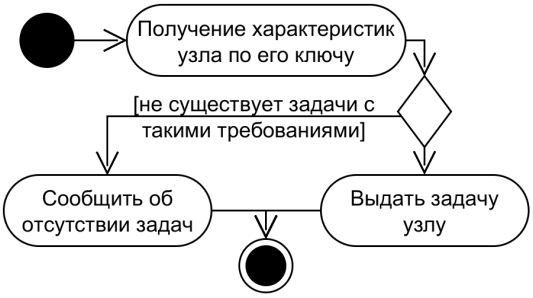

Рис. 9: Диаграмма действий прецедента “запрос новой задачи” СБН

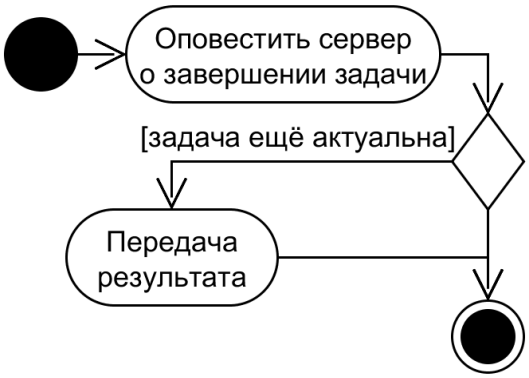

Рис. 10: Диаграмма действий прецедента “завершение выполнения задачи” СБН

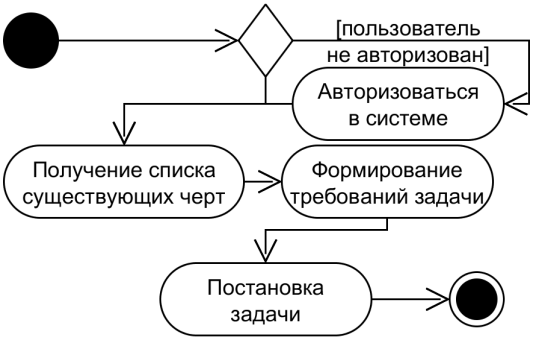

Рис. 11: Диаграмма действий прецедента “постановка задачи” СУ

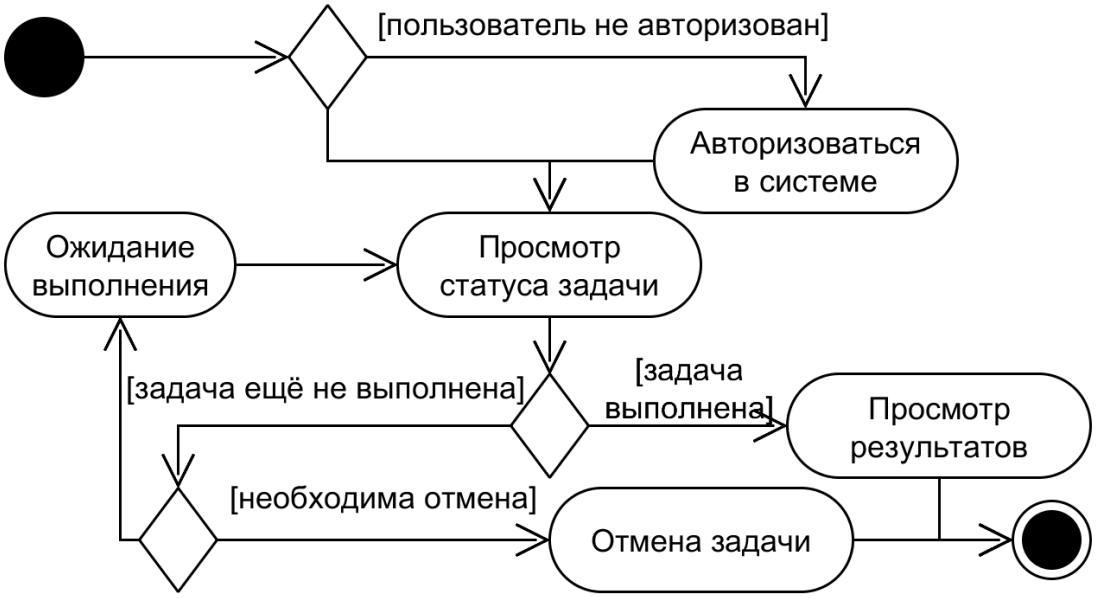

Рис. 12: Диаграмма действий прецедента “просмотр статсуа задачи” СУ

2.3. Диаграмма компонент и развёртывания

Для того, чтобы удовлетворить требованиям по предоставлению механизма деградации функциональности, а также для упрощения процесса разработки, комплекс должен быть разделен на отдельные слабосвязанные элементы.

Различные подсистемы комплекса имеют некую модель поведения. Поведение подсистемы описывается её активной и пассивной частью. Активная часть соответствует действиям, которые подсистема выполняет разово либо с некоторой периодичностью, в автоматическом режиме. Пассивная часть соответствует API подсистемы. Взаимосвязи между различными компонентами системы приведены на диаграмме компонентов на рис. 13. Физическое размещение компонент по отдельным узлам проиллюстрировано на диаграмме развёртывания на рис. 14.

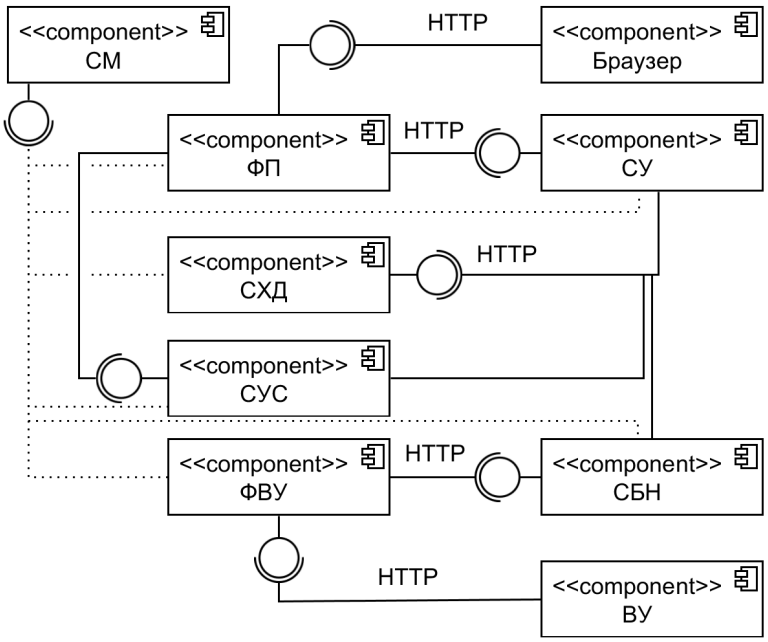

Рис. 13: Диаграмма компонент комплекса

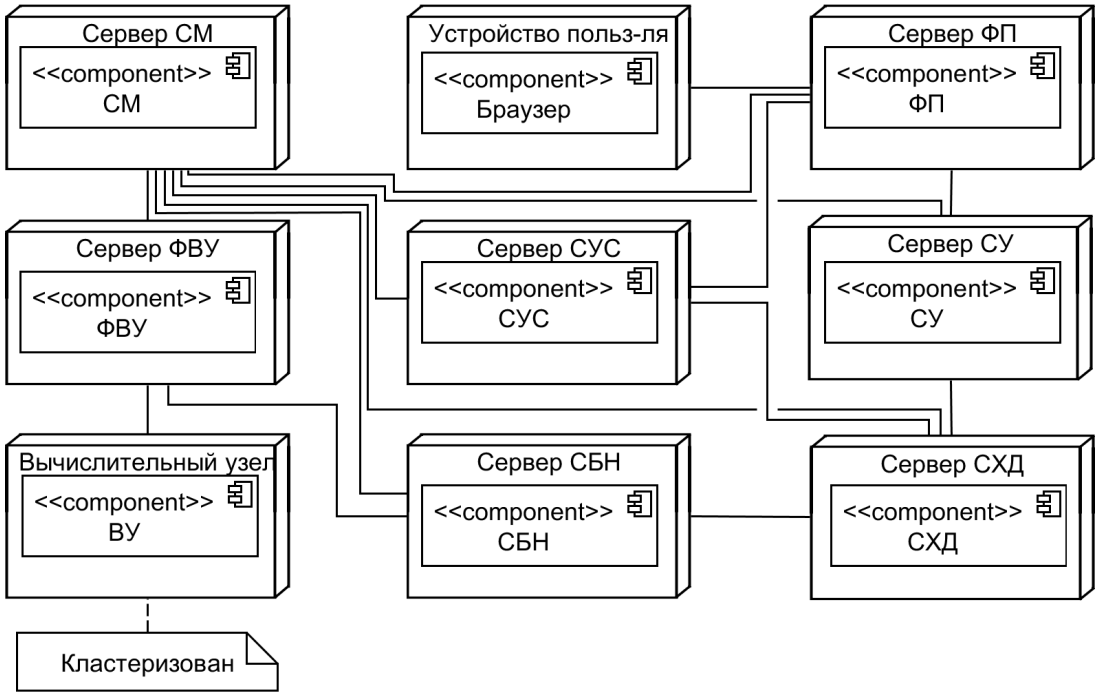

Рис. 14: Диаграмма развёртывания комплекса

2.4. Диаграммы последовательности действий

Диаграммы последовательности действий при выполнении прецедентов “регистрация”, “запрос новой задачи” и “завершение выполнения задачи” приведены на рис. 15, 16 и 17.

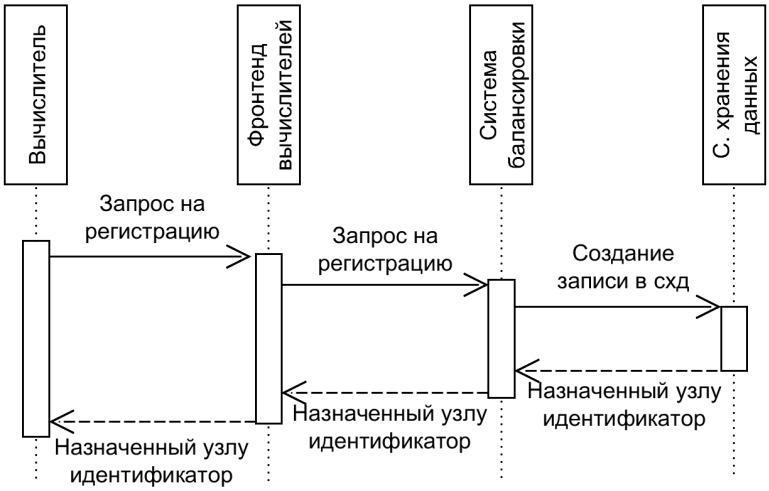

Рис. 15: Диаграмма последовательности действий прецедента “регистрация” ФВУ

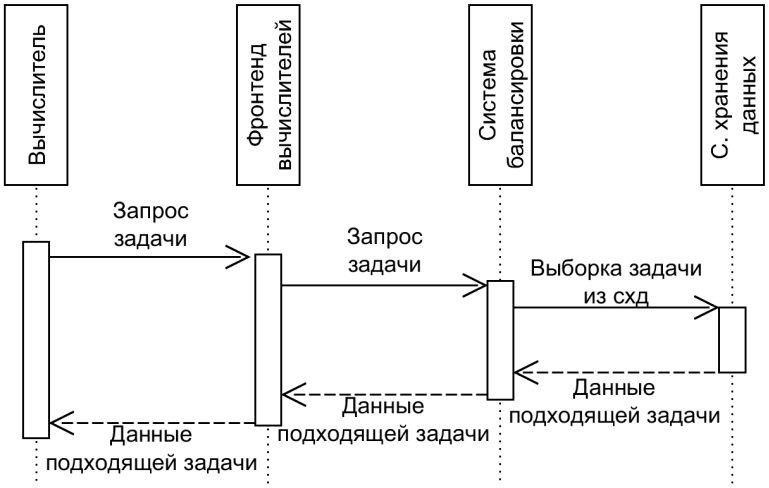

Рис. 16: Диаграмма последовательности действий прецедента “запрос новой задачи” ФВУ

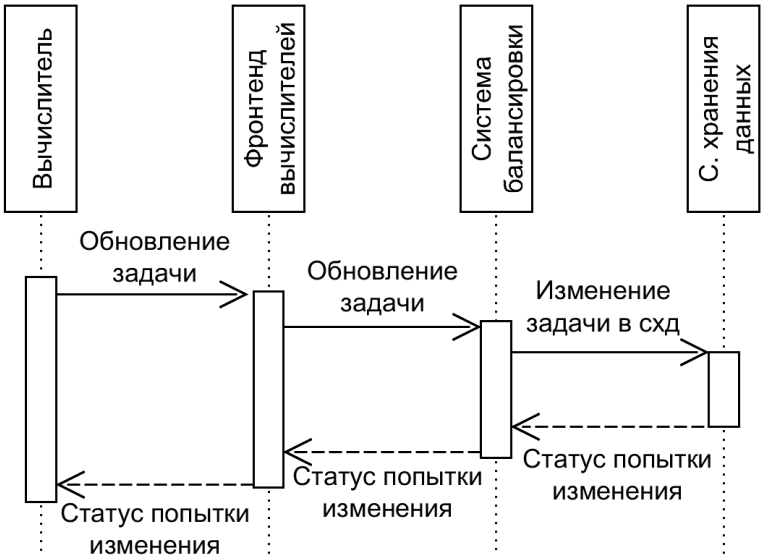

Рис. 17: Диаграмма последовательности действий прецедента “завершение выполнения задачи” ФВУ

2.5. ER-диаграмма

Связи между разными типами хранимых СХД данных предоставлены в виде ER-диаграммы на рис. 18.

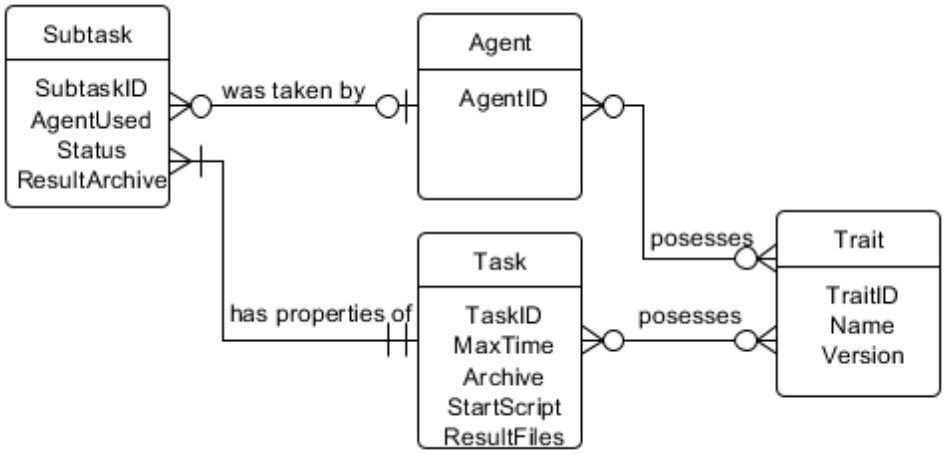

Рис. 18: ER-диаграмма сущностей, хранимых в СХД

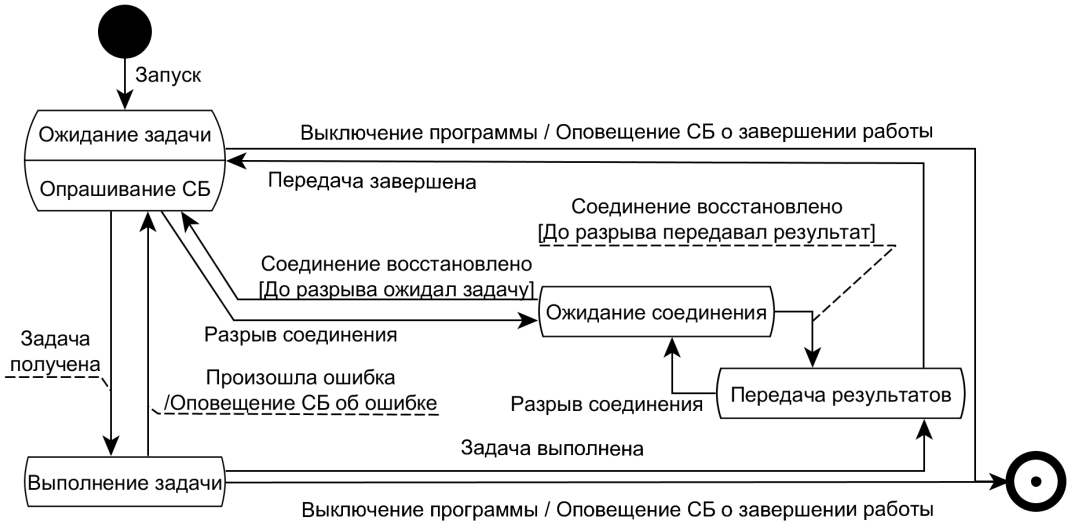

Рис. 19: Диаграмма состояний ВУ

2.6. Диаграмма состояний

ПО вычислительного узла, обеспечивающее функционирование системы, должно удовлетворять следующим требованиям:

- До подключения к серверу балансировки приложение должно предоставлять возможность формирования списка черт, характеризующих АО и ПО вычислительного узла
- После подключения к балансировщику (через фронтэнд вычислительных узлов), с определённой периодичностью вычислительный узел должен опрашивать комплекс на предмет наличия доступных задач
- По получении задачи, вычислительный узел должен с определённой периодичностью оповещать балансировщик о ходе выполнения задачи
- По завершении выполнения задачи, вычислительный узел должен передать балансировщику сведения о результате выполнения задачи

Диаграмма состояний ПО вычислительного узла, иллюстрирующая приведённые выше соображения, приведена на рис. 19.

3. Тестирование

3.1. Модульное тестирование

Статический класс DataMethods:

Представляет собой интерфейс бекенда данных.

Производится тестирование класса DataMethods с использованием методики тестирования – разбиение на уровне класса на категории по функциональности. Категория объединяет в себе методы класса, выполняющие близкую по смыслу функциональность.

Методы класса можно разбить на 3 категории по функциональности:

- методы получения данных;
- методы изменения данных;
- методы удаления данных.

Таблица 1: Методы класса DataMethodsFilter

Название метода	Примечания
get_item	param: table [str] param: value_json[dict] Возвращает все элементы таблицы с именем table, удовлетворяющие фильтру value_json
put_item	param: table [str] param: value_json [dict] Изменяет все элементы таблицы с именем table, удовлетворяющие фильтру value_json
delete_item	param: table [str] param: value_json [dict] Удаляет все элементы таблицы с именем table, удовлетворяющие фильтру value_json

Таблица 2: Категория 1 – Тестирование метода получения данных

Название теста	TestGetItemEmpty
Тестируемый метод	get_item
Описание теста	Проверка получения данных из БД, пустой таблицы trait
Ожидаемый результат	Словарь с пустым списком в ключе “result”
Степень важности	Фатальная
Результат теста	Тест пройден

Таблица 3: Категория 1 – Тестирование метода получения данных

Название теста	TestGetItemGeneral
Тестируемый метод	get_item
Описание теста	Проверка изменения данных в БД, в таблице trait, предварительно наполненной записями name = “test_name”, version = “1.0” name = “test_name”, version = “2.0”, по фильтру {“name”:“test_name”}
Ожидаемый результат	Словарь {“result”:[{“name”:“test_name”, “version”:“1.0”}, {“name”:“test_name”,} “version”:“2.0”}]}
Степень важности	Фатальная
Результат теста	Тест пройден

Таблица 4: Категория 2 – Тестирование метода изменения данных

Название теста	TestPutItem
Тестируемый метод	put_item
Описание теста	Проверка изменения данных в БД, в таблице trait, предварительно наполненной записями name = “test_name”, version = “1.0” name = “test_name”, version = “2.0”, по фильтру {“name”:“test_name”,“changes”: {“version”:“3.0”}}
Ожидаемый результат	Словарь {“count”: 2}, отражающий наличие двух произведенных изменений в БД
Степень важности	Фатальная
Результат теста	Тест пройден

Таблица 5: Категория 3 – Тестирование метода удаления данных

Название теста	TestDeleteItem
Тестируемый метод	delete_item
Описание теста	Проверка удаления данных в БД, в таблице trait, предварительно наполненной записями name = “test_name”, version = “1.0” name = “test_name”, version = “2.0”, по фильтру {“name”:“test_name”}
Ожидаемый результат	Словарь {“count”: 2}, отражающий наличие двух произведенных удалений в БД
Степень важности	Фатальная
Результат теста	Тест пройден

Вывод по результатам тестирования

Все тесты пройдены успешно, класс готов к использованию.

3.2. Системное тестирование

Производится системное тестирования файлового сервера (стратегия чёрного ящика). Тест покрывает все прецеденты взаимодействия с файловым сервером.

При формировании тестовых наборов использовалась методика эквивалентного разбиения для входных данных.

Классы эквивалентности для входных данных:

Параметр	Допустимые классы эквивалентности	Недопустимые классы эквивалентности
Имена файлов	Строки, не содержащие символы &,$,/,:,*,? и другие спецсимволы	Строки, содержащие запрещённые символы
Адрес сервера	Строки вида 'http://address:port'	Строки, не подходящее под описание
Удалённый ресурс	Такие же строки, как и допустимые для имён файлов	Такие же строки, как и недопустимые для имён файлов
Запрос-файл	Запрос с содержимым файла в теле и с хедером 'Content-type':'multipart/form-data'	Запрос без необходимого заголовка либо испорченным телом

В ходе тестирования использовался бинарный файл 'a.out', приведённый в приложении.

Тесты:

№	Описание теста	Входные данные	Ожидаемый результат	Полученный результат
1	Проверка возможности сохранения файла	Файл 'a.out'	Сообщение об успешном сохранении файла	Сообщение протокола HTTP 200 OK, пустой JSON-объект
2	Проверка доступа к существующему файлу	Запрос по адресу файла: '/static/a.out'	Содержимое файла 'a.out'	Сообщение протокола HTTP 200 OK, содержимое искомого файла
3	Проверка удаления существующего файла	Запрос на удаление файла по адресу '/static/a.out'	Сообщение об успешном удалении файла	Сообщение протокола HTTP 200 OK, пустой JSON-объект
4	Проверка удаления несуществующего файла	Запрос на удаление файла по несуществующему пути '/static/a.out'	Сообщение об ошибке	Сообщение протокола HTTP 404 NOT FOUND, JSON-объект с сообщением об ошибке
5	Проверка доступа к несуществующему файлу	Запрос по адресу несуществующего файла '/static/a.out'	Сообщение об ошибке	Сообщение протокола HTTP 404 NOT FOUND, JSON-объект с сообщением об ошибке

В ходе тестирования использовались бинарные файлы 'a.out', 'b.out' и 'c.out', приведённые в приложении.

Подробное описание тестов:

Тестирование надёжности и доступности серверов

№	Описание теста	Входные данные	Ожидаемый результат	Полученный результат
1	Провести тест, в котором несколько пользователей одновременно загружают файл на сервер	Пользовательские файлы 'a.out', 'b.out' и 'c.out'	Все файлы загружены по верным адресам	Каждый из пользователей успешно загрузил файл на сервер
2	Провести тест, в котором происходит разрыв соединения в ходе загрузки файла	Пользовательский файл 'a.out'	Загрузка возобновляется по восстановлении соединения	Возникла ошибка. Необходима повторная загрузка файла.

Отчёт об обнаруженных ошибках:

Сервис: файловый сервер	
Степень важности: средняя	
Надёжность	Сервис некорректно отрабатывает сценарий разрыва соединения

Выводы по результатам системного тестирования

Сервис может быть использован только в средах, где сеть можно считать достаточно надёжной. Для использования в рамках сетей где часты разрывы необходимо производить доработку сервиса.

Код теста

```
#!/bin/bash

echo "#include <iostream>" > test.cpp
echo "int main(){std::cout << \"HELLO WORLD!\" << std::endl;}" >> test.cpp
g++ test.cpp -o a.out

```

```
7  set -v
8
9  curl @"localhost:50002/static"                  -X POST -F file=@a.out; echo
10 curl @"localhost:50002/static?path=1\2\3\4" -X POST -F file=@a.out; echo
11
12 curl @"localhost:50002/static/a.out"          -X GET > a__.out   ; echo
13 chmod +x a__.out                                                 ; echo
14 ./a__.out                                                        ; echo
15
16 curl @"localhost:50002/static/1\2\3\4\a.out" -X GET > a_.out     ; echo
17 chmod +x a_.out                                                  ; echo
18 ./a_.out                                                         ; echo
19
20 curl @"localhost:50002/static/a.out"          -X DELETE          ; echo
21 curl @"localhost:50002/static/1\2\3\4\a.out" -X DELETE          ; echo
22
23 curl @"localhost:50002/static/a.out"          -X DELETE          ; echo
24 curl @"localhost:50002/static/1\2\3\4\a.out" -X DELETE          ; echo
25
26 curl @"localhost:50002/static/a.out"          -X GET             ; echo
27 curl @"localhost:50002/static/1\2\3\4\a.out" -X GET             ; echo
28
29 rm a*.out test.cpp
```

3.3. Интеграционное тестирование

Производится интеграционное тестирование подсистем работы с файлами, данными и подсистемы мониторинга. Целью тестирования являеется проверка корректности регистрации подсистем на маяке, играющем ключевую роль в работе распределенной системы.

Интерфейс подсистемы мониторинга(категория – beacon, сокращенно – маяк):

- GET /services/<service_group>
- PUT /services/<service_group>/<service_host>:<service_port>
- POST /services

Методы подсистемы работы с файлами(категория – filesystem):

- beacon_setter
- beacon_getter

Методы подсистемы работы с данными(категория – database):

- beacon_setter
- beacon_getter

Таблица 6: Тестирование регистрации бекендов на не запущенном маяке

Название файла	TestNoBeaconGetter.sh
Взаимодействующие подсистемы	database, filesystem, beacon
Описание теста	Подсистемы filesystem и database выполняют метод beacon_setter при запуске
Начальные условия Ожидаемый результат	beacon не запущен подсистемы не могут подключиться к beacon и сообщают об этом пользователю; повторный поиск происходит регулярно, с интервалом в 10 секунд
Степень важности	Фатальная
Результат теста	Тест пройден

Таблица 7: Тестирование регистрации бекендов на запущенном маяке

Название файла	TestBeaconPoster.sh
Взаимодействующие подсистемы	database, filesystem, beacon
Описание теста	Подсистемы filesystem и database выполняют метод beacon_setter при запуске
Начальные условия	beacon запущен
Ожидаемый результат	Подсистемы подключаются к маяку и выполняют POST-запрос на адрес /services/<категория бекенда>, передавая через JSON свою адрес Журнал маяка: POST /services/database HTTP/1.1 200 POST /services/filesystem HTTP/1.1 200
Степень важности	Фатальная
Результат теста	Тест пройден

Таблица 8: Тестирование регистрации бекендов на запущенном маяке

Название файла	TestBeaconPutter.sh
Взаимодействующие подсистемы	database, filesystem, beacon
Описание теста	Подсистемы filesystem и database выполняют метод beacon_setter повторно
Начальные условия	beacon запущен, подсистемы уже выполнили первичный POST запрос
Ожидаемый результат	Подсистемы подключаются к маяку и выполняют PUT-запрос на адрес /services/<категория бекенда>/<адрес бекенда>:<порт>. Пример журнала маяка: PUT /services/database/localhost:5000 HTTP/1.1 200 PUT /services/filesystem/localhost:5001 HTTP/1.1 200
Степень важности	Фатальная
Результат теста	Тест пройден

Таблица 9: Тестирование регистрации бекендов на запущенном маяке

Название файла	TestBeaconGetter.sh
Взаимодействующие подсистемы	filesystem, beacon
Описание теста	Подсистема filesystem запрашивает адреса всех подсистем database у маяка
Начальные условия	beacon запущен, подсистемы уже выполнили первичный POST запрос
Ожидаемый результат	filesystem подключаются к маяку и выполняет GET-запрос(метод beacon_getter) на адрес /services/database/. Маяк отвечает все известные адреса бекендов, категории database в JSON-формате. Пример ответа – [{“localhost”: 5000}]
Степень важности	Фатальная
Результат теста	Тест пройден

Выводы по результатам интеграционного тестирования

Интеграционное тестирование выявило фатальную ошибку в реализации подсистемы beacon, связанную с неверно документированным поведением метода доступа к элементу по ссылке в словарях многократной вложенности в языке Python.

Ошибка была устранена с использованием альтернативной библиотечной реализации этого метода.

4. Приложения

Содержимое файла a.out

Содержимое файла b.out

Содержимое файла c.out