Software Architecture Document

Черновик

История изменений

Дата	Версия	Описание	Автор
10.12.2019	1.0	Предварительный	Karinski, Andrey

1. Введение

Этот документ содержит полный архитектурный обзор системы и использует ряд архитектурных представлений в формате «4+1» (Филиппа Крачтена) для отображения различных аспектов системы. Сам документ служит для отражения важных архитектурных решений, принятых при проектировании системы.

1.1. Цель

Документ SAD содержит обзор архитектуры информационной системы «Вега», создаваемой как платформа для конструирования вычислительных процессов в области геолого-инженерных расчетов.

Содержимое документа важно в первую очередь проектной команде (разработчикам и аналитикам), но он также может быть интересен стороне заказчика, чтобы получить подробную информацию о способах реализации функциональных и нефункциональных требований.

Документ разрабатывается системным архитектором, но может дорабатываться членами проектной команды, чьих навыков достаточно для проектирования и документирования архитектуры системы.

Первая версия документа создавалась после аналитической работы, но до перехода к программированию, следовательно, содержит лишь архитектурные концепции, которых следует придерживаться, но никак не подробную инструкцию по реализации. В дальнейшем документ будет дорабатываться и актуализироваться.

UML-диаграммы в документе были созданы в инструменте «Sparx Enterprise Architect».

1.2. Область действия

Документ зависит от «Видения» (описания концепции системы), перечня «Вариантов использования», представляющих описание функциональных требований, и «Дополнительной спецификации», содержащей нефункциональные требования и атрибуты качества системы.

Документ оказывает влияние на «Спецификацию реализации Вариантов использования», «План развертывания» и «План измерений» (описывает метрики системы и критерии их определения).

1.3. Определения, сокращения и аббревиатуры

См. «Глоссарий».

1.4. Ссылки

Здесь должен быть представлен полный список всех документов, на которые есть ссылки в других разделах данного документа. Не забыть указать источники.

- Технологический стек: https://confluence-gpn.nexign.com/pages/viewpage.action?pageId=8555990

1.5. Обзор

Документ описывает архитектуру как серию представлений: Варианты Использования, Логическое, Процессное, Реализации, Развертывания. Каждый архитектурно важный момент может снабжаться дополнительными диаграммами, поясняющими концепцию.

2. Архитектурное представление

Архитектуры системы «Вега» концептуально представляет собой классический вычислитель бизнес-процессов. Предполагается, что пользователь системы будет создавать рабочие процессы, представляющие собой сложные геолого-инженерные расчеты, при помощи визуального конструктора, подобного BPMN-редакторам. Элементы, из которых строятся рабочие процессы (в том числе математические и логические функции) могут содержаться в общей для всех пользователей галерее, а продвинутый пользователь системы сможет создавать новые элементы (при помощи визуального дизайнера и добавления кода на языке Python) и публиковать их в галерее.

Также предполагается, что пользовательский интерфейс будет представлять собой одностраничное веб-приложение (SPA).

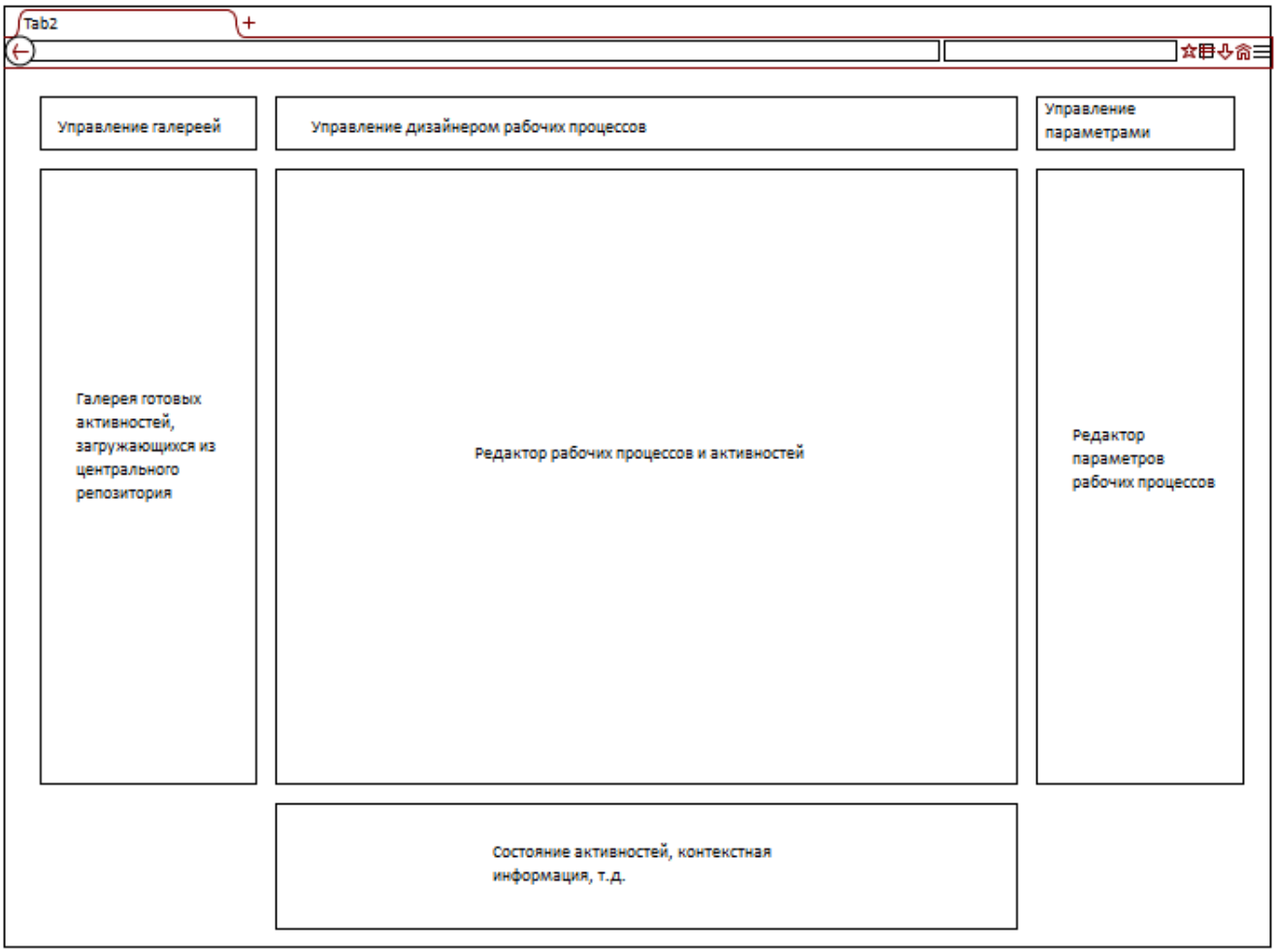

Рис.1 Предполагаемый интерфейс пользователя.

3. Архитектурные цели и ограничения

Одно их важнейших требований к системе – возможность выполнять «тяжелые» математические вычисления с распараллеливанием по многочисленным вычислительным узлам и другими приемами горизонтального масштабирования. Вычисления могут длиться значительное время, следовательно, архитектура системы должна поддерживать полную асинхронность и возможность вычислительных процессов «заснуть» (при недостатке вычислительных ресурсов) и «проснуться» (продолжив вычисления).

Другое важное требование – возможность конструировать процессы вычислений визуальными (специальный дизайнер) и невизуальными (редактор кода) средствами. Логика отдельных элементов рабочих процессов и функций математических библиотек должна описываться на языке Python.

Из вышеперечисленных требований следует, что вся система должна строиться на базе микросервисной архитектуры (для легкой поддержки горизонтального масштабирования) и в облачной инфраструктуре, позволяющей гибко выделять вычислительные ресурсы исходя из текущих потребностей. Отдельные сервисы должны размещаться в docker-контейнерах в кластере Kuberentes, что обеспечит дополнительные возможности масштабирования, управляемости и надежности.

Так как система будет состоять главным образом из инфраструктурного кода, от которого требуется высокой производительности, надежности, предсказуемости и безопасности, в качестве основного языка программирования следует выбрать Java 11. Для реализации математических и логических функций, предназначенных для модифицирования продвинутыми пользователями системы, предполагается использовать язык Python. Для создания API сервисов следует использовать REST-протокол. Для асинхронного обмена событиями между сервисами следует использовать брокер сообщений Apache

4. Представление Вариантов Использования (Use Case View)

Предполагается две основные роли в системе:

- Пользователь – конструирует из готовых элементов (в галерее) и запускает рабочие процессы
- Продвинутый Пользователь – кроме вышеперечисленного, создает новые элементы процесса и публикует их в галерее

Основные сценарии использования системы:

- Конструирование рабочего процесса
- Конструирование элемента рабочего процесса (активности или функции)
- Запуск рабочего процесса
- Мониторинг рабочего процесса

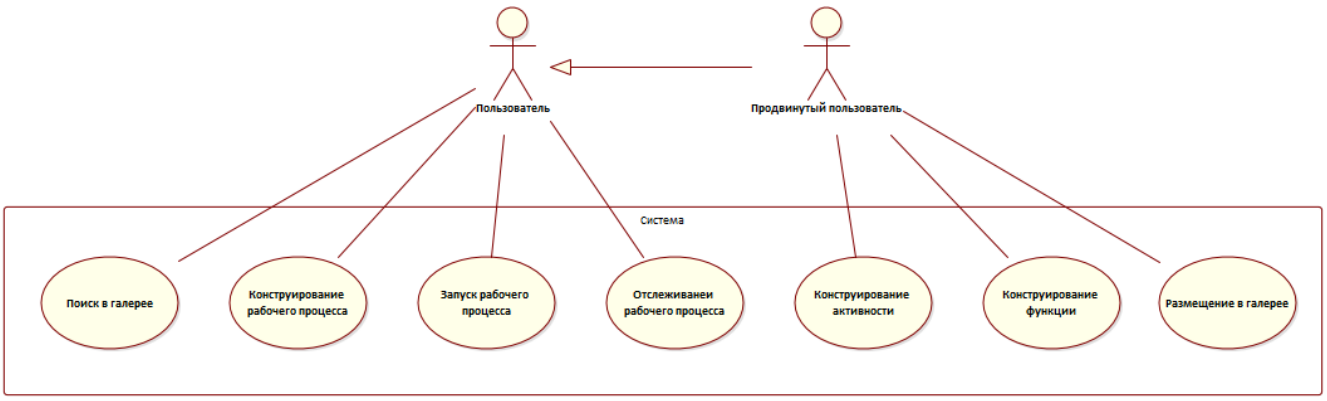

Рис.2 Диаграмма вариантов использования

Наиболее архитектурно-значимый вариант использования – «Запуск рабочего процесса», для реализации которого потребуется создать большую часть инфраструктуры системы.

В данный момент в документе не отражены сценарии администрирования системы, т.к. они не важны с точки зрения архитектуры. Кроме того, возможно, в перспективе появится потребность ввести роль Старшего Пользователя, чьими обязанностями будут разрешение на запуск некоторых процессов или подтверждение их результатов. Также пока не рассматривается модель безопасности.

5. Логическое представление (Logical View)

Описываются архитектурно значимые части модели системы, такие как ее разбиение на подсистемы, пакеты и модули. Для каждого важного пакета потребуется описать

- *значимые классы (разумеется, без подробностей, так как они будут проясняться только в процессе реализации)*
- *их обязанности*
- *несколько важных отношений, атрибутов и операций*

Так же стоит отметить, что описанные в разделе классы являются концептуальными (т.е. артефактами моделирования, а не реализации) и могут не соответствовать программным классам из модели реализации.

5.1. Обзор

Описание общих принципов декомпозиции модели с точки зрения слоев и иерархии пакетов.

Система состоит из 4 основных подсистем, представленных на Рис.3 следующими пакетами:

- Интерфейс пользователя.
- Конструктор процессов - визуальные инструменты для построения рабочих процессов и их элементов.
- Выполнение процессов - средства управления выполнением процессов, диспетчеризации, передачи параметров и т.п.
- Мониторинг процессов - инструменты контроля состояния процессов.

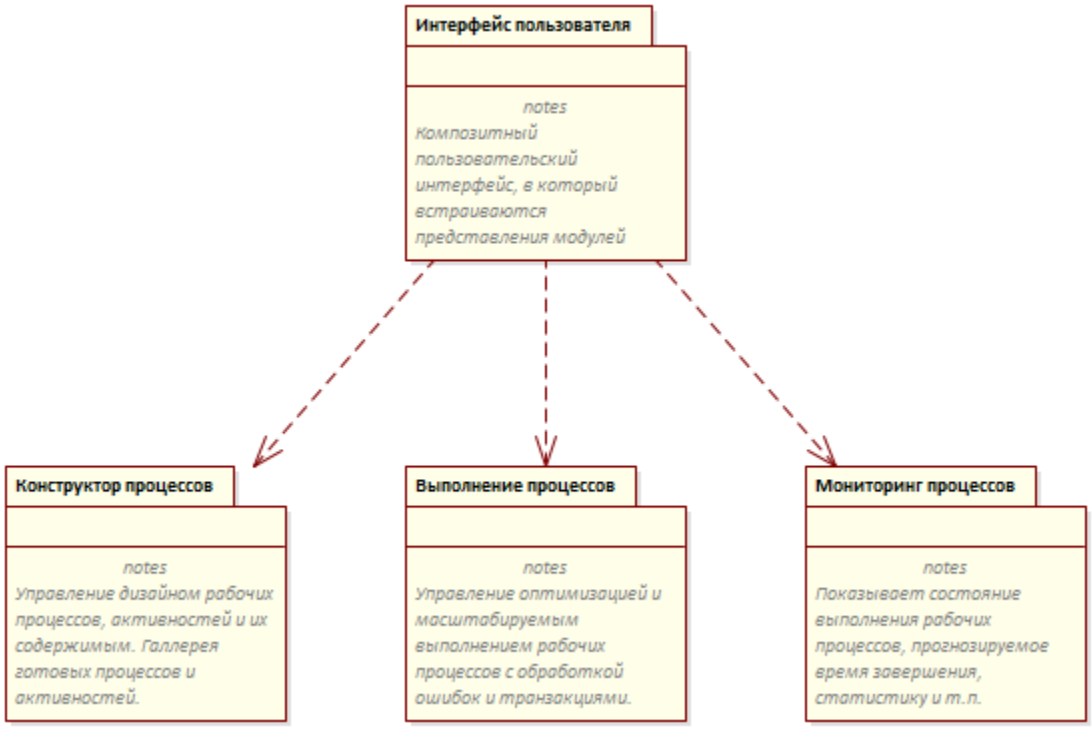

Рис.3 Основные подсистемы.

5.2. Архитектурно-значимые пакеты

Для каждого значимого пакета включить подраздел с его описанием и диаграммой со всеми значимыми содержащимися в нем классами и пакетами. Для каждого значимого класса указать его имя, краткое описание, и при необходимости некоторые из его основных обязанностей, операций и атрибутов.

Наиболее значимые с точки зрения влияния на архитектуру следующие 2 модуля: «Выполнение процессов» и «Конструирование процессов». Кроме того, еще два модуля обеспечивают их совместную работу: «Экспорт-Импорт» и «Инфраструктура».

- Инфраструктура - весь код, обеспечивающий правильное функционирование системы, например механизмы персистентности, управления транзакциями, аудит, безопасность, конфигурации.
- Импорт/Экспорт - модель сериализации (общий формат обмена данными), конвертеры и интеграционные веб-сервисы.

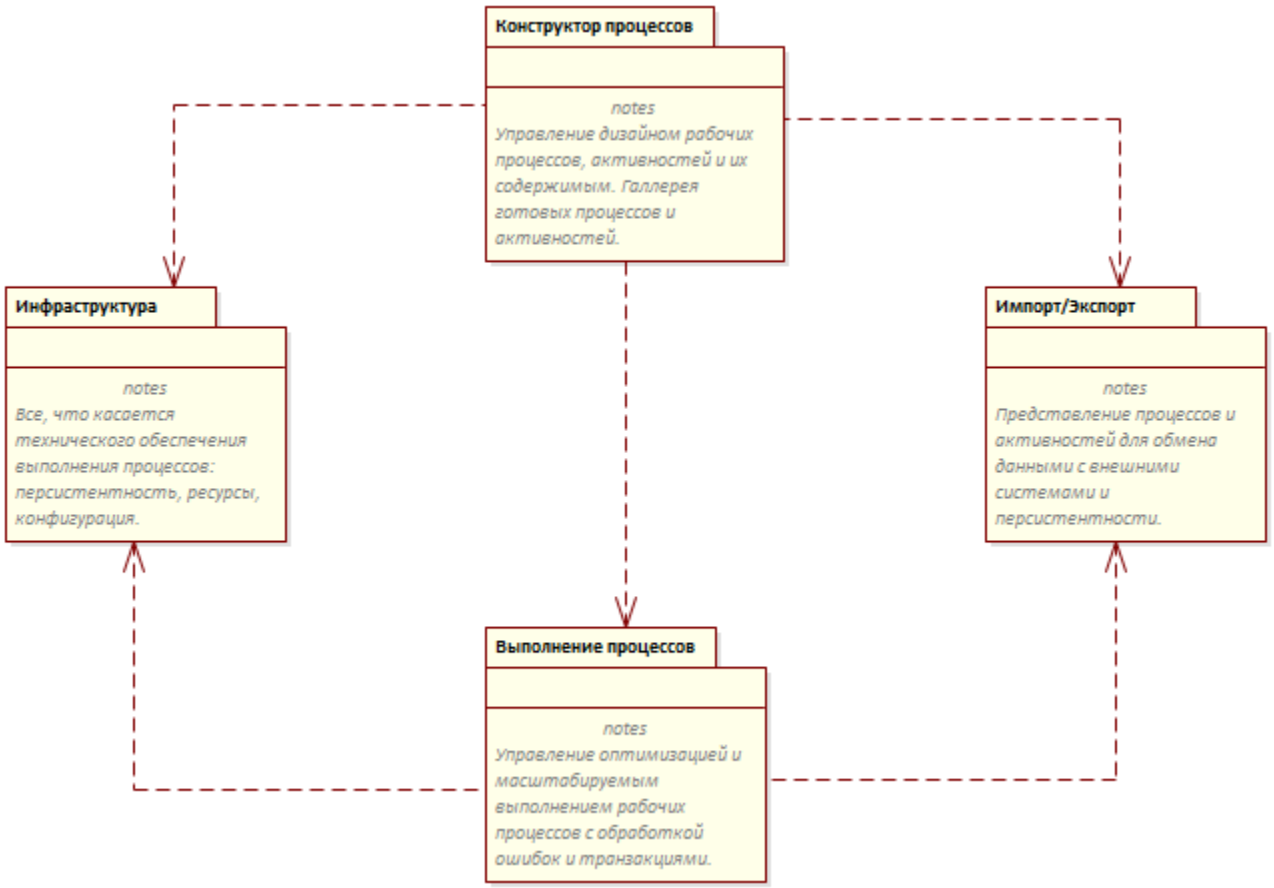

Рис.4 Архитектурно-значимые модули.

Отдельно следует рассмотреть пакет "Доменная модель", содержащий сущности, то есть классы предметной области. Отдельно он рассматривается потому, что с точки зрения архитектуры, доменная модель в данном случае не слишком важна, основа системы - рабочие процессы.

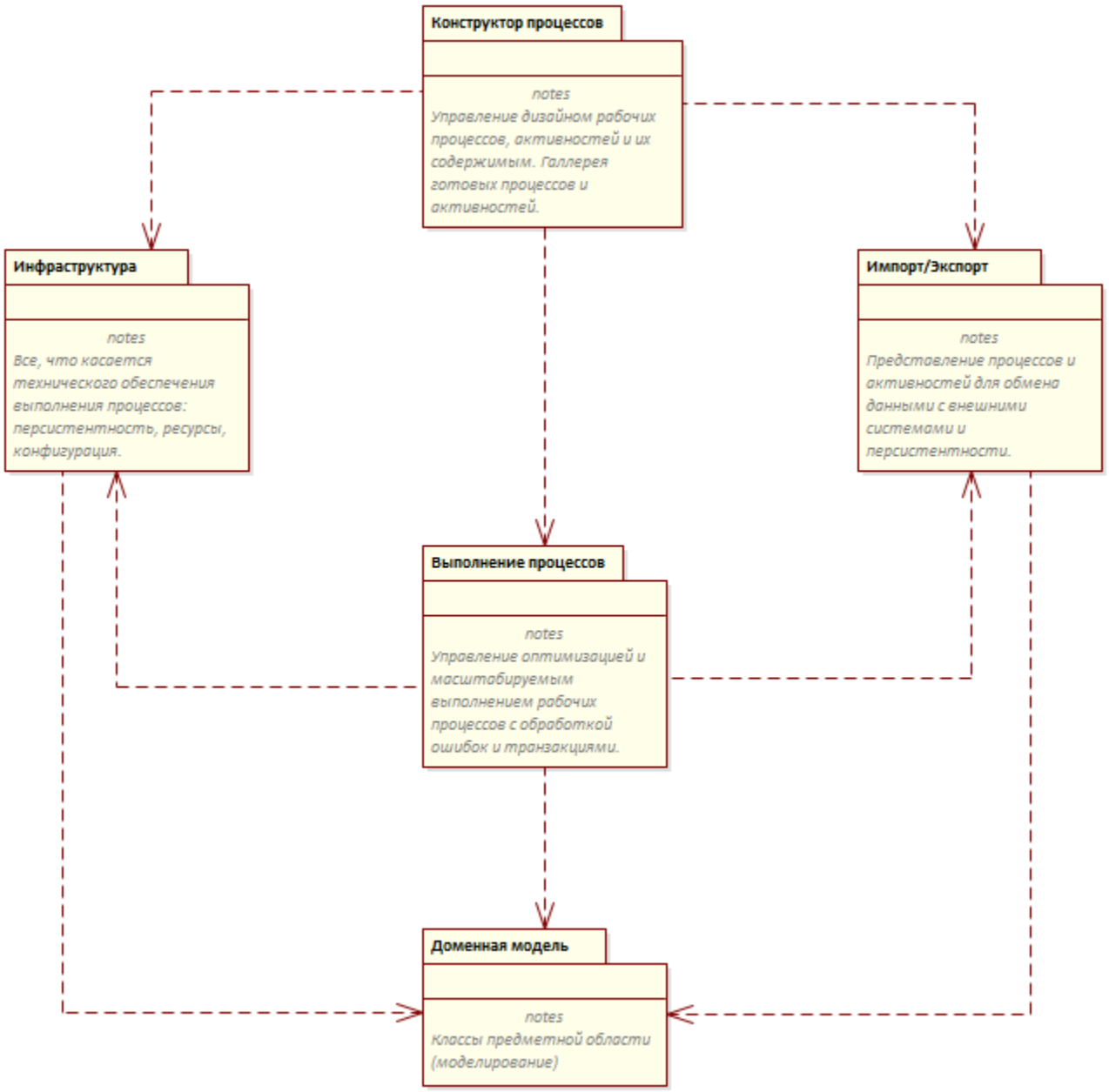

Рис.5 Архитектурно-значимые модули и доменная модель.

На рис.6. показано, что пакет "Доменная модель" содержит основные классы предметной области.

Детальное описание классов и их обязанностей должно быть добавлено в следующих версиях документа, однако их общая структура уже приблизительно определена.

Основополагающей сущностью является "", описывающий произвольную (описываемую ассоциацией " ") иерархию геолого-административных объектов, которые в свою очередь содержат набор "", как заданных заранее, так и вычисленных ("") при применении к "" некоторых " ", содержащих наборы "".

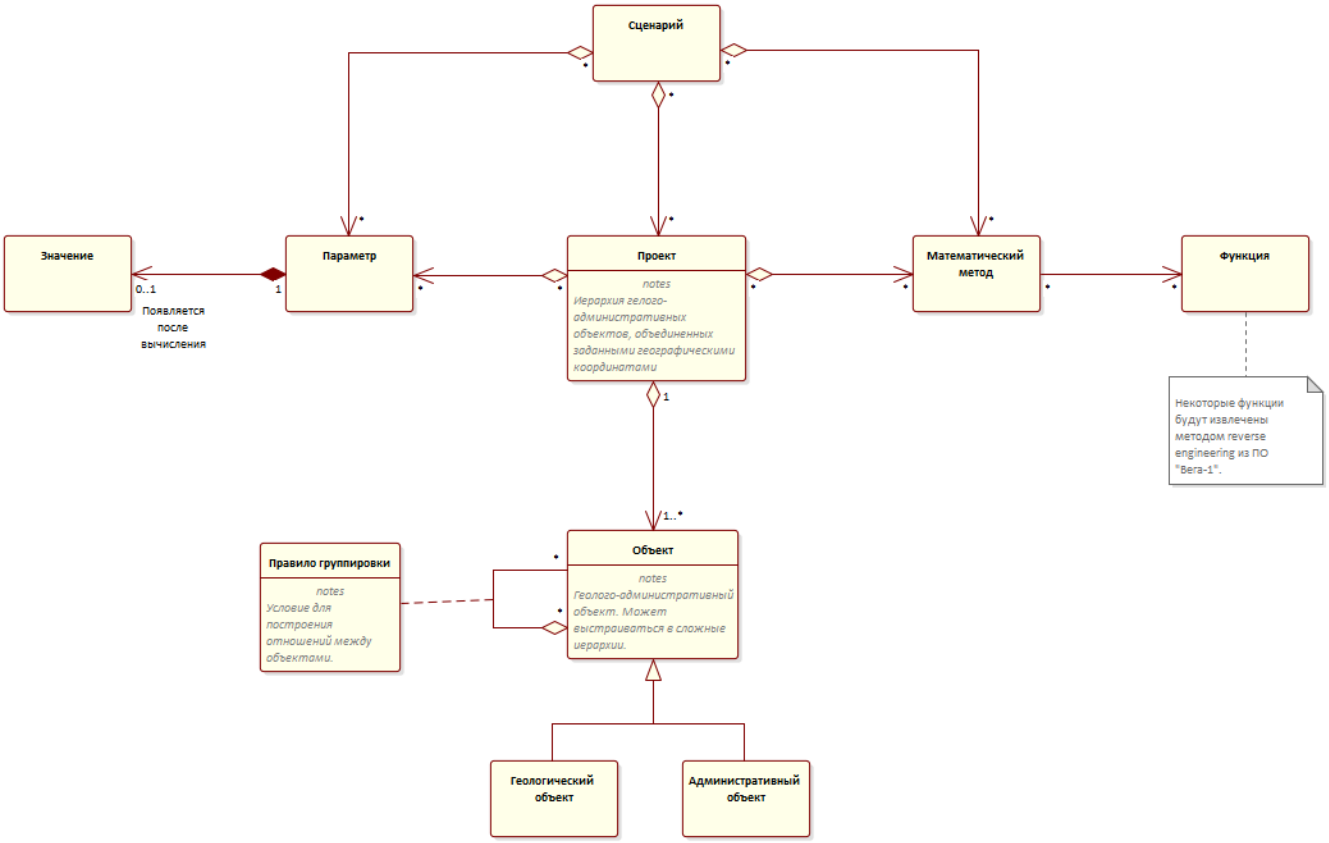

Рис.6 Модель предметной области.

Следует обратить внимание на сущность "". В рамках работ по обратному конструированию системы "Вега-1" предстоит часть полученных функций перенести в создаваемую систему и встроить в ее реестр функций. На рис.7 показано, как это будет по отношению к системе.

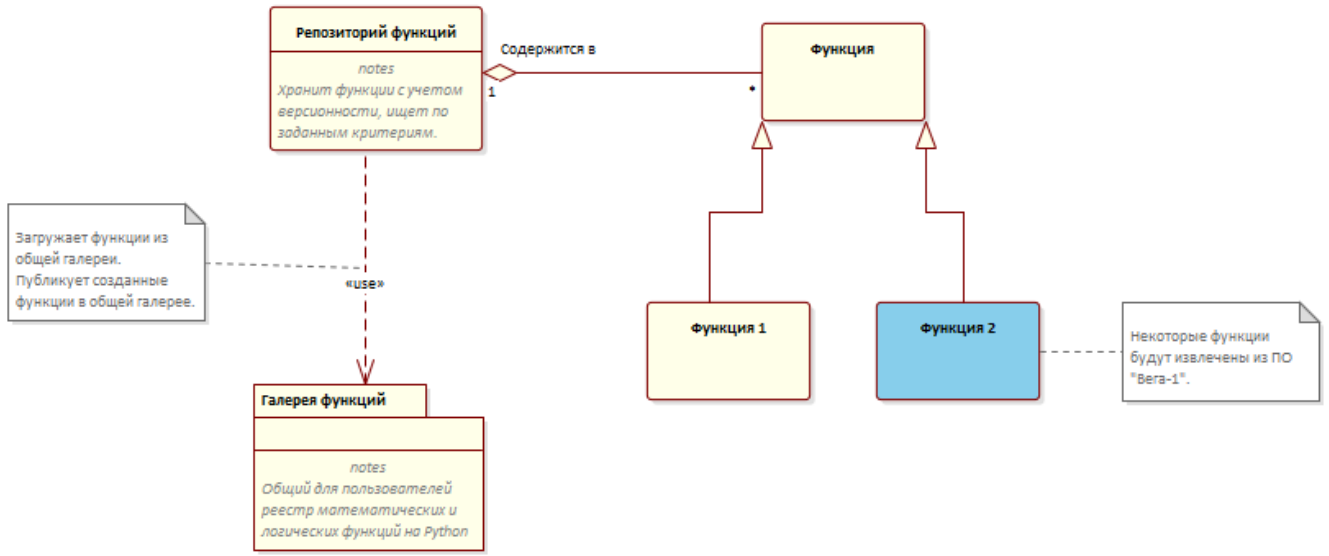

Рис.7 Функции.

Если более детально рассмотреть зависимости основных подсистем, то связь между «Конструктором процессов» и «Выполнением процессов» выглядит следующим образом:

- Активности - элементы, из которых строятся рабочие процессы, как атомарные базовые (например, булевы операторы, множественный выбор и т.п.), так и предназначенные для редактирования пользователем (например, параметризуемые мат.алгоритмы).
- Функции - содержит атомарные математические функции, написанные на Python, реестр для хранения и поиска функций, а так же средства взаимодействия с ними.
- Конструктор функций - визуальные инструменты для создания функций, их отладки, версионирования, комментирования и т.п.

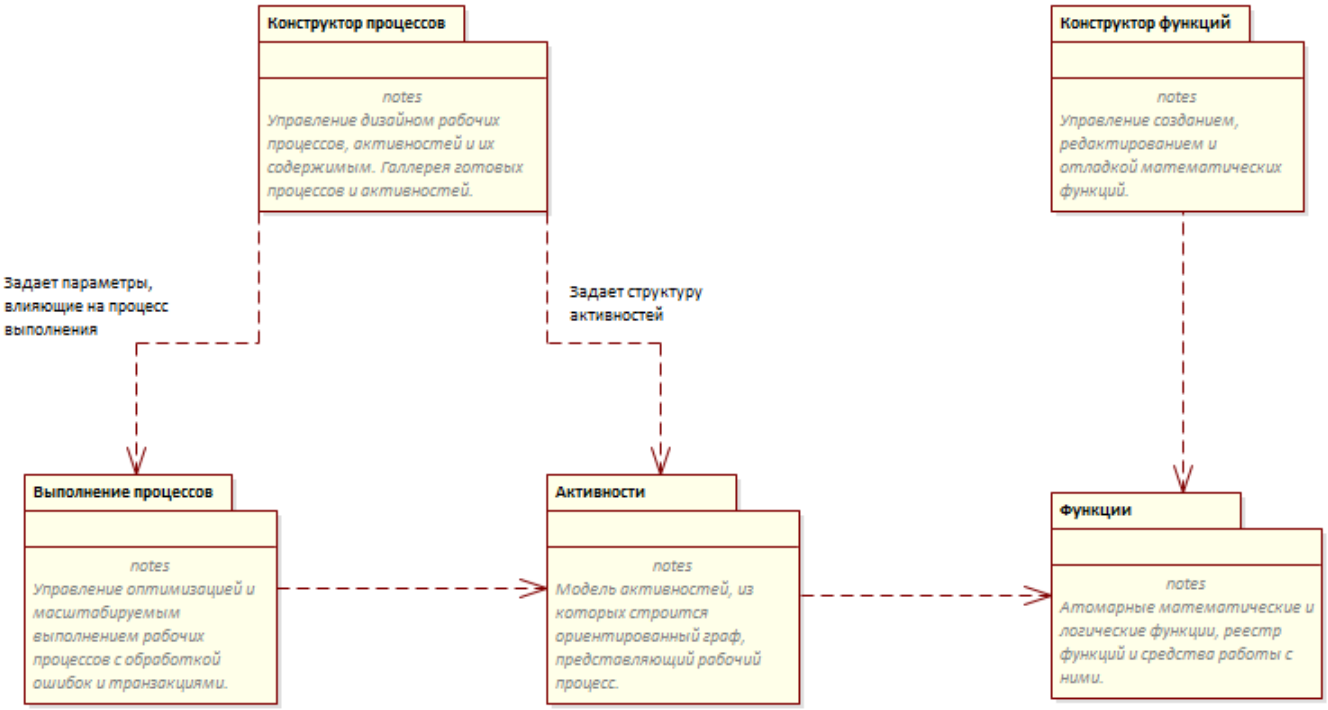

Рис.6. Конструктор процессов и его связь с активностями.

В конструкторе процессов предполагается как создавать его структуру/схему, так и настраивать/конфигурировать параметры и метаданные, влияющие на выполнение процесса (например, приоритеты, входные и выходные аргументы). Кроме того, активности (элементы рабочего процесса) могут быть композитными (состоять из других активностей), но и содержать атомарные функции, которые редактируются и отлаживаются на языке `Python` прямо в пользовательском интерфейсе.

Можно еще детальнее рассмотреть модули системы, связанные с взаимодействием основных подсистем с пользователем. На рис.7 показано, что пользовательский интерфейс ссылается на конструкторы процессов и функций с соответствующими галереями готовых элементов, и на систему мониторинга, позволяющую следить за выполнением рабочих процессов.

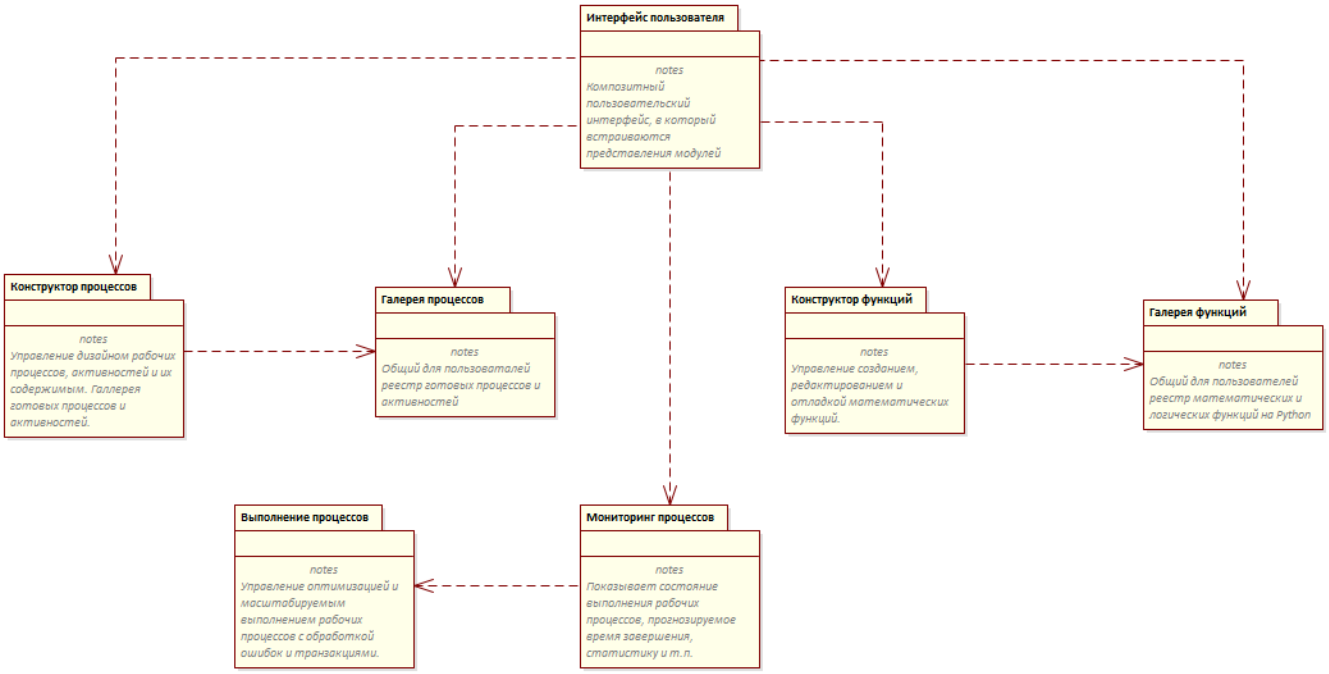

Рис.7 Пользовательский интерфейс и основные подсистемы.

5.3. **Реализация Вариантов использования**

Раздел иллюстрируют, как система работает внутри, посредством описания реализаций нескольких выбранных (и важных) сценариев; объясняет, как для этого используются различные элементы модели.

На этапе предварительного проектирования трудно предполагать, как в деталях будет реализованы основные сценарии работы системы.

Однако стоит продемонстрировать, как приблизительно будет решаться главная задача системы - представление математических расчетов в виде рабочих процессов.

Допустим, что необходимо посчитать длину 3d-вектора. Для этого создается рабочий процесс, состоящий из атомарных арифметических операций, соединяемых друг с другом с учетом приоритетов. Результаты вычисления каждого предыдущего элемента поступают на вход следующего элемента, в результате чего формируется ориентированный (циклический, в случае, если алгоритм содержит условные ветвления) граф. Как можно заметить по рис.8, некоторые компоненты графа не зависят напрямую друг от друга, следовательно, могут быть вычислены параллельно. Кроме того, некоторые операции могут быть выполнены условно "быстро" (вычитания, сложения, присвоения), а другие - условно "медленно" (вычисление квадратного корня), что может быть использовано для различных оптимизаций (например, планирования выделения вычислительных ресурсов).

$$|\overrightarrow{AB}| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (2z_2 - z_1)^2}$$

Пример представления математической формулы в виде рабочего процесса

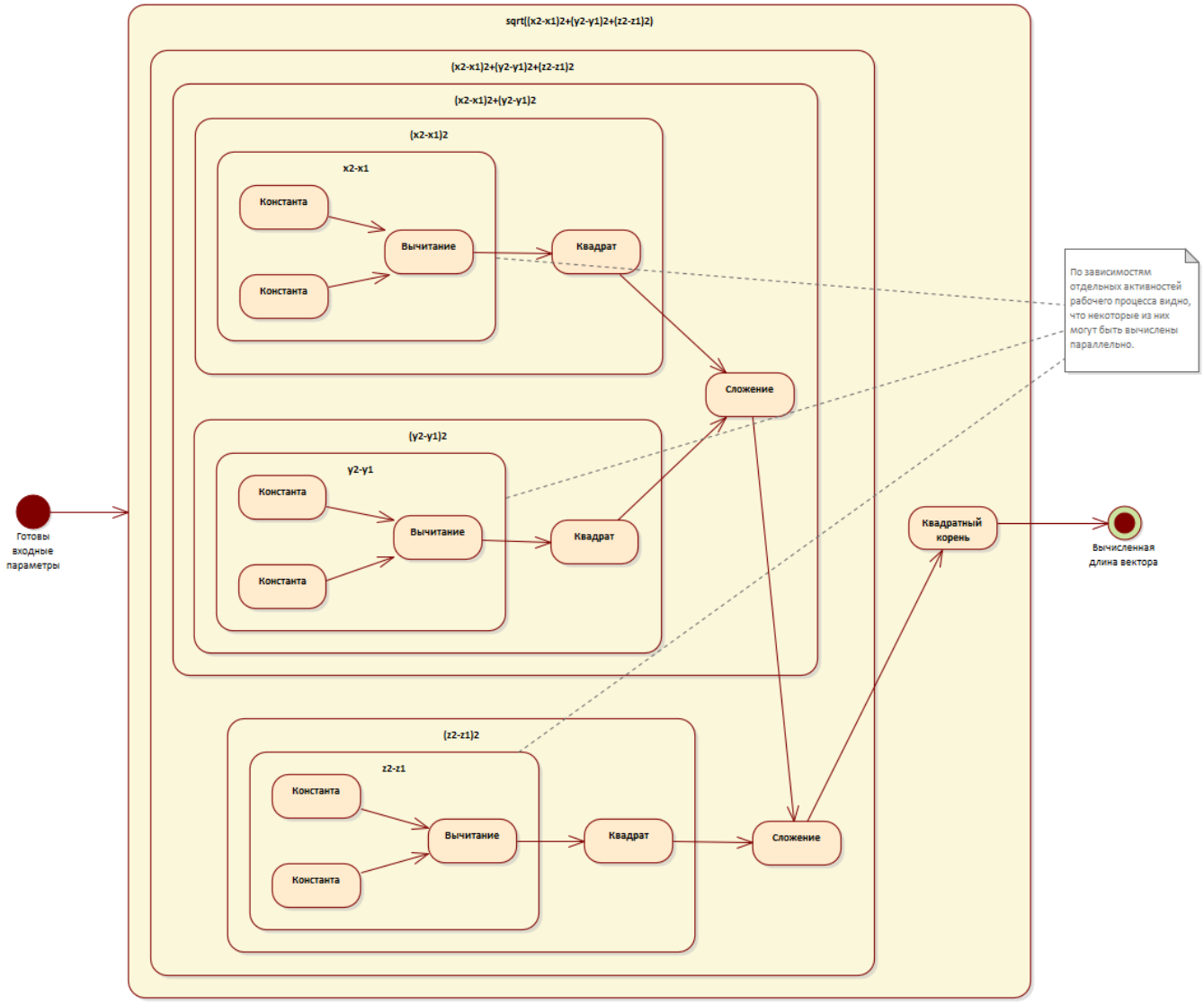

Рис.8 Представление математической формулы в виде рабочего процесса.

6. Процессное представление (Process View)

Описание декомпозиции системы на бизнес-процессы. Удобно организовать раздел по группам взаимодействующих процессов. Хорошо так же проследить зависимость от соответствующих use cases.

Описание вариантов использования в виде процессов будет в следующей версии документа.

7. Представление реализации (Implementation View)

Описание общей структуры модели реализации, декомпозиция на уровни и подсистемы, любые архитектурно значимые элементы реализации.

В данной версии документа основное внимание уделяется не модели предметной области, а модели построения и выполнения рабочих процессов.

7.1. Обзор

Описание основных уровней и слоев системы с точки зрения реализации, правила разделения на слои, границы между слоями. Имеет смысл описывать в виде диаграмм компонентов.

На этапе предварительного проектирования трудно рассуждать о деталях реализации, тем не менее, можно многое сказать о том, как должна быть построена модель рабочих процессов. На рис.9 представлены основные классы, моделирующие рабочий процесс.

- Workflow (Рабочий процесс) - контейнер для активностей и содержащий необходимые метаданные.
- Activity (Активность) - базовый элемент, из которых строится рабочий процесс. Активности могут быть композитными.
- Transition (Переход) - связь/отношение между активностями (как ребро графа между вершинами).
- Parameter (Параметр) - аргумент активности, через который она обменивается данными с другими активностями, бывают входными и выходными.
- Action (Действие) - то, что выполняется при вычислении активности, может быть функцией или ссылкой на внешний потокобезопасный код.
- ContinuationPoint (Точка продолжения) - место, где работающий процесс может быть безопасно приостановлен или, наоборот, его выполнение возобновится.

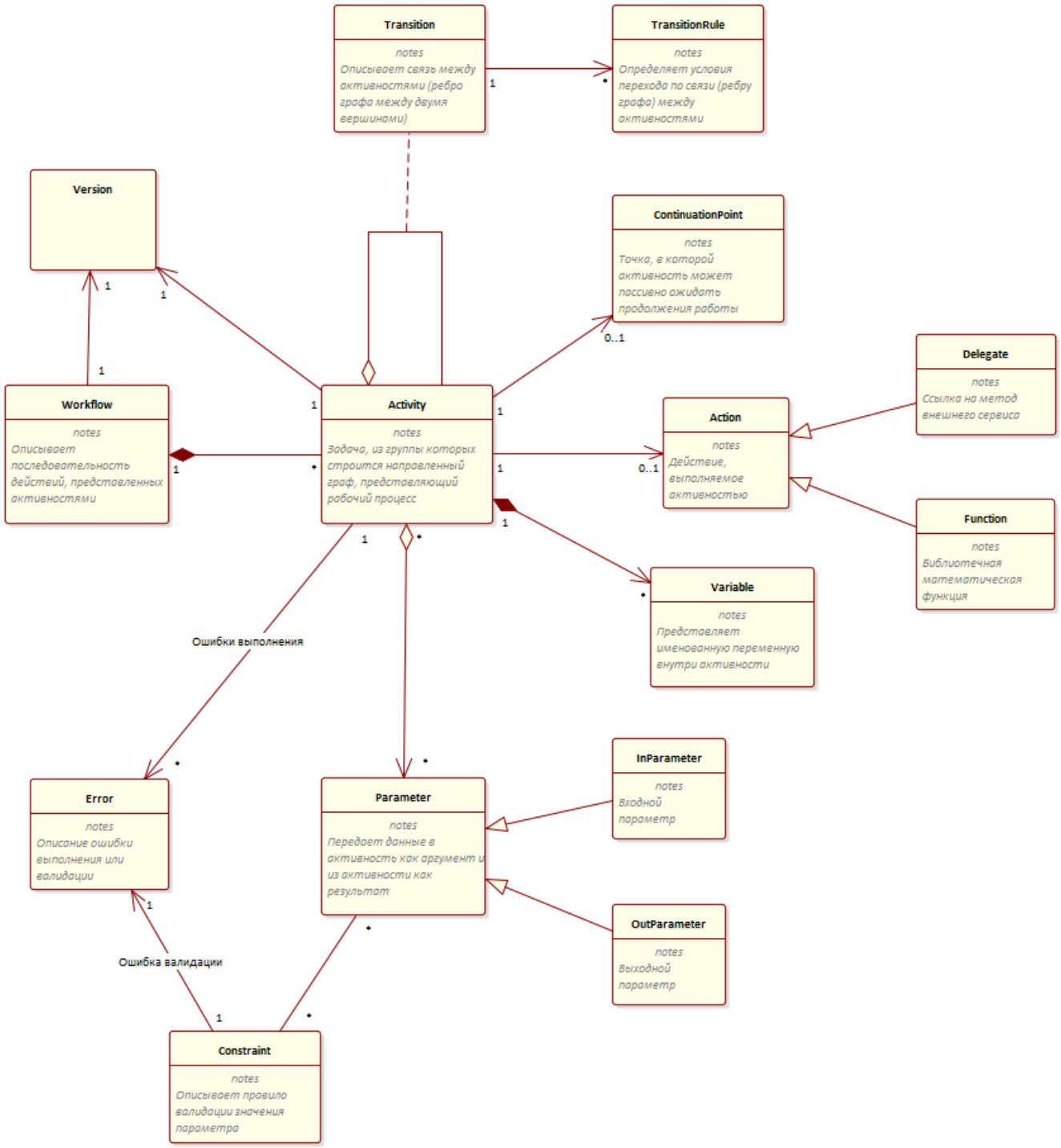

Рис.8 Основные классы рабочего процесса.

После построения рабочего процесса его можно запустить. На рис.9 рассматриваются основные классы, обеспечивающие запуск рабочего процесса и по выполнение (по возможности параллельное) его активностей на доступных системе вычислительных узлах (агентах). Непосредственно перед запуском рабочего процесса должна выполняться некоторая оптимизация, например, строиться план выполнения. То, как выполняется процесс и его активности и функции, необходимо непрерывно контролировать, что особенно важно из-за возможно большой длительности таких вычислений.

Основные классы:

- Executer (Выполнитель) - фасад, точка входа, позволяющая передать рабочему процессу необходимые параметры/аргументы и забрать результаты по окончании.
- Scheduler (Планировщик) - руководит процессом запуска рабочего процесса и получения результатов и событий.
- Optimizer (Оптимизатор) - анализирует структуру рабочего процесса, находит возможности для оптимизации (параллельные ветви алгоритма, удельный "вычислительный вес" активностей и функций) и строит план выполнения.
- Dispatcher (Диспетчер) - запускает отдельные активности из рабочего процесса на наиболее подходящих для этого доступных вычислительных узлах согласно плану выполнения.
- Agent (Агент) - вычислительный узел, может представлять собой лямбда-функцию в облачной среде, процесс со свободными ядрами и памятью или что-то подобное (система должна абстрагироваться от деталей реализации).
- AgentPool (Пул агентов) - выделяет свободные вычислительные узлы, возвращает в пул после окончания вычислений.
- Evaluator (Вычислитель) - in-process-служба, выполняющая потокобезопасное вычисление активностей и функций.

Следует отдельно отметить структуру Weight, принадлежащую активности и формируемую в момент анализа рабочего процесса Оптимизатором. Еще до запуска процесса можно предположить, что каждая его активность имеет некоторый "вычислительный вес". Кроме того, можно ввести понятие "цикломатический вес", который зависит от наличия циклов внутри графа, описывающего рабочий процесс. После анализа графа Оптимизатор, зная "вычислительный вес" каждой активности и устанавливая "цикломатический вес" на основе топологии, формирует план выполнения. Доступные вычислительные узлы могут быть различными (например, много дешевых и несколько дорогих), тогда диспетчер наиболее оптимально распределит между ними активности согласно плану выполнения.

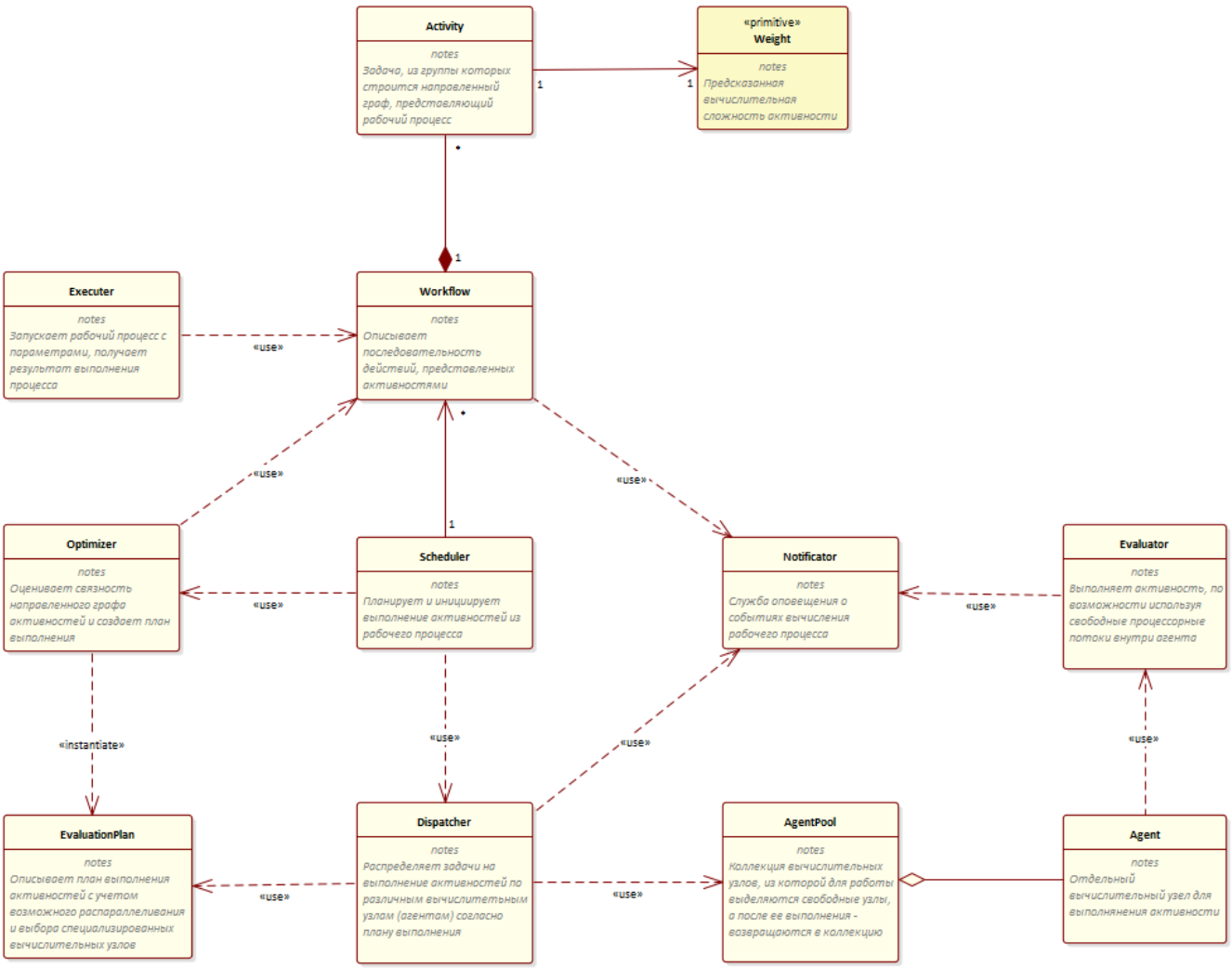

Рис.9 Планировщик рабочих процессов.

Во время выполнения процесс может быть приостановлен из-за нехватки вычислительных ресурсов или по команде оператора. В таком случае состояние процесса должно быть атомарно сохранено в БД, а непосредственно перед "пробуждением" - наоборот, загружено их БД. На рис.10 показаны основные классы для механизма персистентности.

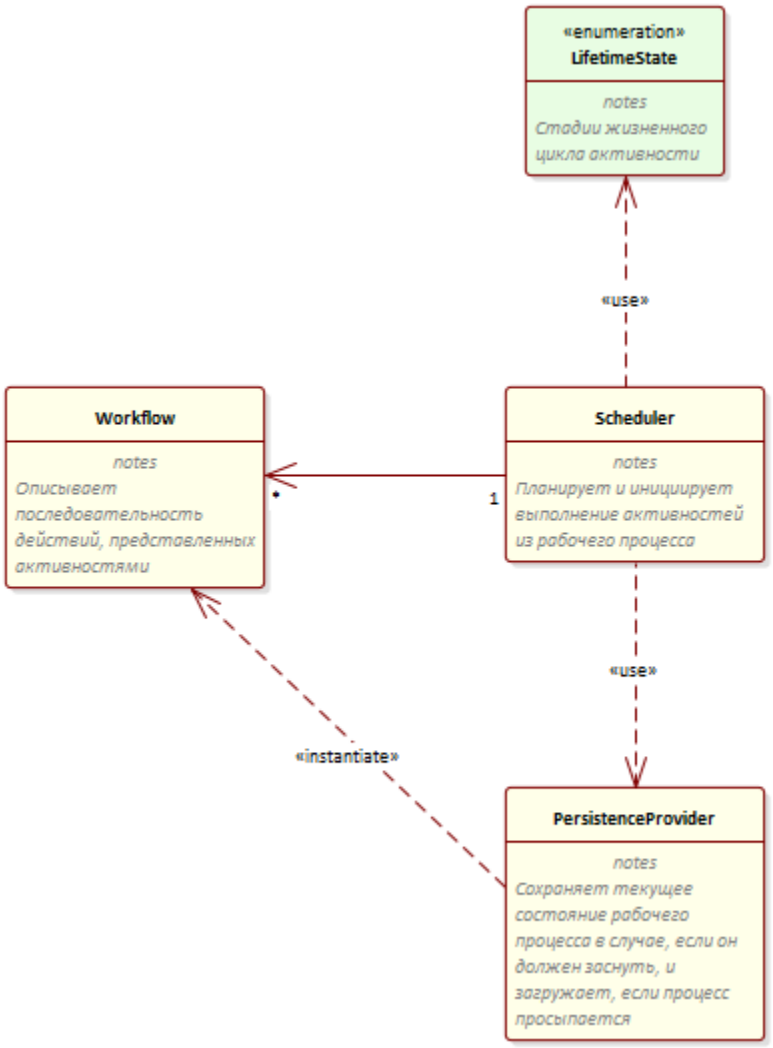

Рис.10 Персистентность рабочего процесса.

Более детально жизненный цикл рабочего процесса представлен на рис.11

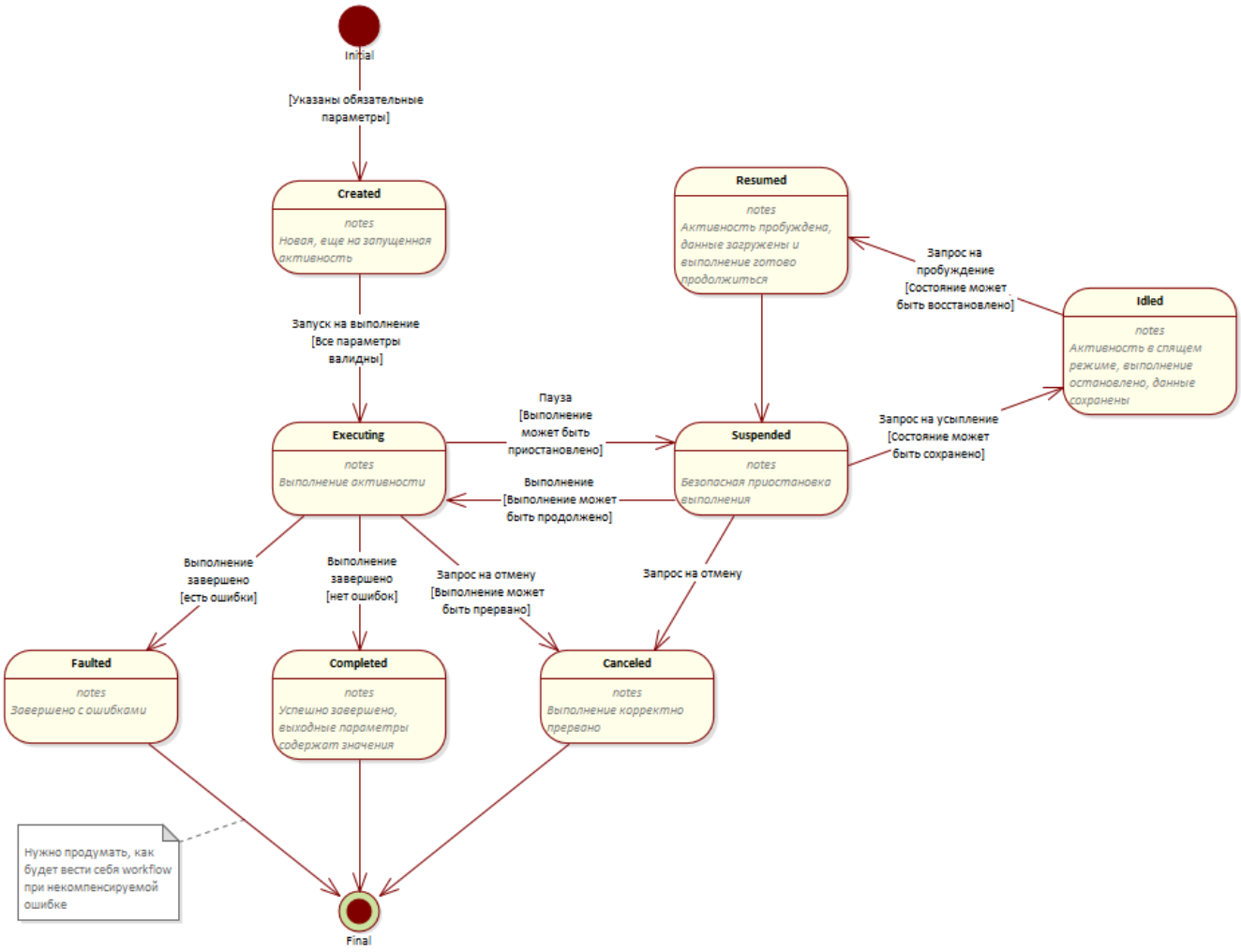

Рис.11 Жизненный цикл рабочего процесса.

7.2. **Слои**

Для каждого уровня включить подраздел с его описанием, перечнем подсистем, диаграмму компонентов.

Раздел будет дополнен в следующих версиях документа.

8. **Представление развертывания (Deployment View)**

Описание физических (сетевых, аппаратных) конфигураций, в которых будет развернута и запущена создаваемая система. Для каждой конфигурации должны быть указаны физические узлы (сервера, процессоры), выполняющие код, и их взаимосвязи (сетевые соединения, очереди, т.п.), отображение процессов на физических узлах.

На этапе предварительного проектирования невозможно предвидеть детали развертывания несуществующей системы.

Пока можно лишь предположить, что отдельные функциональные модули будут развернуты в контейнерах внутри Kubernetes, базы данных в виртуальных машинах, сервисы скрыты от внешнего мира за балансировщиком, обеспечивающим также аутентификацию.

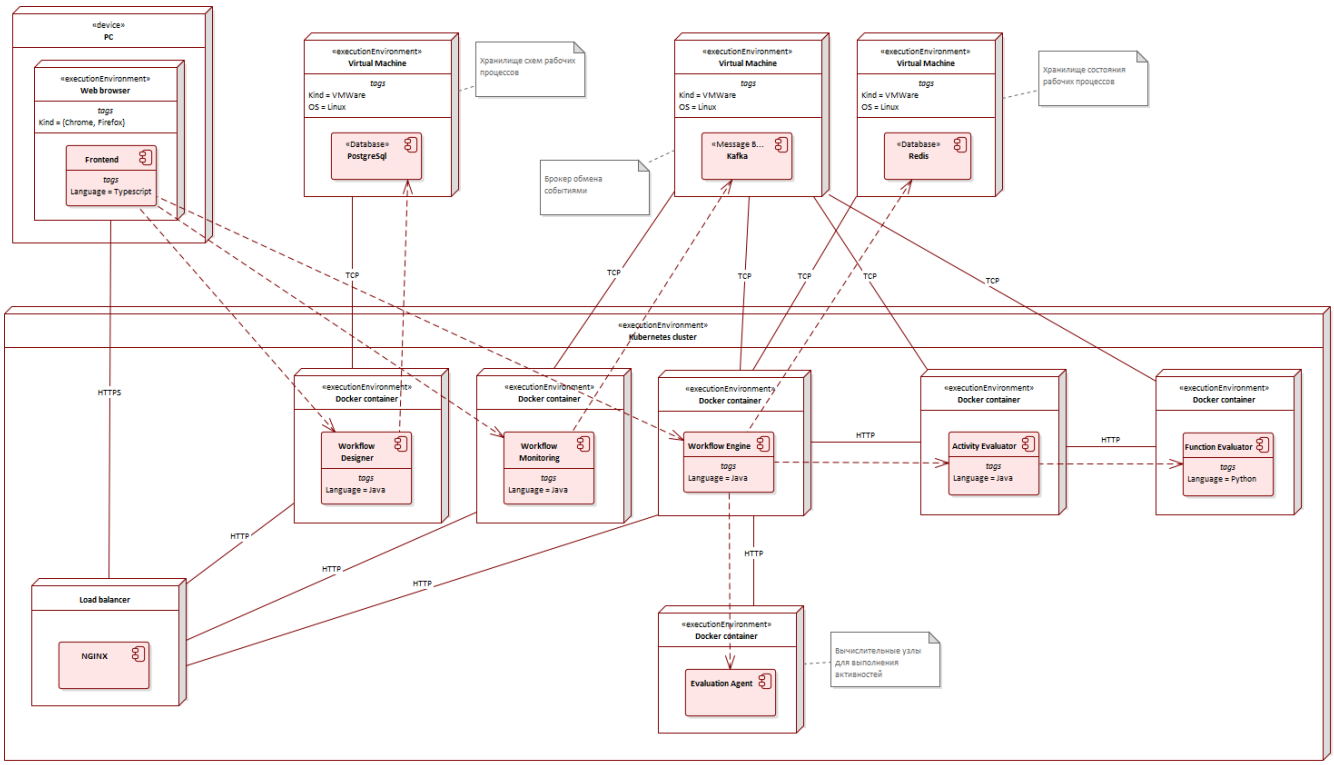

Рис.12 Схема развертывания.

9. **Представление данных (опционально)**

Описание схемы хранения данных. Раздел не является обязательным и наполняется в случае, если хранение данных нетривиально и значимо с точки зрения общей архитектуры.

На этапе предварительного проектирования невозможно описывать детали того, как будут храниться данные, однако некоторые моменты уже ясны. Например, состояние модели предметной области, шаблоны рабочих процессов, активностей, функций будут храниться в реляционной БД (например, "PostgreSQL"), т.к. к ним, предположительно, понадобится строить достаточно произвольные запросы. Однако для хранения состояния выполняющихся рабочих процессов требуется механизм, похожий на сериализацию, то есть способный максимально быстро обеспечить сохранение и загрузку всего состояния, при этом само состояние атомарно. Для данной задачи хорошо подходит key-value БД, например, "Redis".

10. **Размер и Производительность**

Описание основных характеристик системы в плане объема данных и требований к производительности, особенно влияющих на архитектуру.

В данный момент каких-либо особенных нефункциональных требований к системе не существует. Тем не менее, исходя их бизнес-требований, очевидно, что система потребует значительных вычислительных ресурсов для математических расчетов. Также очевидна потребность в существенных объемах оперативной памяти, так как для вычисления потребую многочисленных передач результатов работы элементов рабочих процессов друг другу "по значению" (для потокобезопасности и исключения блокировок), а это потребует частых операций сериализации /десериализации, требовательных к оперативной памяти и нагружающих сборщик мусора.

11. **Качество**

Описание того, как архитектура системы отражает влияние нефункциональных требований и атрибутов качества, таких, как безопасность, масштабируемость, надежность, повторное использование и т.п. Ссылка на матрицу атрибутов качества.

Влияние атрибутов качества на архитектуру системы будет рассмотрено в следующей версии документа. Следует особо отметить "Надежность" и "Масштабируемость".