



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ  
КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)  
НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.03 Прикладная информатика

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**  
***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ***  
***РАБОТЕ***  
***НА ТЕМУ:***  
***Программная система моделирования***  
***искусственной жизни с использованием цифровых***  
***автоматов***

Студент ИУ6-85Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)      В.Д. Шульман  
\_\_\_\_\_  
(И.О. Фамилия)

Руководитель ВКР

\_\_\_\_\_  
(Подпись, дата)      О.Ю. Ерёмин  
\_\_\_\_\_  
(И.О. Фамилия)

Нормоконтролер

\_\_\_\_\_  
(Подпись, дата)      \_\_\_\_\_  
(И.О. Фамилия)

Москва, 2020

# Оглавление

<b>Аннотация.....</b>	<b>5</b>
<b>Реферат .....</b>	<b>6</b>
<b>Обозначения и сокращения .....</b>	<b>7</b>
<b>Введение .....</b>	<b>9</b>
<b>1 Исследовательская часть .....</b>	<b>11</b>
1.1 Генетические алгоритмы .....	11
1.1.1 Идея генетических алгоритмов.....	11
1.1.2 Сравнение генетических алгоритмов с детерминированными алгоритмами .....	11
1.1.3 Классификация генетических алгоритмов .....	12
1.1.4 Анализ предметной области.....	16
1.1.5 Основные информационные структуры предметной области .....	17
1.1.6 Основные процедуры предметной области .....	18
1.1.7 Необходимые информационные технологии .....	19
1.2 Цифровые автоматы .....	21
1.2.1 Классификация абстрактных автоматов .....	22
1.2.2 Базовая модель конечного автомата.....	25
1.3 Клеточные автоматы .....	26
1.3.1 Классификация клеточных автоматов .....	26
1.3.2 Эволюционирующий клеточный автомат .....	28
<b>2 Конструкторская часть .....</b>	<b>33</b>
2.1 Анализ требований к программной системе .....	33
2.1.1 Качества, характерные для высоконагруженной системы .....	33
2.1.2 Обеспечение работоспособности при высоких нагрузках.....	33
2.1.3 Обеспечение модульности и масштабируемости системы за счет использования микросервисного подхода .....	34
2.2 Обеспечение изолированной среды для компонентов программной системы при помощи виртуализации .....	37
2.2.1 Контейнер как метод виртуализации .....	37
2.2.2 Сравнение виртуальных машин и контейнеров.....	38
2.2.3 Анализ и выбор технологии контейнеризации для разрабатываемой системы.....	40
2.2.4 Docker .....	41
2.3 Проектирование инфраструктуры для разрабатываемой системы .....	42

2.3.1	Система оркестровки контейнерами Kubernetes.....	42
2.3.2	Архитектура кластера Kubernetes.....	44
2.3.3	Процесс развертывания компонентов программной системы в Kubernetes .....	45
2.3.4	Преимущества использования Kubernetes .....	46
2.4	Проектирование программной системы с использованием Kubernetes и Docker.....	47
2.4.1	Выбор базовых программных компонентов проектируемой системы .....	47
2.4.2	Простейший вариант программной архитектуры проектируемой системы .....	48
2.4.3	Масштабируемый архитектура проектируемой системы .....	49
2.4.4	Реализация проектируемой системы в кластере Kubernetes.....	50
2.5	Проектирование модели агента и генетического алгоритма .....	52
2.5.1	Построение модели агента на основе цифрового автомата.....	52
2.5.2	Код-геном.....	53
2.5.3	Параметры окружения и агента.....	56
2.5.4	Генетический алгоритм и метод отбора .....	57
2.6	Хранение данных.....	58
2.7	Выявление угроз в проектируемой системе .....	59
2.7.1	Нарушение доступности.....	59
2.7.2	Нарушение целостности рассматриваемой автоматизированной системы .....	60
2.7.3	Нарушение конфиденциальности информации .....	61
2.7.4	Резюме на выявленные угрозы .....	62
2.8	Управление доступом на основе ролей .....	62
<b>3</b>	<b>Технологическая часть .....</b>	<b>63</b>
3.1	Разработка генетического алгоритма .....	63
3.1.1	Реализация модели агента .....	65
3.1.2	Реализация кода-генома и обработки команд агента .....	67
3.1.3	Реализация пространства моделирования .....	67
3.1.4	Реализация сеанса моделирования .....	67
3.2	Создание хранилища данных .....	68
3.3	Обеспечение информационной безопасности .....	70
3.3.1	Построение модели нарушителя.....	70
3.3.2	Обеспечение защиты от нарушения доступности и целостности системы в единой точке доступа к системе .....	72

3.3.3	Обеспечение защиты в приложении frontend-сервера .....	72
3.3.4	Обеспечение защиты на backend-сервере .....	73
3.3.5	Обеспечение защиты данных в БД .....	73
<b>Заключение .....</b>		<b>75</b>
<b>Список использованных источников .....</b>		<b>76</b>
<b>Приложение А – Техническое задание .....</b>		<b>78</b>
<b>Приложение Б – Руководство пользователя .....</b>		<b>79</b>
<b>Приложение В – Фрагменты исходного кода системы .....</b>		<b>80</b>
<b>Приложение В – Копии листов графической части .....</b>		<b>81</b>

## **Аннотация**

В настоящей выпускной квалификационной работе бакалавра описан процесс разработки системы моделирования искусственной жизни с использованием цифровых автоматов/

Произведен анализ и классификация существующих подходов к использованию генетических алгоритмов. Были рассмотрены дискретные автоматы, как одно из возможных средств при реализации генетического алгоритма, что и было сделано в данной работе.

Разработаны технологии задания параметров, отправки данных на удаленный сервер, тестирование системы. Разработан интерфейс программного обеспечения.

## ***ABSTRACT***

*This bachelor's final qualifying paper describes the process of developing a system for modeling artificial life using digital automata.*

*The analysis and classification of existing approaches to the use of genetic algorithms is made. Discrete automata were considered, as one of the possible means for implementing a genetic algorithm, which was done in this paper.*

*Technologies for setting parameters, sending data to a remote server, and testing the system have been developed. The software interface has been developed.*

## Реферат

Расчётно-пояснительная записка с. 60, рис. 28, табл. 5, источников 15, приложений 4.

ГЕНЕТИЧЕСКИЙ АЛГОРИТМ, СЕЛЕКЦИОННЫЙ МЕТОД, ОТБОР, АГЕНТ, ХРАНИЛИЩЕ ГЕНОВ, КОНЕЧНЫЙ АВТОМАТ, КЛЕТОЧНЫЙ АВТОМАТ, ОКРЕСТНОСТЬ КЛЕТОЧНОГО АВТОМАТА, ЭВОЛЮЦИОНИРУЮЩИЙ КЛЕТОЧНЫЙ АВТОМАТ

Рассматриваемыми объектами в данной работе являются генетические алгоритмы и методы отбора при решении прикладных задач, где в основе лежит использование цифровых автоматов.

Целью работы является поиск и нахождения примеров эффективного применения цифровых автоматов в генетических алгоритмах, а также формулирование основных принципов при построении моделей с использованием цифровых автоматов. В работе рассматриваются способы представления цифрового автомата и также подходы к его модификации в процессе работы генетического алгоритма. Также в данном исследовании рассматриваются некоторые подходы к выработке метода отбора и выявления оптимальных параметров для него.

## Обозначения и сокращения

Генетический алгоритм – алгоритм, осуществляющий поиск решения задачи оптимизации или моделирования путем случайного комбинирования параметров и отбора лучших решений.

Селекция – процесс, в рамках которого производится отбор по определенным критериям наиболее оптимального решения.

Селекционный метод – алгоритм, согласно которому производится оценка решений и формирование нового поколения.

Агент – элементарная эволюционная единица, осуществляющая решение поставленной перед ней задачи.

Популяция – множество агентов.

Поколение – текущая популяция агентов, сформированная путем на основании предыдущих поколений, прошедших через селекционный отбор.

Жизненный цикл – период времени от появления агента в системе до его исключения из системы в результате работы селекционного алгоритма.

Генетический код – набор параметров и инструкций, которые выполняет агент в процессе своего жизненного цикла.

Мутация – случайное изменение генетического кода агента, приводящее к приближению или отдалению агента от оптимального решения.

Оптимальное решение – решение, которое по набору признаков является наиболее предпочтительным.

Эвристический алгоритм – алгоритм решения задачи, который не является гарантированно точным или оптимальным.

Детерминированный алгоритм – алгоритмический процесс, который выдает уникальный и предопределенный результат для заданных входных данных.

Цифровой автомат – математическая модель дискретного устройства, которое принимает и выдает сигналы, принимая различные состояния.

Абстрактный автомат – математическая абстракция, модель дискретного устройства, имеющая один вход, один выход, которое в каждый момент времени находится в одном из множества возможных состояний.

Конечный автомат – автомат, у которого количество внутренних состояний, которые он может принимать, ограничено.

Элементарный автомат – автомат, который описывается как автомат Мура, имеет двоичный алфавит, обладает двумя внутренними состояниями, обладает полной системой переходов и системой выходов.

Клеточный автомат – дискретная модель, изучаемая в математике и теории вычислимости, представляющая собой решетку ячеек, каждая из которых может принимать одно из доступных состояний согласно окрестности и правилу клеточного автомата.

ГА – генетический алгоритм.

ПГА – параллельный генетический алгоритм.

АА – абстрактный автомат.

КА – конечный автомат.

БА – бесконечный автомат.

ЭА – элементарный автомат.

УТК – указатель текущей команды.



## Введение

На данный момент достаточно широкое распространение получили методы решения задач, основанные на использовании генетических алгоритмов. Эти методы в основном используются для решения задач, решение для которых либо слишком сложно найти аналитически, составив алгоритм, либо они являются настолько абстрактными, что решение для них в виде детерминированного алгоритма вовсе не существует. Генетические алгоритмы позволяют решать такие задачи.

Генетические алгоритмы (ГА) являются частью более общей области – эволюционного моделирования. В отличие от других технологий оптимизации, ГА содержат популяцию пробных решений, которые конкурентно управляются с помощью определенных операторов. Генетическим алгоритмам присуще итеративное обучение популяции агентов.

Генетические алгоритмы относятся к классу эвристических алгоритмов поиска, т.е. они решают задачи, используя практические методы, которые не являются абсолютно точными и оптимальными, однако они достаточно точны для решений поставленной перед ними задачи при заданных условиях [1].

На волне популярности искусственного интеллекта генетические алгоритмы являются поводом для постоянных дискуссий и исследований. Несмотря на то, что ИИ представляет из себя целый комплекс различных систем, таких как системы распознавания, анализа, обучения, адаптации и т.п., генетические алгоритмы вполне могут решить часть этих задач.

Генетические алгоритмы актуальны на данный момент тем, что уже сейчас способны решать широкий спектр задач. Генетические алгоритмы могут использоваться для управления процессорами, балансируя нагрузку в многопроцессорной системе, они могут использоваться для решения задач по поиску наилучшего соотношения веса/прочности/размера/плотности, для решения задач по размещению объектов различных форм на определенной

площади (трассировка плат или нарезка ткани с наименьшими потерями, например) и т. д. и т. п.

Один из недавних коммерческих примеров: израильская компания Schema разработала программный продукт Channeling для оптимизации работы сотовой связи путем выбора оптимальной частоты, на которой будет вестись разговор. В основе этого программного продукта и используются генетические алгоритмы.

Генетические алгоритмы приобрели большую популярность на фоне новой волны подъема интереса к нейронным сетям. Сейчас довольно быстро развиваются технологии по распознаванию изображения и звука. Для обучения нейронных сетей также используется генетический алгоритм, где после каждого раунда нейронные сети сравниваются по своей эффективности, после чего часть из них отсеивается, а на основании оставшихся генерируется новое поколение нейронных сетей с измененными весами переходов.

Одно из наибольших достоинств генетических алгоритмов – это возможность распараллеливания вычислений при поиске решений. Т.к. агенты, которые пытаются решить задачу, чаще всего не должны взаимодействовать друг с другом, то обработку каждого из них можно вести параллельно. Раньше это не давало больших преимуществ из-за одноядерных процессоров, но сейчас с развитием многопроцессорных архитектур и асинхронных языков программирования поиск решения с использованием генетических алгоритмов может проходить в разы быстрее.

Основной целью НИР является нахождение и оценка оптимальных подходов к реализации генетических алгоритмов с использованием цифровых автоматов и определению круга задач, для которых использование такого вида генетических алгоритмов было бы оправдано и эффективно.

# **1 Исследовательская часть**

## **1.1 Генетические алгоритмы**

### **1.1.1 Идея генетических алгоритмов**

Идея генетических алгоритмов не нова. Еще в 60-ых годах прошлого века начались первые симуляции эволюционных процессов на тогдашних маломощных огромных ЭВМ. В 70-ых годах была разработана концепция эволюционного программирования с использованием конечных автоматов. Достаточно большую популярность генетические алгоритмы обрели после экспериментов Джона Холланда с так называемыми клеточными автоматами.

Однако из-за скромных вычислительных мощностей длительно время генетические алгоритмы оставались в основном в теоретической области и почти не использовались для решения прикладных задач. Лишь к концу 80-ых годов с ростом вычислительных мощностей стало внедряться оборудование, включавшее в себя генетические алгоритмы для решения поставленных перед ними задач. В 89-ом году компания Axcelis выпустила свой первый продукт для персональных компьютеров, в котором применялись генетические алгоритмы. Носил он символическое название “Evolver”.

Сейчас существует много вариантов реализации генетических алгоритмов, каждый из которых больше подходит для решения определенных задач. Например, нейронные сети хорошо решают задачу распознавания изображений, но решить задачу самообучающегося ИИ нейронные сети так и не смогли, несмотря на то, что прошло несколько витков их популярности.

### **1.1.2 Сравнение генетических алгоритмов с детерминированными алгоритмами**

Генетические алгоритмы применяются чаще для нахождения приближенного решения. Это обосновывается тем, что для сложной задачи чаще всего требуется найти не идеальное решение, а удовлетворяющее хотя бы имеющимся требованиям [4]. При этом достижение идеального «оптимального» решения отходит на второй план. Однако при этом другие

методы, ориентированные на нахождение оптимального решения, из-за их чрезвычайно высокой сложности становятся и вовсе нереализуемыми [6].

Основные отличия генетических алгоритмов от традиционных методов:

- 1) генетические алгоритмы работают с кодами, в которых представлен набор параметров, напрямую зависящих от аргументов целевой функции;
- 2) для поиска генетический алгоритм использует несколько точек поискового пространства одновременно, а не переходит от точки к точке, как это делается в традиционных методах;
- 3) генетические алгоритмы в процессе работы не используют никакую дополнительную информацию;
- 4) генетический алгоритм использует как вероятностные правила для порождения новых точек, так и детерминированные правила для перехода от одних точек к другим.

### **1.1.3 Классификация генетических алгоритмов**

Генетические алгоритмы являются частным случаем еще более обобщенной сферы – генетического моделирования (рис. 1), но генетические алгоритмы достаточно разнообразны и обладают собственной классификацией [3].

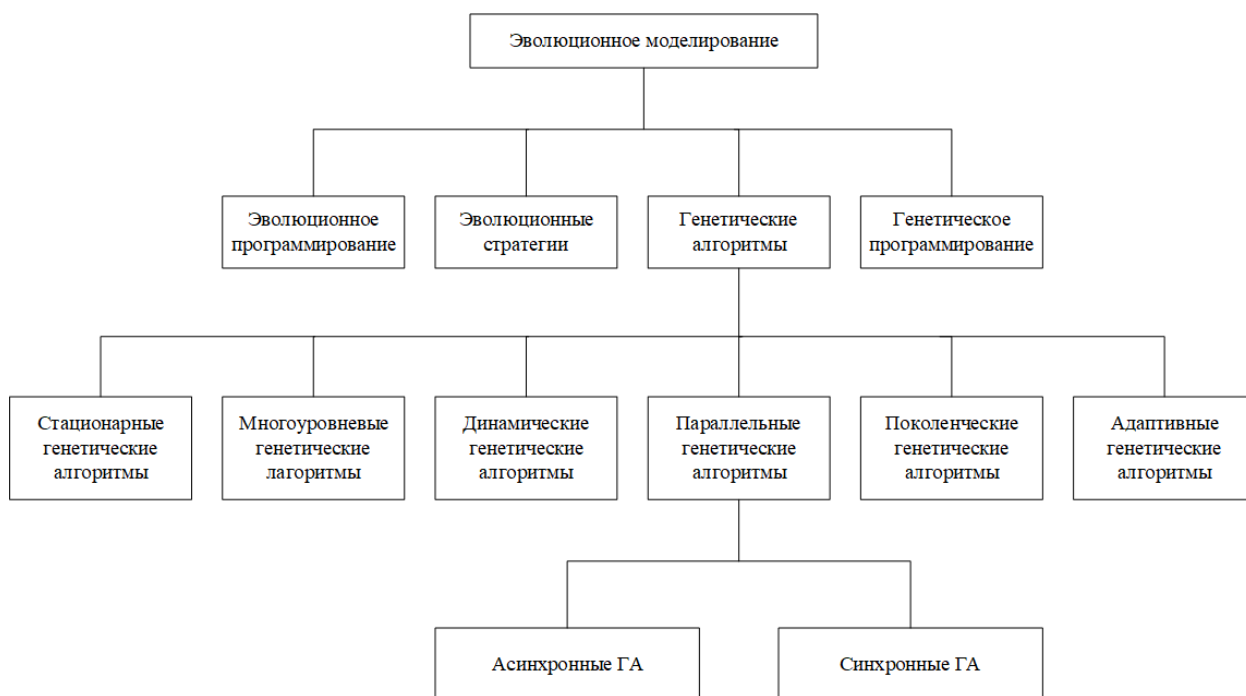


Рисунок 1 – Классификация алгоритмов эволюционного моделирования

ГА не обязательно относится только лишь к одной из возможных классификаций, т. к. не все из них являются взаимоисключающими. Генетический алгоритм может быть отнесен сразу к нескольким категориям, которые представлены на рисунке 1.

### 1.1.3.1 Стационарные генетические алгоритмы

Стационарные генетические алгоритмы подразумевают использование подхода, в котором популяция обновляется частями, а не вся сразу. Идея заключается в итеративном создании одного или двух потомков и добавления их непосредственно в популяцию, с предварительным исключением некоторых существующих агентов, чтобы освободить необходимое пространство.

Данный генетический алгоритм, по сравнению с поколенческим, использует примерно в два раза меньше памяти, т.к. всегда рассматривается только одна популяция. Также в этом алгоритме агенты-родители остаются в популяции на более длительное время, т. к. длительность их жизненного цикла обусловлена не «жесткими параметрами моделирования», а параметрами

самого агента, которые формируются под воздействием генетического алгоритма случайным образом.

Такой тип алгоритмов более характерен для реального мира, поколения не имеют между собой четких временных границ, а сменяемость поколений происходит не резко, «между раундами», а плавно, в процессе конкуренции.

#### **1.1.3.2 Динамические генетические алгоритмы.**

Противоположен по своим идеям стационарному генетическому алгоритму. Здесь подразумевается использования итеративного подхода, где после каждой итерации популяция проходит полное обновление, в рамках которого удаляются и добавляются агенты.

#### **1.1.3.3 Поколенческие генетические алгоритмы**

Поколенческие генетические алгоритмы подразумевают итеративное следование определенной последовательности действий:

- 1) создание первоначальной популяции;
- 2) вычисление функций приспособленности для агентов популяции (оценивание);
- 3) выборка агентов из текущей популяции (селекция);
- 4) мутации агентов;
- 5) вычисление функций приспособленности для всех агентов;
- 6) формирование нового поколения;
- 7) оценить результаты и, если они не удовлетворительны, то повторить, начиная с 3 пункта.

#### **1.1.3.4 Адаптивные генетические алгоритмы**

Основная идея адаптивного генетического алгоритма заключается в том, что такой алгоритм должен уметь изменять свои параметры в процессе работы. В качестве параметров здесь выступает размер популяции, вероятность мутации, количество удаляемых и добавляемых агентов, а также множество других параметров, которые зависят от специфики самой задачи. Целью создания адаптивного алгоритма является ускорение поиска решения или уменьшения затрат ресурсов при его поиске (например, памяти).

### **1.1.3.5 Многоуровневые генетические алгоритмы.**

Такие алгоритмы представляют собой многоуровневую систему, где каждый уровень генетического алгоритма оптимизирует работу вышестоящего генетического алгоритма и так по цепочке. Говоря более формально, многоуровневый генетический алгоритм – это система, в которой генетические алгоритмы нижнего уровня (оптимизируемые генетические алгоритмы) оптимизируют генетические алгоритмы верхнего уровня.

Этот принцип часто используется в нейронных сетях, где вершины нейронной сети разбиты по слоям, представляющим собой уровни генетического алгоритма. При этом возникает очень сложная задача оптимизации, где необходимо вычислить оптимальное количество слоев для наиболее быстрого поиска решения задачи, т.к. никакой универсальной формулы, дающей ответ на этот вопрос, не существует.

### **1.1.3.6 Параллельные генетические алгоритмы.**

Такие алгоритмы основаны на разбиении популяции на несколько отдельных подпопуляций, каждая из которых будет, независимо от других подпопуляций, обрабатываться генетическим алгоритмом. Одновременно с этим, разнообразные миграции агентов порождают обмен генетическим кодом среди популяций, которые, улучшают точность и эффективность алгоритма.

Выделяют три типа параллельных генетическим алгоритмов:

- глобальные однопопуляционные ПГА (master-slave);
- однопопуляционные ПГА;
- многопопуляционные ПГА.

Модель «master-slave» характеризуется тем, что в алгоритмах такого типа селекция принимает во внимание целую популяцию, в отличии от двух других моделей.

В алгоритмах второго класса существует главная популяция, но оценка целевой функции распределена среди нескольких процессоров. Master хранит популяцию, выполняет операции ГА и распределяет агентов между

подчиненными. Они же лишь оценивают агентов. Однопопуляционные ГА пригодны для массовых параллельных компьютеров и состоят из одной популяции.

Третий класс - многопопуляционные ГА, более сложная модель, так как она состоит из нескольких подпопуляций, которые периодически, по установленным правилам, обмениваются агентами. Такой обмен агентами называется миграцией и управляется несколькими параметрами. Многопопуляционные ГА имеют сходство с «островной моделью» в популяционной генетике, которая рассматривает относительно изолированные общины; поэтому параллельные ГА в некоторых случаях называют «островными» параллельными ГА.

#### 1.1.4 Анализ предметной области

Предметная область информационной системы, реализующая генетические алгоритмы, включает в себя ряд объектов, данных и функций, которые её описывают. Они представлены на рис. 2.

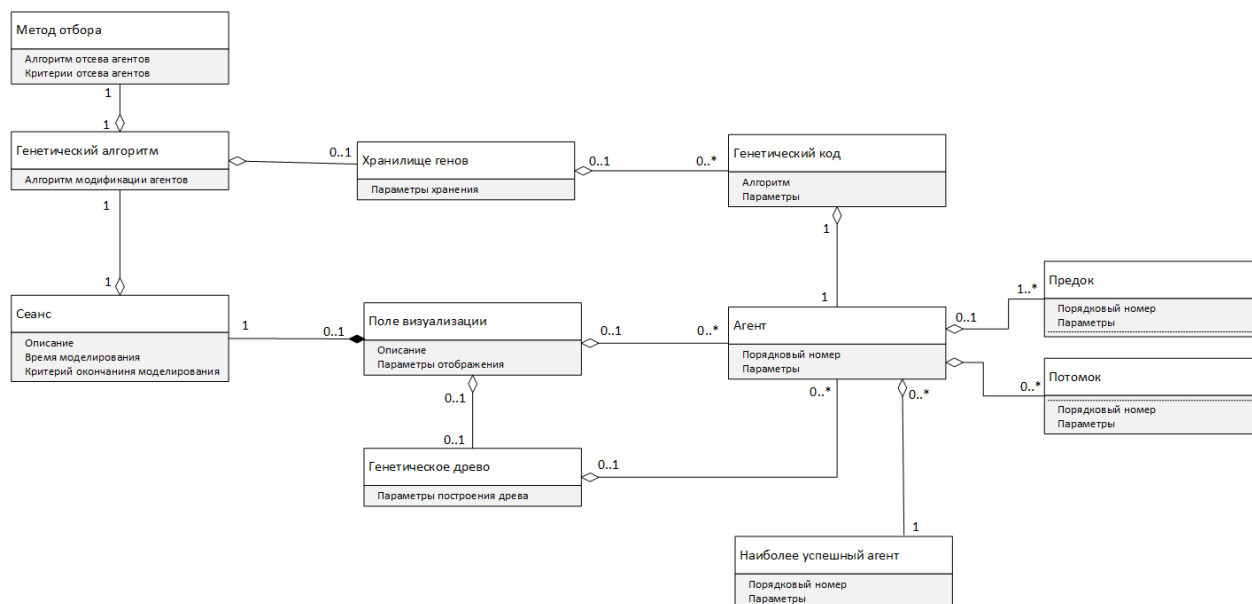


Рисунок 2 – Концептуальная схема предметной области

Объекты представляют собой действующие лица, между которыми идет обмен информацией и которые осуществляют некоторые функции друг над другом.



#### **1.1.4.1 Агент**

Агент представляет собой элементарную рабочую единицу ИС. Перед агентом ставятся задачи, для которых в системе идет поиск решения, как можно более приближенного к оптимальному. Существует множество агентов, параллельно решающих поставленную перед ними задачу. В начале процесса нахождения оптимального решения задачи все агенты находятся в схожих условиях и имеют идентичную конфигурацию. В течение некоторого отрезка времени в модели порождаются новые агенты с несколько отличной конфигурацией согласно генетическому алгоритму. Периодически с помощью метода отбора часть агентов удаляется из процесса поиска оптимального решения.

#### **1.1.4.2 Хранилище генов**

Хранилище генов представляет из себя банк с данными. В этих данных содержится информация о конфигурациях агентов. Хранилище генов содержит в себе информацию как о тех агентах, которые участвуют в текущем сеансе, так и данные о конфигурации агентов из хранилища генов используются для формирования конфигурации новых агентов в процессе работы генетического алгоритма.

#### **1.1.5 Основные информационные структуры предметной области**

Информационные структуры представляют из себя совокупность данных, которыми обмениваются объекты предметной области. Также данные используются функциями (алгоритмами) для получения результата, который может представлять из себя набор еще каких-то данных.

##### **1.1.5.1 Генетический код**

Включает в себя набор параметров и инструкций, который использует агент в процессе своего жизненного цикла. Генетический код агента также можно определить, как его конфигурацию. Генетический код изменяется случайным образом в процессе работы генетического алгоритма. Генетический код агента, который наиболее эффективно решает

поставленную перед ним задачу, является тем самым оптимальным решением, поиск которого производится в системе.

#### **1.1.5.2 Генетическое древо**

Представляет из себя совокупность информации о родителях и потомках каждого агента, когда-либо появившегося в процессе сеанса поиска решения. Генетическое древо показывает, какие ветви развития агентов имели какую численность, и какая последовательность изменений привела к текущим генетическим кодам агентов.

#### **1.1.5.3 Поле визуализации**

Поле визуализации показывает текущее состояние сеанса работы генетического алгоритма.

Оно может в себя включать различные графические элементы:

- графики;
- диаграммы;
- графы состояний и переходов;
- поля пространств, где размещаются и перемещаются агенты;
- визуализации наполнения банков данных;
- генетическое древо.

Информация в поле призвана дать возможность объективно оценить текущее состояние сеанса и определить, был ли достигнут требуемый результат работы генетического алгоритма.

#### **1.1.6 Основные процедуры предметной области**

Процедуры (алгоритмы), реализуемые в системе и использующие генетические алгоритмы, необходимы для обработки имеющихся данных и формирования новых.

##### **1.1.6.1 Генетический алгоритм**

Наиболее важная часть ИС, определяющая всю логику ее работы и множество задач, которые эта система может решать.

Генетический алгоритм определяет, каким образом будет происходить наследование генетического кода от агентов одного поколения к агентам последующего, а также какие случайные изменения будут происходить в генетическом коде новых агентов и с какой вероятностью.

#### **1.1.6.2 Метод отбора**

Это алгоритм, который осуществляет отсев агентов в процессе моделирования. Отсеиваются агенты, которые решили поставленную перед ними задачу менее эффективно, чем позволяет пороговый критерий.

#### **1.1.6.3 Сеанс**

Сеанс представляет из себя весь временной отрезок, в рамках которого агенты решают представленную перед ними задачу. Сеанс в зависимости от условий и специфики задач, стоящих перед ними, может проходить в различных формах:

- агенты решают поставленную задачу независимо, никак не контактируя и не взаимодействуя друг с другом;
- агенты решают поставленную задачу в одной среде, весь сеанс поделен на периоды времени, представляющие из себя циклы, после каждого из которых происходит удаление и добавление агентов в зависимости от эффективности решения каждого;
- агенты решают задачу одновременно в одной среде соревнуясь друг с другом. При таком виде сеанса отбор осуществляется естественным образом в ходе конкуренции между агентами. При этом наиболее продвинутые агенты порождают других агентов, согласно генетическому алгоритму, поддерживая тем самым их общее количество. Этот тип сеанса больше всего напоминает естественные процессы, проходящие в живой природе.

#### **1.1.7 Необходимые информационные технологии**

Для успешного построения системы, которая будет обрабатывать данные с использованием генетического алгоритма, необходимо использование различных информационных технологий.

### **1.1.7.1 Централизованная обработка информации**

Централизованная обработка подразумевает, что работа системы будет осуществляться на удаленном высокопроизводительном сервере.

Такой подход к обработке эффективен, если решаемые задачи требуют затрат больших вычислительных ресурсов или специализированного программного обеспечения. В случае с генетическими алгоритмами использование такого подхода будет преследовать цель нахождения оптимального решения в максимально сжатые сроки.

Ожидаемый эффект использования технологии – быстрое моделирование работы генетического алгоритма.

### **1.1.7.2 Децентрализованная обработка информации**

Децентрализованная обработка информации подразумевает использование распределенной системы, где в роли вычислительных мощностей выступают компьютеры абонентов.

Этот вариант подходит, если скорость работы генетического алгоритма не столь важна, сколько возможность проводить параллельно множество таких процессов моделирования. При этом это дает каждому абоненту более широкие возможности по настройке параметров моделирования.

Ожидаемый эффект использования технологии – возможность вести множество моделирований работы генетического алгоритма, не ограничивая себя вычислительными мощностями центрального сервера системы.

### **1.1.7.3 Информационная технология поддержки принятия решений**

Эта технология организует взаимодействие пользователя и вычислительной системы, где в результате обработки входных данных от пользователя, выдается рекомендация.

Технология поддержки принятия решения ориентирована на решение плохо структурированных задач. Использование генетического алгоритма позволяет решать такие задачи, где не найдено аналитическое решение, но при этом можно составить математическую модель, которая описывала бы предметную область и критерии оптимальности решения.

Ожидаемый эффект использования технологии – возможность принимать от пользователя набор данных, описывающий задачу, и находить максимально приближенное к оптимальному решение этой задачи с использованием генетического алгоритма.

Особенность применения – необходимо обеспечивать дополнительный анализ информации от пользователя для составления математической модели решаемой задачи.

## 1.2 Цифровые автоматы

Понятие абстрактного автомата [АА] позволяет рассматривать дискретные объекты с точки зрения алгоритмов их функционирования, то есть реализуемых последовательностей действий по преобразованию дискретной информации [2].

Абстрактным автоматом называют модель, которая описывается кортежем, состоящим из 5 элементов:

$$A = (X, Y, S, f_y, f_s)$$

Здесь первые три компонента – это непустые множества:

$X$  – множество входных сигналов,

$Y$  – множество выходных сигналов,

$S$  – множество состояний.

Остальные два компонента кортежа являются характеристическими функциями:

$f_y$  – функция выходов,

$f_s$  – функция переходов АА из одного состояния в другое

Если множества  $X$ ,  $Y$ ,  $S$  – конечные, то такой АА называется конечным автоматом [КА]. В том случае, если бы хотя бы одно из множеств автомата было бесконечным, то и сам АА являлся бы бесконечным.

В данной исследовательской работе основным объектом для анализа являются именно КА, который, в отличие от БА, программно-реализуемым и

может быть полностью представлен с помощью таблицы или графа переходов, что дает возможность их практического применения.

### 1.2.1 Классификация абстрактных автоматов

Для классификации автомата рассматривается множество признаков, например, определенность функции переходов и функции выходов, однозначность заданных функций, устойчивость состояний. Все три признака перечислены и определены в таблице 1.

Классификация абстрактных автоматов может быть представлена в виде схемы на рисунке 3.

Таблица 1

Признак	Определение
Определенность характеристических функций	В автоматах полностью определенных областью определения функций $f_s$ и $f_y$ является множество всех пар $(s_i, x_k) \in S \times X$ , где $s_i \in S$ , $x_k \in X$ . В автоматах частично определены либо обе характеристические функции, либо одна из них имеют областью определения строгое подмножество декартова произведения $S \times X$ . характеристические функции определены не для всех пар $(s_i, x_k)$ .
Однозначность функции переходов	В детерминированных автоматах выполняется условие однозначности переходов: если АА находится в некотором состоянии $s_i \in S$ , то под воздействием произвольного входного сигнала $x_k \in X$ автомат может перейти в одно и только одно состояние $s_j \in S$ . В автоматах вероятностных при воздействии одного и того же входного сигнала возможны переходы из состояния $s_i$ в различные состояния из множества $S$ с заданной вероятностью.
Устойчивость состояний	В устойчивых автоматах выполняется условие устойчивости: если автомат под воздействием входного сигнала $x_k \in X$ оказался в состоянии $s_i \in S$ , то выход из него и переход в иное состояние возможен только при поступлении на вход автомата другого сигнала $x_z \in X$ , $x_z \neq x_k$ .

Для практического применения чаще всего используются те автоматы, которые по перечисленным выше признакам являются полностью определенными, детерминированными и устойчивыми конечными автоматами.



Рисунок 3 – Классификация абстрактных автоматов

Также можно классифицировать автоматы по виду их характеристической функции. Если аргументами характеристической функции являются только текущее значение входного сигнала и текущее состояние, то такой автомат является автоматом 1 рода или автоматом Мили. Если выходные сигналы автомата зависят исключительно от текущего состояния автомата, то такой автомат принято считать автоматом 2 рода или автоматом Мура.

Пример графов переходов для автомата Мура и автомата Мили представлены на рисунках 4 и 5 соответственно/

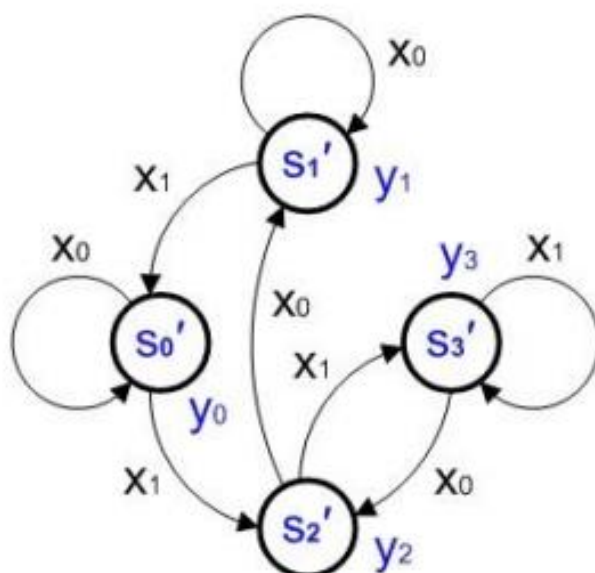


Рисунок 4 – Граф переходов автомата Мура

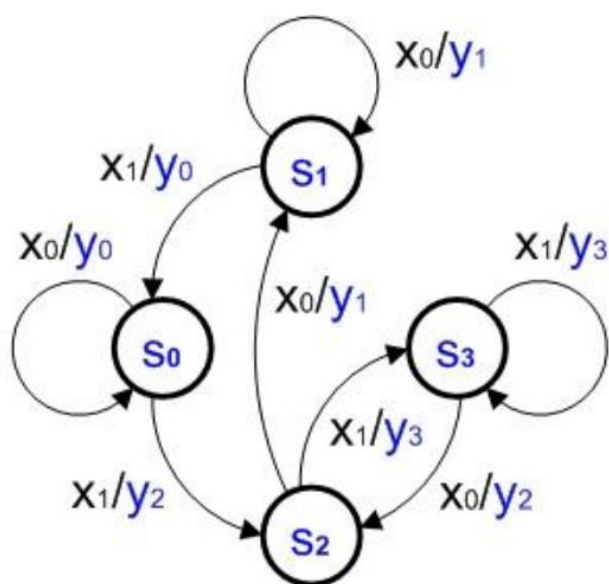


Рисунок 5 – Граф переходов автомата Мили

Следует при этом отметить, что автомат Мили запаздывает на один дискретный момент времени по входному сигналу по отношению к автомату Мура

Если в автомате присутствуют свойства как автомата Мура, так и автомата Мили, то такой автомат можно считать смешанным.



### 1.2.2 Базовая модель конечного автомата

Конечный автомат, в описание которого входят таким образом определенные множества, называют  $(n, p, q)$ -автоматом, а самим множествам присваивают наименование векторов, например, вектор входных сигналов, вектор состояний.

Все автоматы, и в том числе конечные, функционируют в дискретном исчислении времени. Моменты времени образуют ряд целых неотрицательных чисел:  $t = 0, 1, 2, 3, \dots$  В каждый дискретный момент времени КА находится в одном и только одном состоянии  $S_i$ , воспринимает одно значение вектора  $X$  и выдает на выходе одно значение вектора  $Y$ .

Принято считать, что в момент времени  $t = 0$  автомат находится в начальном состоянии  $S_0$ , которое можно включить в кортеж отдельным, шестым компонентом:  $A = (X, Y, S, f_y, f_s, S_0)$ .

Автомат с выделенным начальным состоянием называют инициальным.

Общую схему автомата можно представить в виде «черного ящика», осуществляющего преобразование вектора входных сигналов в вектор выходных.

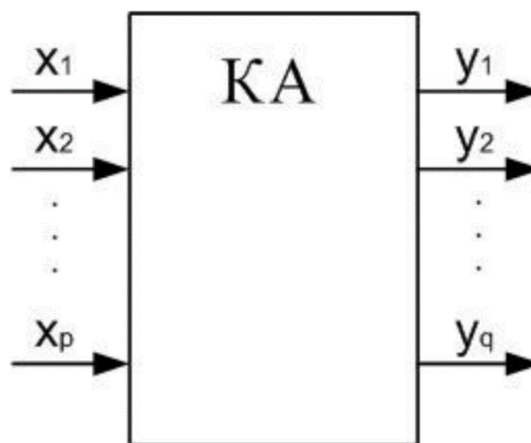


Рисунок 6 – Общая схема конечного автомата

Исходя из этого, можно записать следующее уравнение:  $Y = f_y(X, t)$ .

Фактор времени в приведенном уравнении учитывается введением вектора состояний  $S$ , как своего рода «памяти о прошлом». Действительно, на

один и тот же набор входных сигналов (значений компонентов вектора  $X$ ) автомат будет выдавать разные выходные сигналы (значения компонентов вектора  $Y$ ) в зависимости от состояния, в котором он находится в данный момент времени. Текущее состояние, в свою очередь, определяется алгоритмом функционирования автомата.

### 1.3 Клеточные автоматы

#### 1.3.1 Классификация клеточных автоматов

Клеточный автомат – это дискретная модель, которая представляет из себя сетку произвольной размерности. Каждая клетка в каждый момент времени может принимать одно из конечного множества состояний, при этом существует правило, по которому осуществляется переход клеток из одного состояния в другое.

Клеточные автоматы могут быть классифицированы критериями, представленными в таблице 2.

*Таблица 2*

Критерий	Возможные варианты
Размерность решетки	Одномерная, двумерная, трехмерная и т.д.
Количество возможных состояний	Бинарные, троичные и т.д.
Определение окрестности клетки	Окрестность Фон-Неймана, окрестность Мура и др.
Синхронизация	Синхронные и асинхронные клеточные автоматы
Тип поведения	4 класса

Клеточные автоматы можно разделить на 4 класса по типу поведения:

1) все клетки быстро принимают одинаковое состояние, после чего автомат стабилизируется;

2) состояние всех клеток быстро стабилизируется, либо возникают периодические колебания состояний клеток;

3) автомат порождает хаотические, непериодические структуры. Небольшие изменения исходного состояния влекут за собой значительные изменения в будущем;

4) автомат порождает сложные, взаимодействующие между собой структуры, способные выживать длительное время. Однако при этом автомату не удастся достичь стабильного состояния.

Самой простой, так называемый простейший клеточный автомат - это одномерный бинарный клеточный автомат, где состояния клетки в каждый момент времени зависят только от ее собственного состояния и состояний смежных с ней клеток в предыдущий момент времени [7].

Простейших клеточных автоматов существует всего 256, и поведение некоторых из них дублирует другие. Но, несмотря на это, широко известный в узких кругах Стивен Вольфрам посвятил годы жизни их изучению [5].

Вариантов простейших автоматов всего 256. Каждый из вариантов таким автоматов принято называть по порядковому номеру, или же «Правило N». Возьмем для примера наиболее интересное из них, правило 110.

Двоичный код десятичного числа 110 представляет собой последовательность бит 01101110. Данная последовательность бит формирует функцию переходов клетки.

*Таблица 3*

111	110	101	100	011	010	001	000
0	1	1	0	1	1	1	0

В зависимости от состояний соседа слева, самой клетки, соседа справа (первая строка таблицы) на следующем шаге клетка примет одно из состояний, указанных во второй строке таблицы 3.

Графическая иллюстрация представлена на рисунке 7.

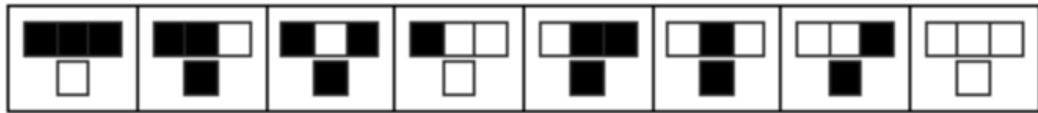


Рисунок 7 – графическое представление правила 110

Клеточные автоматы, в отличие от классических конечных автоматов, являются Тьюринг-полными. Это делает возможным, посредством клеточного автомата реализовывать любую вычислимую функцию, т.е. реализовывать любой возможный алгоритм.

Стоит заметить, что каждая клетка клеточного автомата представляет простой конечный автомат. В простом случае, когда имеется двумерное поле и каждая клетка способна принимать всего два состояния, эта самая клетка реализует автомат, который способен принимать два возможных состояния под воздействием входных сигналов. В данном случае входными сигналами будут являться окрестность клетки, т.е. текущие состояния клеток, входящих в эту окрестность. При этом текущее состояние клетки-автомата будет являться входным сигналом для других клеток.

### **1.3.2 Эволюционирующий клеточный автомат**

#### **1.3.2.1 Базовые концепции эволюционирующего клеточного автомата**

Клеточный автомат весьма мощное средство моделирования. Он позволяет строить модели различных динамических систем. Даже простейшие одномерные клеточные автоматы являются Тьюринг-полными и способны генерировать сложные непериодические структуры.

Однако все клетки такого автомата равны между собой и работает каждая из них по заранее определенному правилу. Соединение идеи клеточного автомата и генетического алгоритма позволит избежать статичности правила и позволит его видоизменять при каждой итерации алгоритма [8].

В этом случае правило автомата фактически превращается в генетический код, а сам клеточный автомат становится агентом.

### 1.3.2.2 Генетический код эволюционирующего клеточного автомата в случае использования одномерного клеточного автомата

В случае простейшего генетического автомата, где правило формируется по состояниям двух смежных клеток и состоянию самой клетки, существует 8 возможных комбинаций, из которых можно породить 256 правил (рис. 8). Как раз переход в одно из двух возможных состояний для каждой из 8 комбинаций будет определять один из соответствующих им 8 генов.

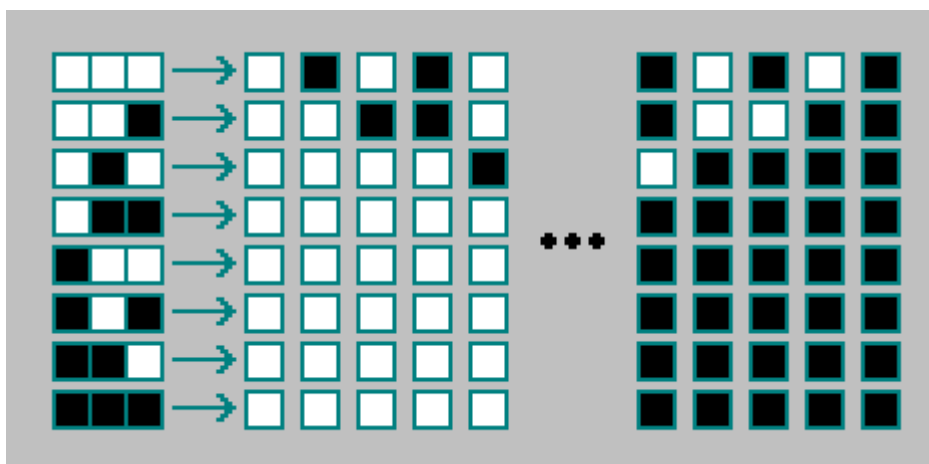


Рисунок 8 – Возможные комбинации состояний одномерного клеточного автомата

Для решения задач моделирования чаще приходится использовать моделирование в двумерном или трехмерном пространстве. Практических задач для одномерного пространства значительно меньше чем, например, задач на плоскости или связанных с объемным пространством. Поэтому стоит рассмотреть работу генетического алгоритма с использованием двумерного клеточного автомата для большей наглядности и практичности.

У двумерного клеточного автомата для каждой клетки, если использовать окрестность Мура, существует 8 соседних клеток. Существует порядка  $2^9 = 512$  возможных комбинаций состояний клетки и её соседей, из чего следует, что поведение данного клеточного автомата определяет комбинация из 512 генов. Комбинация этих генов дает  $2^{512}$  возможных генетических конфигураций клеточного автомата (рис 9), что является

огромным множеством всевозможных правил, согласно которым может работать данный клеточный автомат.

Рассмотренные клеточные автоматы являлись автоматами первого порядка, т. е. их последующее состояние зависело только от текущего состояния. Автоматы также могут быть и высших порядков. В таком случае следующее состояние клетки такого автомата будет зависеть не только от текущего состояния клетки и клеток в ее окрестности, но и от её состояния на предыдущих шагах

Такая большая комбинация генов дает возможность генерировать большое множество клеточных автоматов с различными правилами, но, исходя из идеи генетических алгоритмов, необходимо среди этого большого множества клеточных автоматов отбирать те, которые лучше решают поставленную перед ними задачу.

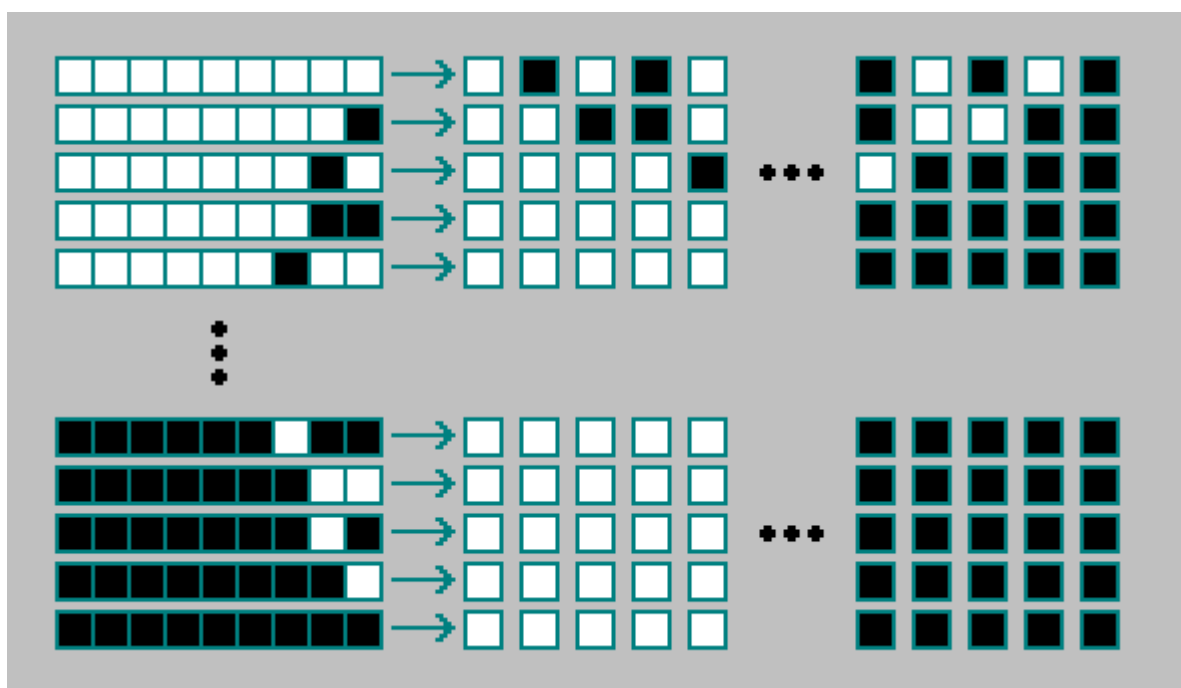


Рисунок 9 – Возможные комбинации состояний двумерного клеточного автомата

### 1.3.2.3 Отбор и селекция клеточных автоматов

Для оценки, насколько близко эволюционирующий клеточный автомат приблизился к цели в ходе работы генетического алгоритма, необходимо

сформировать определенные критерии, по которым будет происходить оценивание.

Критерии оценивания можно задать любые, в зависимости от поставленных задач. Для двумерного клеточного автомата первого порядка будет достаточно наглядно в качестве примера взять задачу по образованию заданного рисунка или узора.

В этом случае критерием оценивания для клеточных автоматов (агентов) будет выступать количество рисунков и их точность (процент ошибок). Поиск рисунков в этом случае будет производиться относительно каждой клетки в ее окрестности. Стоит отметить, что такая окрестность не должна совпадать с окрестностью, по которой сформировано правило в клеточном автомате. Более того, она может быть произвольного размера и формы (рис. 10).

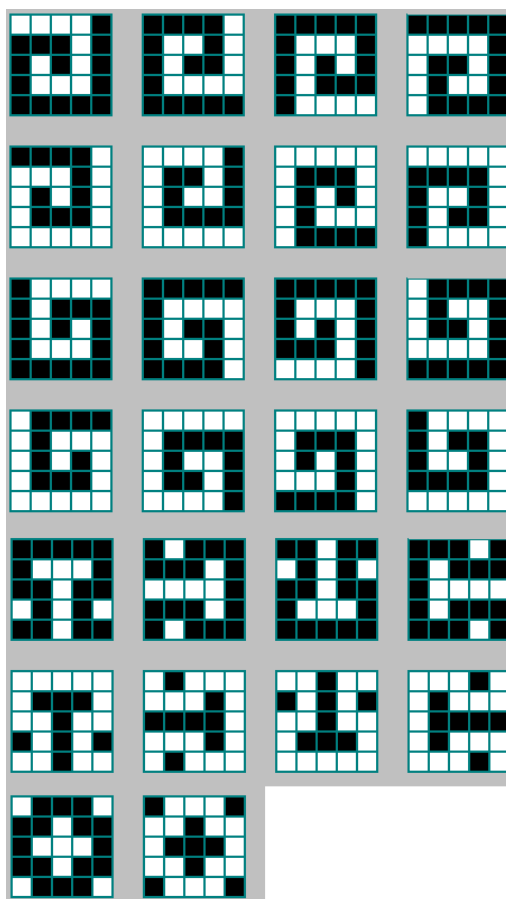


Рисунок 10 – Возможные критерии отбора для клеточного автомата с прямоугольной окрестностью 5x5

Согласно сформулированным критериям должен производиться отбор и селекция клеточных автоматов, участвующих в описке решения.

#### **1.3.2.4 Формирования генетического кода клеточного автомата в ходе селекции**

Для формирования генетического кода нового клеточного автомата можно использовать генетический код случайного автомата из предыдущего поколения со случайной мутацией одного из гена. Вероятность мутации определяется для каждой задачи индивидуально, при этом вычислить ее аналитически, как правило, не представляется возможным.

Можно использовать для формирования генетического кода нового автомата код сразу нескольких автоматов из предыдущего поколения (рис. 11). При этом автоматы из скрещивания выбираются случайным образом. В случайном гене полученного генетического кода с некоторой вероятностью также происходит мутация.



Рисунок 11 – Формирование генетического кода нового клеточного автомата из генетических кодов двух клеточных автоматов

В процессе отбора, селекции, мутаций и оценки полученных результатов происходит эволюционный процесс, в ходе которого происходит постепенное приближение к оптимальному решению поставленной перед генетическим алгоритмом задачи.



## **2 Конструкторская часть**

### **2.1 Анализ требований к программной системе**

Для проектируемой системы необходимо определить перечень качеств и свойств, которым она должна соответствовать. Это позволяет на ранних этапах проектирования более детально определить архитектуру программного обеспечения, а также, исходя из выявленных особенностей, подобрать более оптимальный набор технологий и языков программирования.

#### **2.1.1 Качества, характерные для высоконагруженной системы**

Т.к. будущая программная система представляет из себя веб-приложение, состоящее из нескольких программных модулей, доступное для всех пользователей интернета (что потенциально может привести к большой нагрузке как на систему в целом, так и на отдельные ее компоненты в случае высокой популярности данного приложения), то можно ее рассматривать с точки зрения высоконагруженной системы. С точки зрения высоконагруженных приложений для разрабатываемой системы будут применены перечни качеств и характеристик, свойственных для подобных систем.

Высоконагруженная система может обладать следующими качествами:

- обслуживает большое количество клиентов по отношению к среднему показателю на рынке (например, Facebook, Amazon, Youtube, ВКонтакте или почта Mail.ru);
- система является распределенной;
- постоянно растущая по количеству клиентов система;
- развивающаяся система;
- система, обладающая большим количеством ресурсов и растущей потребностью в них;

#### **2.1.2 Обеспечение работоспособности при высоких нагрузках**

Ключевым элементом в такой разработке является правильное проектирование архитектуры приложения, т. к. повышение нагрузки на

разрабатываемую систему не должно вызывать необходимость в изменении логики работы компонентов, добавлении новых, удалении старых.

Обеспечение горизонтальной масштабируемости компонентов системы на всех уровнях и предварительное выявление и ликвидация узких мест – наиболее эффективный шаг для обеспечения эффективного роста и легкой модернизации разрабатываемой системы в отдаленном будущем, когда рост нагрузки приведет к дефициту ресурсов при текущей конфигурации оборудования.

На данный момент для реализации больших систем с большим количеством различных компонентов (сейчас почти любая высоконагруженная система обладает большими размерами и большим разнообразием используемых компонентов) используются технологии виртуализации.

Независимо от того, сколько отдельных компонентов разрабатывается и развертывается, одна из самых больших проблем, с которой всегда приходится сталкиваться разработчикам и системным администраторам – это различия в окружениях, в которых они выполняют свои приложения. Существует огромная разница между окружением, в котором проходила разработка, и рабочим окружением. Различия даже существуют между отдельными физическими машинами при эквивалентном наборе программного обеспечения. Еще одним неизбежным фактом является то, что окружение одной рабочей машины будет меняться с течением времени.

Виртуализация позволяет абстрагироваться от конфигурации реальных физических машин и запускать компоненты системы в отдельных виртуальных машинах, тем самым делая их выполнение независимым от ОС, установленной на физическом сервере.

### **2.1.3 Обеспечение модульности и масштабируемости системы за счет использования микросервисного подхода**

Монолитные приложения состоят из компонентов, которые тесно связаны друг с другом и должны разрабатываться, развертываться и

управляться как одна сущность, поскольку все они выполняются как один процесс ОС. Изменения в одной части приложения требуют новой выкладки всего приложения, и со временем отсутствие жестких границ между частями приводит к увеличению сложности и последующему ухудшению качества всей системы из-за неограниченного роста взаимосвязей между этими частями.

Для запуска монолитного приложения обычно требуется небольшое количество мощных серверов, которые могут предоставить достаточно ресурсов для запуска приложения.

Для того чтобы справиться с растущей нагрузкой на систему, нужно либо масштабировать серверы вертикально (так называемое масштабирование вверх), добавляя больше процессоров, оперативной памяти и других серверных компонентов, либо масштабировать всю систему по горизонтали, настраивая дополнительные серверы и запуская несколько копий (или реплик) приложения (масштабирование вширь). Хотя масштабирование вверх обычно не требует каких-либо изменений в приложении, оно относительно быстро становится дорогостоящим и на практике всегда имеет верхний предел.

Масштабирование вширь, с другой стороны, является относительно дешевым аппаратно, но может потребовать больших изменений в программном коде приложения и не всегда возможно – некоторые части приложения с большим трудом поддаются горизонтальному масштабированию или почти невозможны для него (например, реляционные базы данных). Если какая-либо часть монолитного приложения не масштабируется, то все приложение становится немасштабируемым, если только каким-то образом этот монолит не разделить.

Эти и другие проблемы заставили начать разбиение сложных монолитных приложений на небольшие независимые развертывания компонентов, называемых микросервисами. Каждый микросервис выполняется как независимый процесс (рис. 12) и взаимодействует с другими микросервисами через простые, четко определенные интерфейсы (API).

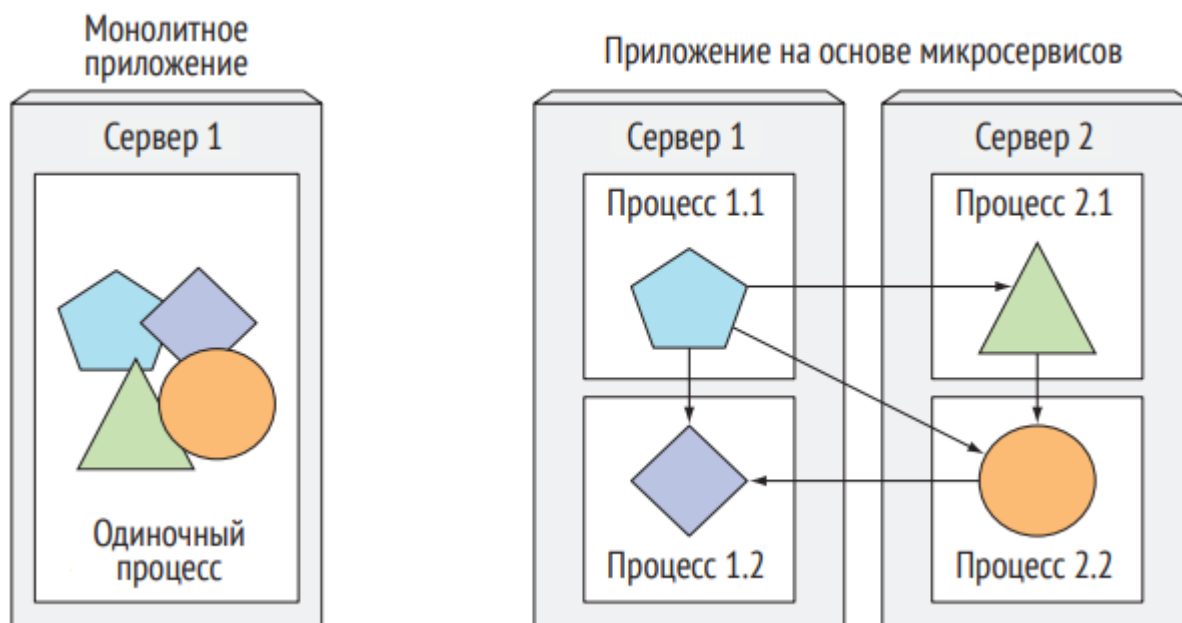


Рисунок 12 – Компоненты программной системы внутри монолитного приложения и независимых микросервисов

Микросервисы взаимодействуют через синхронные протоколы, такие как HTTP, используя которые, они обычно предоставляют RESTful, или через асинхронные протоколы, такие как AMQP (Advanced Message Queuing Protocol, расширенный протокол организации очереди сообщений). Эти протоколы просты, хорошо понятны большинству разработчиков и не привязаны к какому-либо конкретному языку программирования. Каждый микросервис может быть написан на языке, который наиболее целесообразен для реализации конкретных микросервисов.

Поскольку каждый микросервис представляет собой автономный процесс с относительно статическим внешним API, существует возможность разрабатывать и развертывать каждый микросервис отдельно. Изменение одного из них не требует изменений или повторного развертывания какого-либо другого сервиса, при условии, что API не изменяется или изменяется только обратно совместимым образом.

## **2.2 Обеспечение изолированной среды для компонентов программной системы при помощи виртуализации**

### **2.2.1 Контейнер как метод виртуализации**

Когда приложение состоит лишь из небольшого количества крупных компонентов, вполне допустимо предоставить каждому компоненту выделенную виртуальную машину и изолировать их среды, предоставив каждому из них собственный экземпляр операционной системы.

Однако, когда эти компоненты начинают уменьшаться в объеме и их количество начинает расти, становится невозможным предоставлять каждому из них свою собственную виртуальную машину.

Поскольку каждая виртуальная машина обычно должна настраиваться и управляться индивидуально, увеличение количества виртуальных машин также приводит к трате человеческих ресурсов, поскольку они значительно увеличивают рабочую нагрузку на системных администраторов.

Вместо того чтобы использовать виртуальные машины для изоляции сред каждого микросервиса (или программных процессов в целом), можно использовать контейнерные технологии Linux. Данные технологии позволяют запускать несколько сервисов на одной хост-машине, не только обеспечивая доступ к разным средам, но и изолируя их друг от друга, подобно виртуальным машинам, но с гораздо меньшими затратами.

Контейнеризация — метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного.

Эти экземпляры (обычно называемые контейнерами) с точки зрения пользователя полностью идентичны отдельному экземпляру операционной системы.

Ядро обеспечивает полную изолированность контейнеров, поэтому программы из разных контейнеров не могут воздействовать друг на друга.

При контейнеризации отсутствуют дополнительные ресурсные накладные расходы на эмуляцию виртуального оборудования и запуск полноценного экземпляра операционной системы, характерные при аппаратной виртуализации.

### 2.2.2 Сравнение виртуальных машин и контейнеров

По сравнению с виртуальными машинами, контейнеры гораздо облегченнее, что позволяет запускать большее количество программных компонентов на одном и том же оборудовании, главным образом потому, что каждая виртуальная машина должна запускать свой собственный набор системных процессов, который требует еще вычислительных ресурсов в дополнение к тем, которые потребляются собственным процессом компонента.

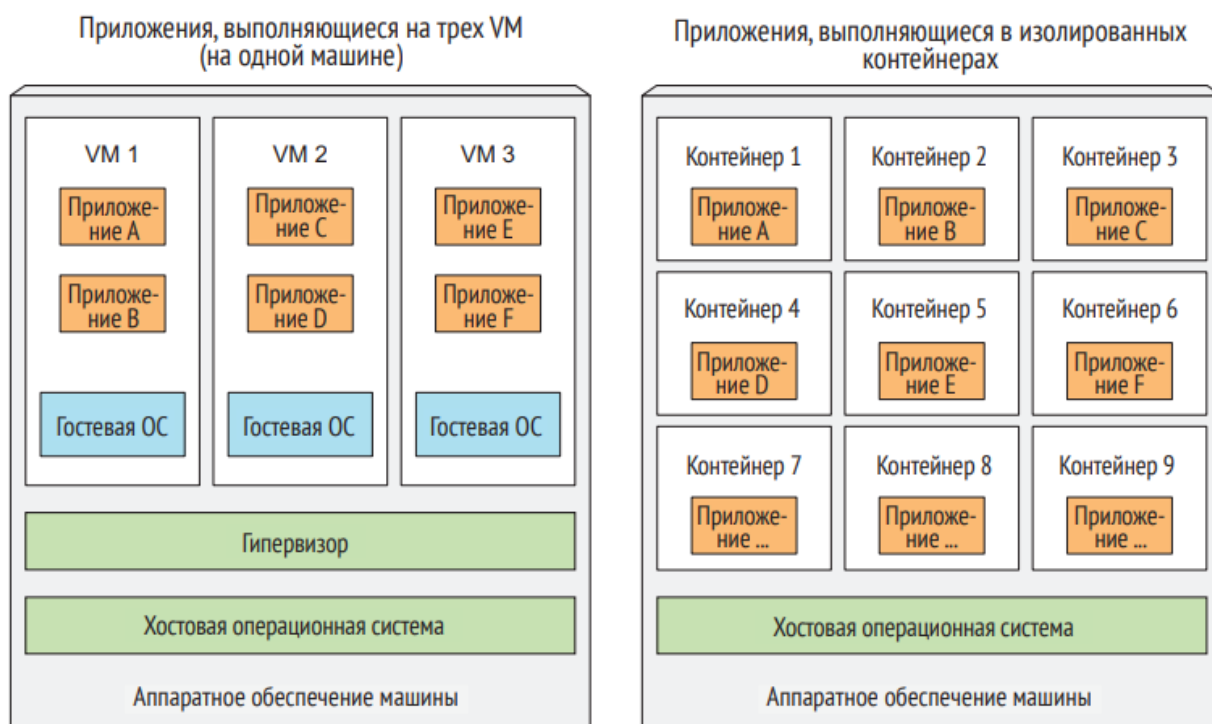


Рисунок 13 – Сравнение виртуальных машин для изоляции групп приложения с изоляцией отдельных приложения с помощью контейнеров

Контейнеры выполняют системные вызовы на одном и том же ядре, работающем в хостовой ОС. Это единственное ядро, выполняющее

инструкции на процессоре хоста. ЦП не нужно делать какой-либо виртуализации, как он делает с виртуальными машинами.

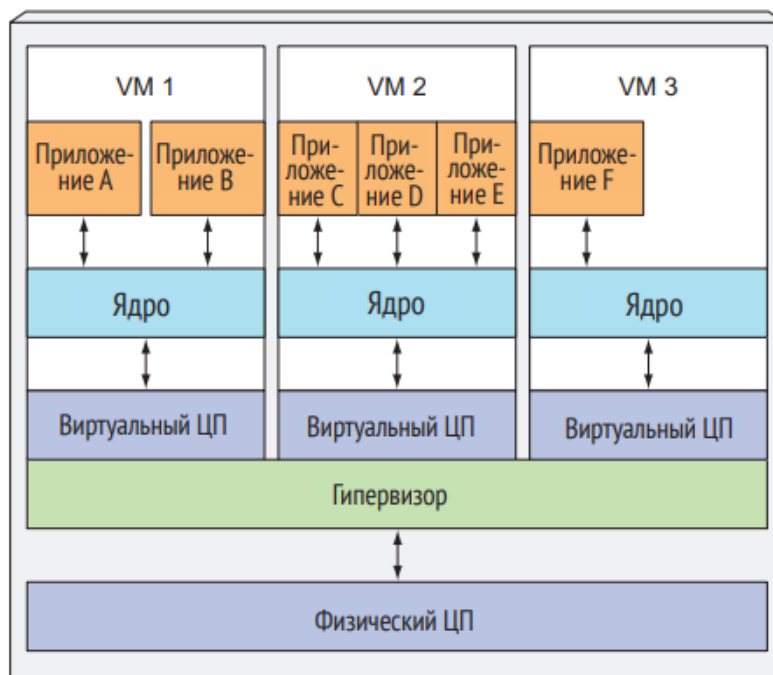


Рисунок 14 – Приложения, выполняющиеся на множестве виртуальных машин

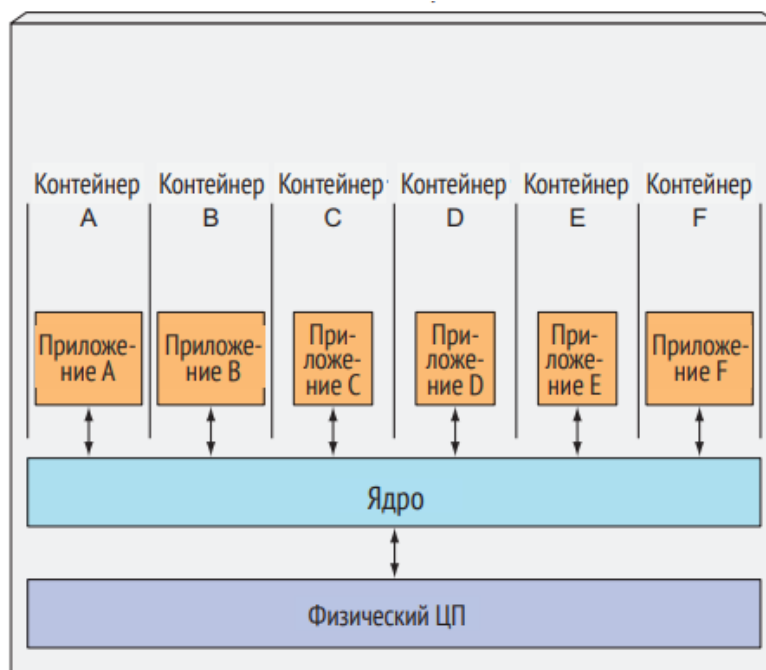


Рисунок 15 – Приложения, выполняющиеся в изолированных контейнерах

### 2.2.3 Анализ и выбор технологии контейнеризации для разрабатываемой системы

Существуют реализации, ориентированные на создание практически полноценных экземпляров операционных систем (Solaris Containers, контейнеры Virtuozzo, OpenVZ), так и варианты, фокусирующиеся на изоляции отдельных сервисов с минимальным операционным окружением (jail, Docker).

Таблица 4 – Сводная таблица технологий контейнеризации

Механизм	Операционная система	Лицензия	Дата выпуска	Особенности					
				Изоляция файловой системы	Квоты на пространство хранения	Лимиты на ввод-вывод	Лимиты на память	Квоты ЦПУ	Изоляция сети
chroot	встроено в большинство Unix-подобных операционных систем	в зависимости от лицензии на операционную систему	1982	Частично	Нет	Нет	Нет	Нет	Нет
Docker	Linux, FreeBSD, Windows, macOS	Apache 2.0	2013	Да	Да	Да	Да	Да	Да
Solaris Containers	Solaris, OpenSolaris	CDDL	01/2005	Да	Да	Нет	Да	Да	Да <sup>[1]</sup>
FreeVPS	Linux	GNU GPL	-	Да	Да	Нет	Да	Да	Да
iCore Virtual Accounts	Windows XP	Проприетарное	06/2008	Да	Да	Нет	Нет	Нет	Да
Linux-VServer <sup>[en]</sup>	Linux	GNU GPL v.2	-	Да	Да	Да	Да	Да	Да <sup>[3]</sup>
LXC	Linux	GNU GPL v.2	2008	Да	Нет	Да	Да	Да	Да
OpenVZ	Linux	GNU GPL v.2	-	Да	Да	Да <sup>[4]</sup>	Да	Да	Да <sup>[5]</sup>
Virtuozzo Containers	Linux, Microsoft Windows	Проприетарное	-	Да	Да	Да <sup>[6]</sup>	Да	Да	Да <sup>[5]</sup>
FreeBSD Jail	FreeBSD	BSD	03/2000	Да	Да	Нет	Да	Частично	Да
sysjail <sup>[en]</sup>	OpenBSD, NetBSD	BSD	-	Да	Нет	Нет	Нет	Нет	Да
WPAR <sup>[en]</sup>	AIX	Проприетарное	10/2007	Да	Да	Да	Да	Да	Да <sup>[7]</sup>

По таблице 4 можно увидеть, что существует большое количество реализаций технологии контейнеризации с различными особенностями, однако самая популярная на данный момент – это Docker.



## 2.2.4 Docker

Хотя контейнерные технологии существуют уже давно, они стали более широко известны с появлением контейнерной платформы Docker.

Docker была первой контейнерной системой, которая сделала контейнеры легко переносимыми на разные машины. Это упростило процесс упаковки не только приложения, но и всех его библиотек и других зависимостей, даже всей файловой системы ОС, в простой, переносимый пакет, который может использоваться для подготовки приложения к работе на любой другой машине, на которой работает Docker.

При выполнении приложения, упакованного с помощью Docker, оно видит точное содержимое файловой системы, поставляемое вместе с ним. Оно видит одни и те же файлы, независимо от того, работает ли оно на машине, предназначенной для разработки, или же на машине из рабочего окружения, даже если на рабочем сервере запущена совершенно другая ОС Linux.

Процесс создания образа и запуска контейнера проиллюстрирован на рисунке ниже.

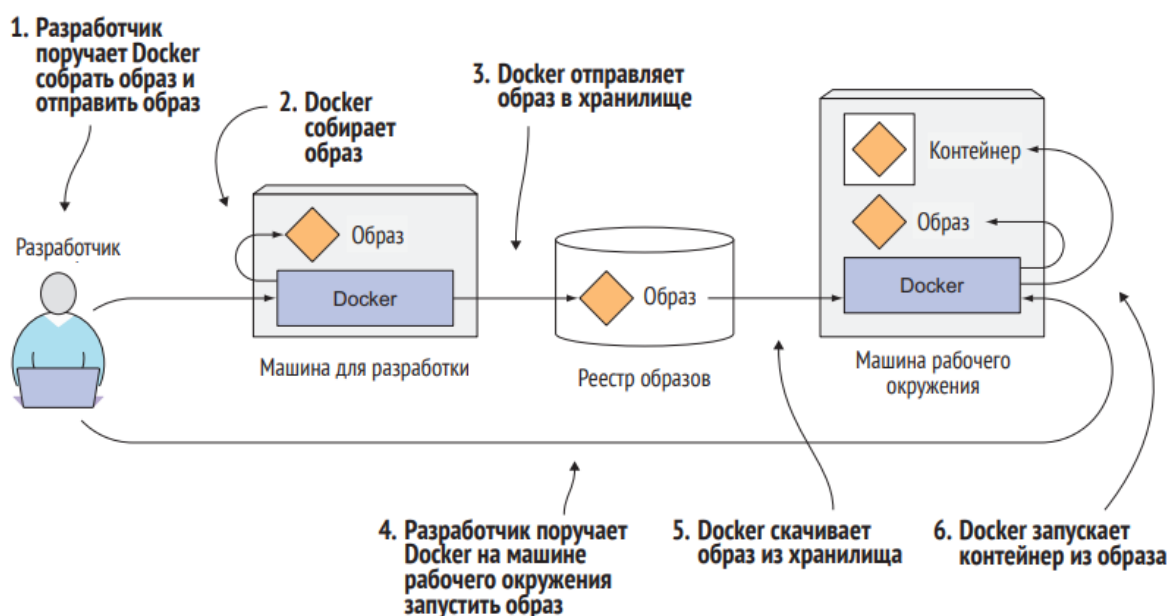


Рисунок 16 – Процесс создания докер-образа и запуска контейнера на рабочем узле программной системы

## **2.3 Проектирование инфраструктуры для разрабатываемой системы**

По мере роста программной системы ею становится все сложнее управлять из-за количества разнородных разворачиваемых компонентов.

Построение инфраструктуры – это добавление в систему компонентов, которые не обеспечивают никакой функционал для пользователя. Инфраструктурные компоненты обеспечивают логирование, упрощение настройки системы, мониторинг компонентов системы, её масштабируемость и отказоустойчивость.

По мере роста системы и количества её компонентов применение инфраструктурных компонентов становится все более выгодным и полезным.

### **2.3.1 Система оркестровки контейнерами Kubernetes**

Kubernetes – это программная система, которая позволяет легко разворачивать контейнеризированные приложения и управлять ими. Она использует возможности контейнеров Linux для запуска разнородных приложений без необходимости знать какие-либо внутренние детали этих приложений и без необходимости вручную разворачивать эти приложения на каждом хосте.

Поскольку данные приложения работают в контейнерах, они не влияют на другие приложения, работающие на том же сервере, что имеет решающее значение при запуске приложений для совершенно разных организаций на одном и том же оборудовании.

Это имеет первостепенное значение для облачных провайдеров, поскольку они стремятся к максимально возможной задействованности своего оборудования, сохраняя при этом полную изоляцию размещенных приложений.

Kubernetes позволяет выполнять программные приложения на тысячах компьютерных узлов, как если бы все эти узлы были одним огромным компьютером. Она абстрагируется от базовой инфраструктуры и тем самым

упрощает разработку, развертывание и управление как для разработчиков, так и для системных администраторов.

Процедура развертывания приложений через Kubernetes всегда одинаковая, независимо от того, содержит ли кластер всего несколько узлов или тысячи. Размер кластера не имеет никакого значения. Дополнительные узлы кластера просто представляют собой дополнительный объем ресурсов, доступных для развернутых приложений.

Простейший вид системы Kubernetes представлен на рисунке ниже. Система состоит из ведущего узла (мастера) и любого количества рабочих узлов. Когда разработчик отправляет список приложений ведущему узлу, Kubernetes развертывает их в кластере рабочих узлов. То, на какой узел приземляется компонент, не имеет (и не должно иметь) значения ни для разработчика, ни для системного администратора.

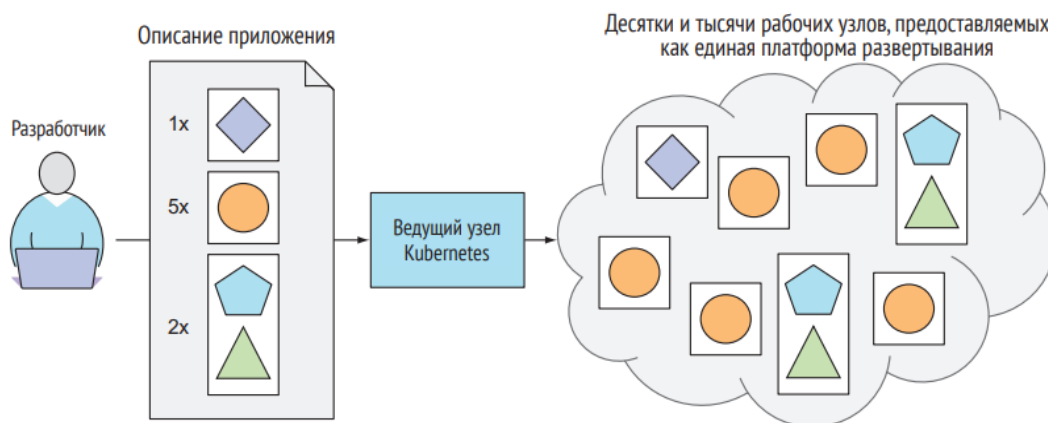


Рисунок 17 – Общий вид системы Kubernetes

Kubernetes можно рассматривать как операционную систему для кластера. Она избавляет разработчиков приложений от необходимости внедрять в свои приложения определенные службы, связанные с инфраструктурой; вместо этого в вопросе предоставления этих служб они опираются на Kubernetes. Это включает в себя такие аспекты, как обнаружение службы, масштабирование, балансировка нагрузки, самовосстановление и даже выбор лидера. Поэтому разработчики приложений могут

сосредоточиться на реализации реального функционала приложений и не тратить время на то, чтобы разбираться в том, как интегрировать их с инфраструктурой.

### 2.3.2 Архитектура кластера Kubernetes

На аппаратном уровне кластер Kubernetes состоит из множества узлов, которые можно разделить на два типа:

- ведущий узел (мастер), на котором размещена плоскость управления (Control Plane) Kubernetes, контролирующая и управляющая всей системой Kubernetes;
- рабочие узлы, на которых выполняются разворачиваемые приложения.

На рисунке ниже показаны компоненты, работающие на этих двух типах узлов Kubernetes.

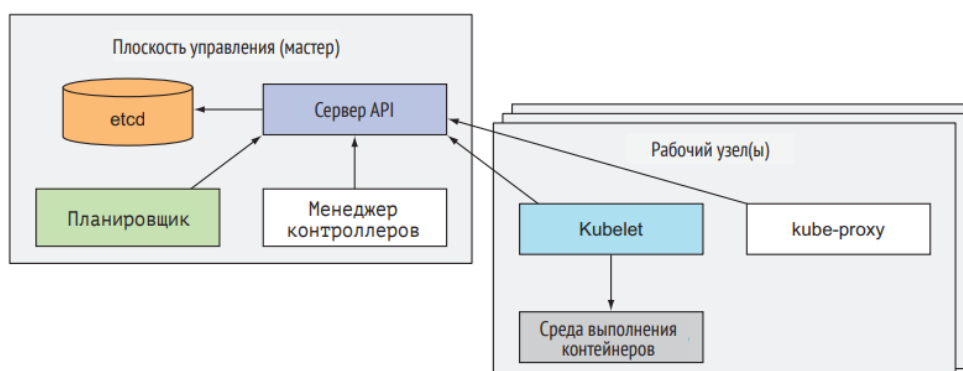


Рисунок 18 – Компоненты кластера Kubernetes

Плоскость управления – это то, что управляет кластером и заставляет его функционировать.

Рабочие узлы – это физические машины, на которых запускаются контейнеры.

Для запуска приложения в Kubernetes необходимо упаковать его в образ контейнера, отослать его в хранилище образов, после чего опубликовать описание приложения на сервере API Kubernetes.

Описание содержит такие сведения, как образ или образы контейнеров, содержащие компоненты приложения, как эти компоненты связаны друг с другом и какие из них должны выполняться совместно, а какие нет.

### 2.3.3 Процесс развертывания компонентов программной системы в Kubernetes

Когда сервер API обрабатывает описание приложения, планировщик назначает указанные группы контейнеров доступным рабочим узлам, исходя из вычислительных ресурсов, требуемых каждой группой, и нераспределенных ресурсов на каждом узле в данный момент. Агент Kubelet на этих узлах затем поручает среде выполнения контейнеров (например, Docker) извлечь из хранилища требуемые образы контейнеров и запустить контейнеры.

После отправки дескриптора в Kubernetes он запланирует использование указанного количества реплик каждого модуля на доступных рабочих узлах. Затем агенты Kubelet на узлах поручают Docker извлечь образы контейнеров из реестра образов и запустить контейнеры

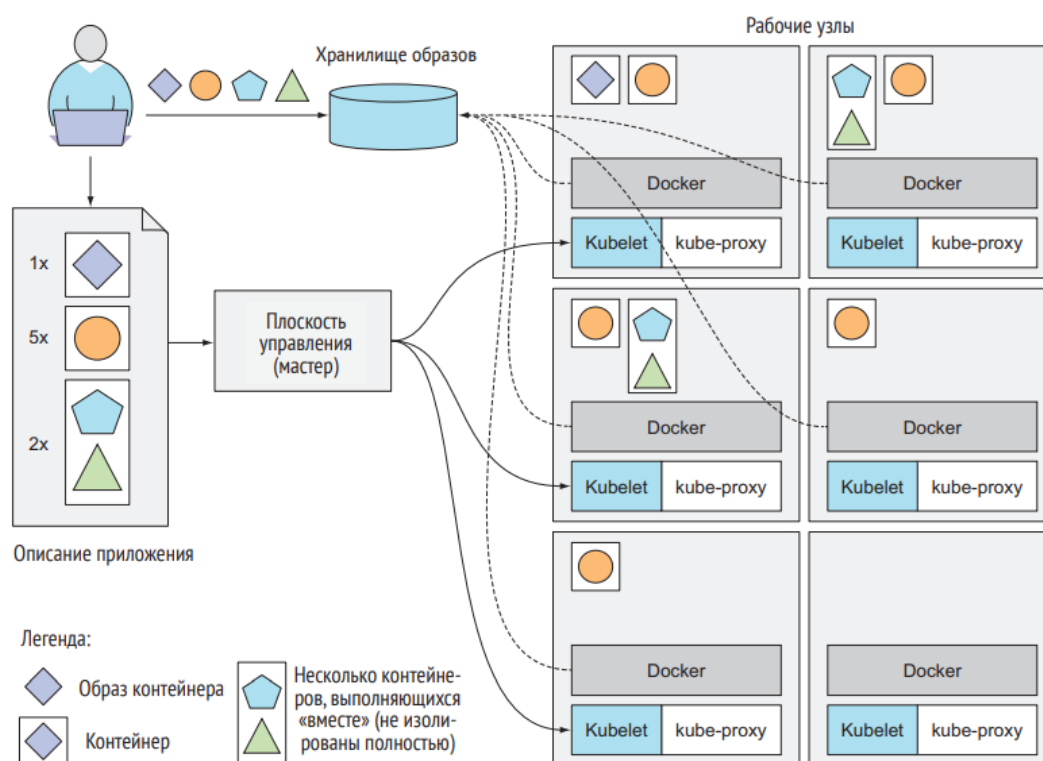


Рисунок 19 – Процесс развертывания приложения в Kubernetes

### **2.3.4 Преимущества использования Kubernetes**

1) Упрощенное развертывание приложений. Kubernetes обеспечивает доступ ко всем своим рабочим узлам как к единой платформе развертывания. По сути, все узлы теперь представляют собой единую группу вычислительных ресурсов, которые ждут, когда приложения будут их потреблять.

2) Повышение эффективности задействования оборудования. Настройка кластера Kubernetes позволяет отделить программную систему от инфраструктуры. Компоненты программной системы могут свободно запускаться и перемещаться между доступными узлами в кластере, что позволяет эффективно использовать аппаратные ресурсы.

3) Проверка здоровья и самолечение. Kubernetes отслеживает компоненты приложения и узлы, на которых они выполняются, и автоматически переносит их на другие узлы в случае аварийного сбоя узла. Также в случае аварийного завершения работы одного из рабочих физических узлов Kubernetes, работавшие до этого на нем модули будут автоматически переназначены на другие узлы, тем самым обеспечивая бесперебойность работы всей системы.

4) Автоматическое масштабирование. Если Kubernetes работает в облачной инфраструктуре, где добавлять дополнительные узлы так же просто, как запрашивать их через API поставщика облака, Kubernetes даже может автоматически масштабировать размер всего кластера вверх или вниз в зависимости от потребностей развернутых приложений.

5) Упрощение разработки приложений. Система позволяет быстро разворачивать новые версии программных компонентов. Kubernetes позволяет также настроить процесс постепенного обновления программных компонентов, сделав его наиболее безопасным и плавным. В случае аварийной ситуации Kubernetes может прервать процесс выкладки и вернуться к предыдущей стабильной конфигурации кластера.

## **2.4 Проектирование программной системы с использованием Kubernetes и Docker**

Для проектирования программной системы с использованием технологии контейнеризации и системы оркестровки контейнерами Kubernetes необходимо выделить компоненты разрабатываемой системы, а также взаимосвязи между ними, после чего интегрировать их в кластер Kubernetes, используя предоставляемые им инструменты и сервисы.

### **2.4.1 Выбор базовых программных компонентов проектируемой системы**

Поскольку разрабатываемая система представляет из себя веб-приложение, доступ к которому осуществляется посредством использования сети Интернет, то для отображения графического интерфейса будет использоваться браузер.

Минимальная реализация подобной программной системы должна включать в себя минимум три компонента: сервис для отдачи веб-приложения браузеру, сервис для обработки запросов от веб-приложения и хранилище данных.

Для реализации данных программных компонентов разрабатываемой системы были выбраны следующие языки программирования и технологии:

- для отдачи веб-приложения на браузер пользователя будет использоваться frontend-сервер Node.js;
- для обработки действия пользователя, отдачи данных и обработки запросов будет использоваться веб-сервис, реализованный на языке программирования GO;
- для хранения данных пользователей и служебных данных разрабатываемой системы, планируется использовать реляционную базу данных Postgres 10.1.

## 2.4.2 Простейший вариант программной архитектуры проектируемой системы

Наиболее простой вариант программной архитектуры с использованием frontend-backend подхода представлен на рисунке 20.

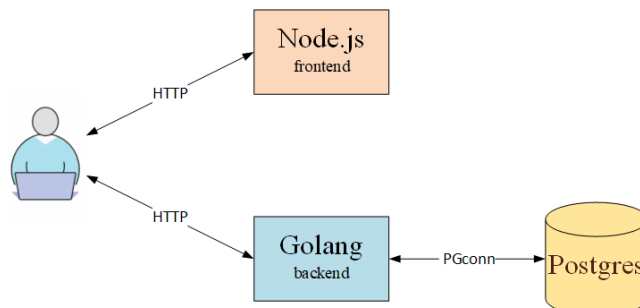


Рисунок 20 – Простой вариант программной архитектуры веб-приложения

Данный подход к проектированию программной системы имеет как плюсы, так и минусы.

Плюсы:

- при такой архитектуре количество разнородных программных компонентов не велико;
- нет накладных расходов на дополнительные сервисы, обеспечивающие инфраструктуру приложения (балансировки нагрузки, прокси-серверы, регистраторы сервисов и другие)/

Минусы:

- отсутствует единая точка входа для пользователя в программную систему;
- масштабируемость для каждого из компонентов системы требует изменение конфигурации других элементов системы;
- компоненты системы представляют из себя большие монолитные программы, некоторый функционал из которых может быть вынесен в отдельные сервисы;
- отсутствуют компоненты программной системы, позволяющие равномерно распределить нагрузки между однородными компонентами системы;



– выход из строя одного из компонентов приводит к отказу всей системы.

С учётом всех плюсов и минусов описанная выше архитектура может быть использована в небольших проектах, но для высоконагруженных систем она не годится в связи с громоздкостью отдельных компонентов и потенциальными проблемами при масштабировании.

### 2.4.3 Масштабируемый архитектура проектируемой системы

На рисунке 21 представлен вариант архитектуры программной системы с учетом необходимости в масштабировании отдельных компонентов и требованием к отказоустойчивости.

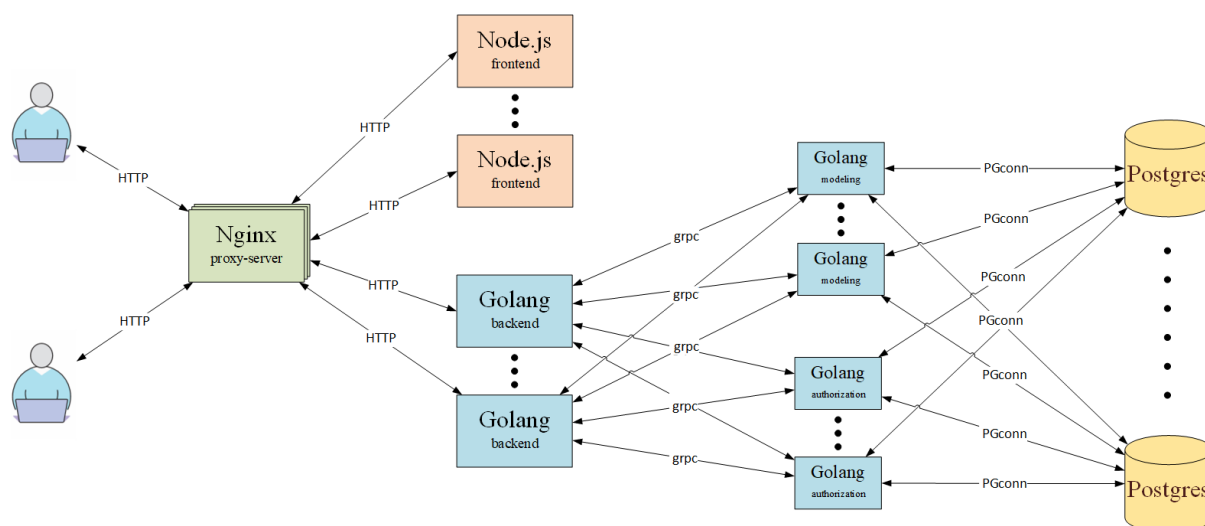


Рисунок 21 – Программная система с масштабируемой архитектурой

Заметна существенна разница между первым вариантом архитектуры и новым. Можно также выделить в данной архитектуре список достоинств и недостатков.

Достоинства:

- каждый компонент проектируемой системы может быть горизонтально отмасштабирован;
- нет единой точки отказа, выход из строя одного из компонентов системы не ведет к потере ее работоспособности;

- для добавления дополнительного frontend или backend сервера достаточно только изменить конфигурацию прокси-сервера nginx.

Недостатки:

- в данной системе добавление компонентов при масштабировании все еще требует изменению конфигурации других компонентов;
- отсутствует инфраструктура для автоматического масштабирования в зависимости от текущей нагрузки;
- данная архитектура требует от администратора системы дополнительных временных затрат времени из-за в целом более высокой сложности по сравнению с предыдущим вариантом.

Всё выявленные недостатки в данной архитектуре можно исправить при помощи использования системы оркестровки контейнеров Kubernetes.

#### **2.4.4 Реализация проектируемой системы в кластере Kubernetes**

Для переноса программных компонентов системы необходимо прежде всего использовать технологию контейнеризации Docker. За счет этого каждый программный компонент будет работать изолированно от других запущенных на физической машине контейнеров и независимо от самой ОС и конфигурации этой самой машины.

Для построения инфраструктуры для проектируемой программной системы в кластере Kubernetes потребуются перечень инструментов Kubernetes, представленный в таблице 5.

Таблица 5 – Используемые средства Kubernetes

Название средства Kubernetes	Тип	Выполняемая функция
Deployment	Контроллер развертывания	Развертывание экземпляров одного из компонентов программной системы
Service	Сервис	Обеспечение доступа между компонентами программной системы и из вне
Vault	Хранилище	Создание временных и постоянных хранилищ данных
Secret	Чувствительная информация	Хранения паролей и секретной информации для обеспечения контроля доступа

Итоговая архитектура представлена на рисунке ниже.

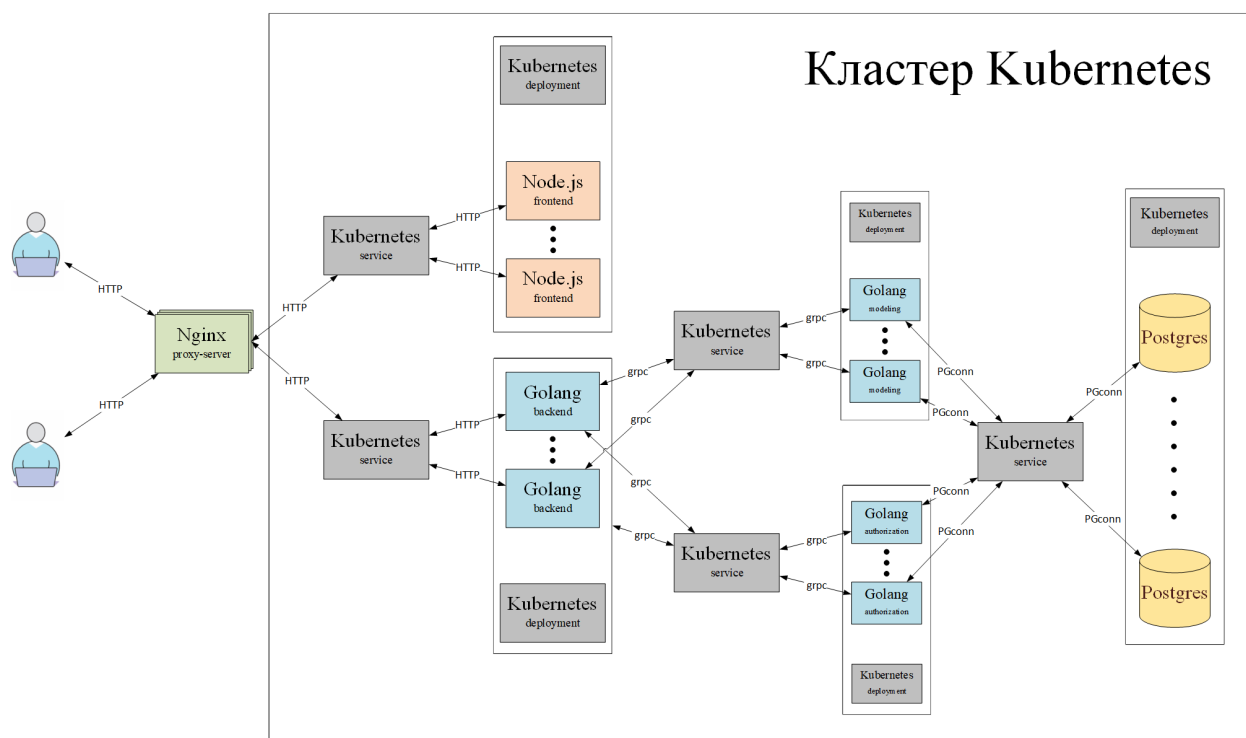


Рисунок 22 – Архитектура программной системы в Kubernetes

Такие средства Kubernetes, как vault и secret создаются и настраиваются посредством конфигурации соответствующих опций в ресурсах deployment. Кластер Kubernetes самостоятельно определит машины для размещения хранилища данных (vault) и паролей (secret).

Ресурсы deployment позволяют в автоматическом режиме разворачивать экземпляры программных компонентов и увеличивать/уменьшать их количество в зависимости от текущей нагрузки (если соответствующие критерии были определены администратором).

В случае использования такого ресурса kubernetes, как service, больше нет необходимости менять конфигурацию компонентов программной системы в случае изменения количества работающих элементов. Сервисы самостоятельно отслеживают запуск новых экземпляров программных компонентов и обеспечивают равномерное распределение нагрузки.

Kubernetes использует декларативный подход. Это значит, что администратору достаточно только описать желаемое состояние кластера, а дальше сама система уже сделает все необходимое, чтобы прийти к заданному состоянию. Это значительно облегчает расходы на администрирование и обслуживание подобных систем.

## **2.5 Проектирование модели агента и генетического алгоритма**

Для разработки агента необходимо спроектировать его модель и поведение, а также определить его свойства и правила взаимодействия с окружающей средой.

### **2.5.1 Построение модели агента на основе цифрового автомата**

В качестве основы для модели агента берется абстрактный цифровой или конечный автомат. При этом автоматы являются конечными.

Абстрактный автомат включает в себя два основным блока:

- Логический преобразователь (комбинационная схема)
- Блок памяти (каскад регистров)

По аналогии с логическим преобразователем в цифровом автомате, в модели агента будет представлен блоком кода-генома, а блок памяти будет представлен параметрами окружения и состояния самого агента.

Код-геном состоит из набора ячеек, в которых записываются числа, где каждое число соответствует команде. Набор этих команд определяет поведение агента и изменение этого поведения для последующих поколений.

Параметры окружения – это ближайшее пространство вокруг агента, доступное для анализа. Состояние агента – это набор параметров, характеризующие самого агента, они могут быть представлены текущим запасом энергии, органических и неорганических веществ, воды, возрастом и т.д.

Описанное выше соответствие представлено на рисунке 23.

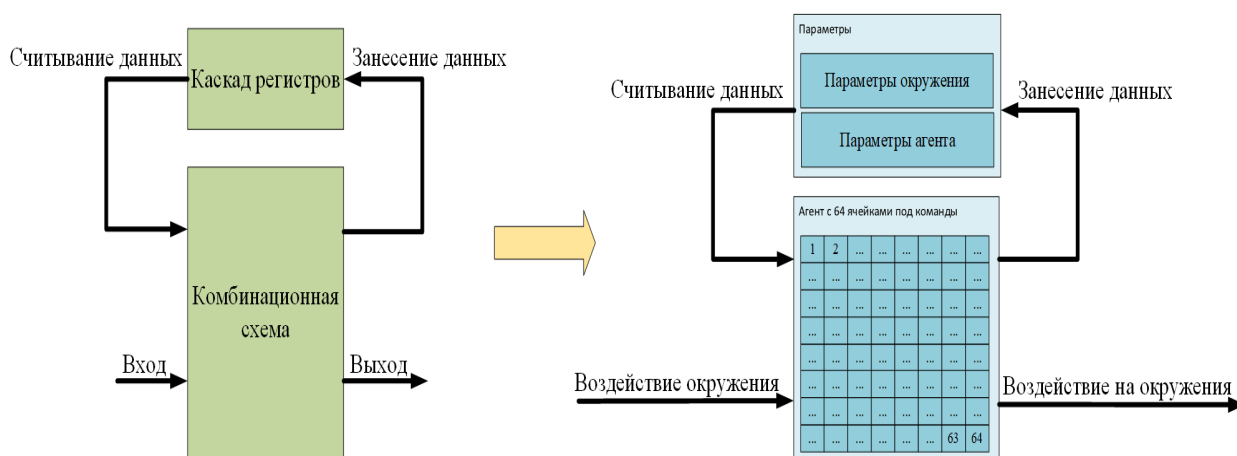


Рисунок 23 – Построение модели агента на основе модели цифрового автомата

### 2.5.2 Код-геном

Код-геном представляет из себя цепочку чисел. Каждое число — это какая-то команда (в простейшем случае). Также есть указатель текущей команды (далее УТК), который показывает, какая команда будет сейчас выполняться и после выполнения команды, указатель перемещается к следующей команде. Если указатель вышел за край цепочки, то он появляется с противоположной стороны, то есть цепочка команд замкнута по кругу.

В качестве примера на рисунке 24 показана лента на 64 команды, где все ячейки забиты командой «23».

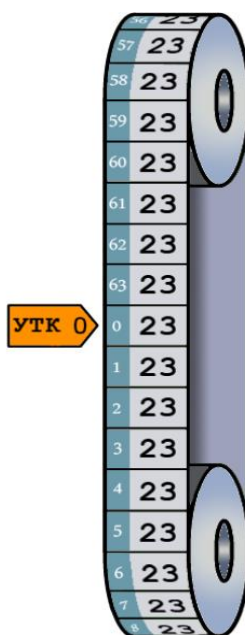


Рисунок 24 – Лента команд агента

Некоторым числам соответствуют определенные команды. Если числу не соответствует никакая команда, то это число является безусловным переходом. Например, для данной ленты, размером в 64 ячейки, можно предусмотреть диапазон возможных значений для ячеек от 0 до 127, где первые 64 значения являются командами безусловного перехода, то есть им не присвоена никакая другая команда.

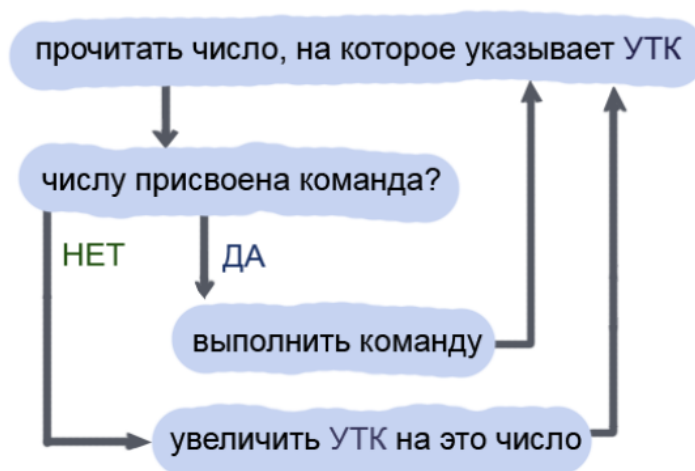


Рисунок 25 – Алгоритм определения присвоена ли числу команда

Стоит отметить, что команда, кроме выполнения основной функции, также отвечает за изменение УТК. После выполнения команды УТК увеличивается на единицу. Пусть для значения 23 назначена какая-нибудь команда, тогда изменение УТК будет выглядеть следующим образом, представленным на рисунке 26.

УТК+62	63	23	
УТК+63	0	23	команда
<b>УТК</b>	1	63	УТК = УТК + 1
УТК+1	2	23	не назначена команда
			УТК = УТК + 63

Рисунок 26 – Пример изменения УТК

Некоторые команды могут быть более сложными и требовать параметры для своего исполнения.

В качестве примера можно взять команду поворота для агента, обитающего в среде, поделенной на клетки. В это случае агент может быть повернут в 8 возможных направления (4 грани и 4 угла). Для выбора направления может использоваться параметр (рис. 27).

УТК+63	8	18	
<b>УТК</b>	9	25	команда "повернуть"
УТК+1	10	3	УТК = УТК + 2
УТК+2	11	43	команда
			параметр = 3 % 8 = 3
			параметр

Рисунок 27 – Команда с параметром

Например, числу 25 присвоена «поворот». Когда УТК укажет на ячейку с числом 25 (повернуть), то для выполнения данной команды будет взято следующее. Из этого числа будет выбрано направление, куда необходимо повернуть. Число может быть от 0 до какого-либо еще, а направлений 8. Что бы узнать направление, можно поделить параметр на 8 и взять остаток от

деления. Получится одно из 8 значений (0..7) — это и есть направление, куда необходимо совершить поворот. В данном случае важно, чтобы количество возможных чисел было кратно 8. В таком случае вероятность выбора одинакова для всех направлений. После выполнения команды, УТК перемещается не на одну ячейку, а на две, перепрыгивая параметр.

УТК может прийти к ячейке, которая в другой раз служила параметром. Теперь значение из этой ячейки будет командой. Это сильно усложняет анализ кода человеком, зато «виртуальная машина», выполняющая код бота очень проста в реализации.

Для реализации ветвления может использоваться вариант команды, когда изменение значения УТК зависит от параметра.

УТК+63	9	60	
<b>УТК</b>	10	33	команда "сколько энергии?" команда
УТК+1	11	14	параметр = 14 параметр
УТК+2	12	23	if ( энергии >= 14 ) { УТК = УТК + 23 } смещения для
УТК+3	13	8	if ( энергии < 14 ) { УТК = УТК + 8 } переходов
УТК+4	14	17	

Рисунок 28 – Команда, использующее разное смещение для УТК после выполнения

Где значения 23 и 8 – это безусловные относительные переходы.

### 2.5.3 Параметры окружения и агента

Параметры показывают текущее состояния агента или его окружения.

Под параметрами окружения подразумевается ближайшая область или окрестность, которую может проанализировать агент. Параметрами окружения могут быть уровень освещенности, наличие других агентов по близости, источники энергии и др.

Под состоянием агента может иметься ввиду уровень запасов энергии, органики, жидкости, значение УТК и прочее.



#### **2.5.4 Генетический алгоритм и метод отбора**

В разрабатываемой системе генетический алгоритм, в зависимости от параметров моделирования, настроенных пользователем системы, может работать по-разному, однако в любом случае он обязан включать 4 основных этапа:

- генерирование начальной популяции;
- селекция
- скрещивание
- мутация

Основной преследуемой целью при реализации данных выше механизмов – обеспечение максимальной гибкости настройки для пользователя.

Создание абсолютно универсального программного компонента для моделирования чего-угодно не представляется возможным. Вместо этого предполагается создания нескольких отличных друг от друга шаблонов, которые будут обладать обозримым числом параметров для гибкой настройки моделирования.

Несмотря на то, что метод отбора часто рассматривается как отдельная часть, работающая рука об руку с генетическим алгоритмом (где генетический алгоритм отвечает за рекомбинацию генов, а метод отбора за селекцию агентов и формирование поколений), в данной разрабатываемой системе, ввиду специфики тематики (моделирование естественных эволюционных процессов) далее, непосредственно при программной реализации, эти две части будут рассматриваться как единое целое в лице разрабатываемого генетического алгоритма, где метод отбора является одним из этапов работы генетического алгоритма, то есть этапом «селекция».

## 2.6 Хранение данных

Для корректного функционирования системы необходимо хранить большие объемы разнородной информации. В контексте данной автоматизированной системы данные можно разделить на следующие типы:

- данные пользователя (логин, пароль, почта, сессия пользователя, прочие личные данные)
- данные моделирования (владелец сеанса моделирования, различные параметры сеанса моделирования, данные о текущей популяции и т.д.)
- служебные данные (количество зарегистрированных пользователей, количество пользователей онлайн, список активных симуляций, логи ошибок и прочее)

Для хранения столь разнообразной информации было принято решения использовать реляционную базу данных ввиду универсальности таких баз, а также большого количества инструментов в современных инструментов для выборки данных, их модификации, архивирования, восстановления и защиты от нежелательного пагубного воздействия со стороны потенциальных злоумышленников.

Для формирования ядра БД используется даталогическая модель.

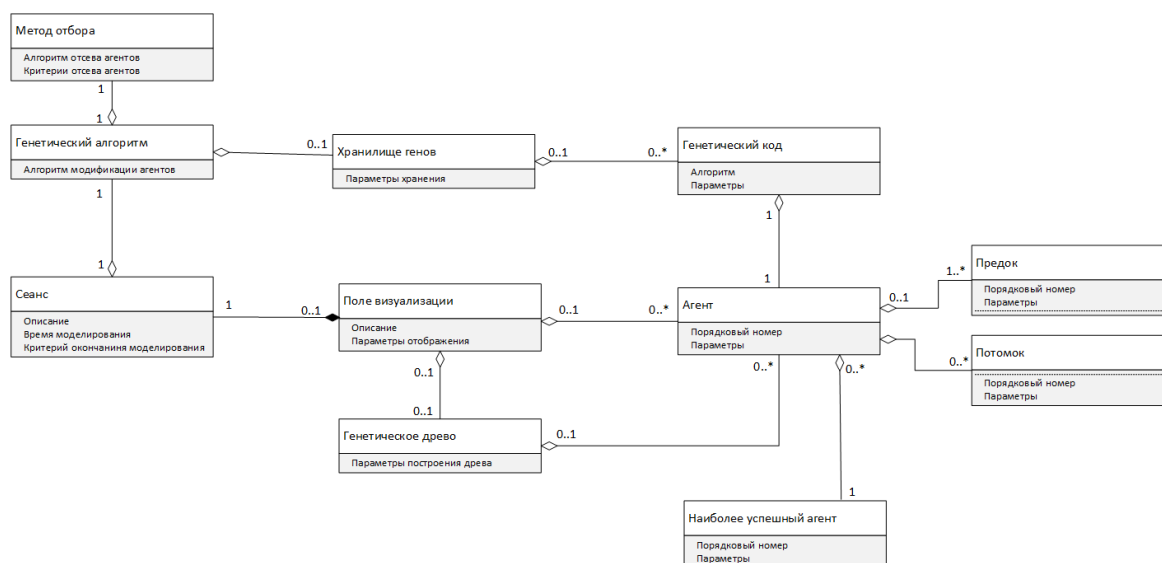


Рисунок 29 – Даталогическая модель предметной области

## 2.7 Выявление угроз в проектируемой системе

Данная автоматизированная система представляет собой веб-приложение, доступное множеству пользователей в сети Интернет. Архитектурная схема программного обеспечения рассматриваемой системы представлена на рисунке 21. Согласно данной схеме будут выявляться риски и возможные уязвимости рассматриваемой системы, а также будут рассмотрены возможные способы их решения.

В представленной системе присутствуют следующие компоненты:

- Прокси-сервер nginx;
- Frontend-сервер Node.js;
- Backend-сервер Golang;
- Веб-сервис моделирования Golang;
- База данных PostgreSQL.

Для каждого из перечисленных компонентов будут рассмотрены возможные уязвимости.

### 2.7.1 Нарушение доступности

Нарушение доступности может произойти вследствие атаки на ряд компонентов системы.

#### 2.7.1.1 Атака на proxy-server

Нарушения доступности системы может произойти в первую очередь из-за вывода из строя сервера nginx, ввиду того, что он представляет из себя единую точку входа в систему.

Нарушение работы nginx может произойти по следующим причинам:

- падение из-за высокой нагрузки и недостатка ресурсов;
- падение под воздействием DDoS атаки (формирование большого количества запросов);
- падение под натиском большого количества медленных клиентов (один из вариантов DDoS атаки, приводит к быстрому росту необходимых поддерживаемых соединений);

– падение под тяжестью «толстых клиентов» (необходимо проксировать недопустимо большие объемы данных).

#### **2.7.1.2 Потенциальные угрозы для frontend-сервера**

Преднамеренное большое количество запросов на получение веб-приложения может привести к увеличению среднего времени отклика, что приведет к дополнительному времени ожидания загрузки сайта и понижению уровня доступности приложения в целом.

#### **2.7.1.3 Высокая нагрузка на backend-server**

Большинство проблем, которые возникают на стадии проксирования, также актуальны и для этого компонента, если они не были решены ранее.

### **2.7.2 Нарушение целостности рассматриваемой автоматизированной системы**

Нарушение целостности может быть вызвано выводом из строя любого из компонентов программного обеспечения данной автоматизированной системы.

#### **2.7.2.1 Внедрение вредоносного кода и sql-инъекции**

Внедрение в веб-приложение вредоносного кода и запросов может привести к целому ряду проблем, в том числе и к нарушению целостности из-за появления ошибок при работе отдельных компонентов и выхода их из строя.

#### **2.7.2.2 Целенаправленное создание высокой нагрузки на связующие элементы системы**

Создание нагрузки на компоненты системы, обеспечивающие жизнеспособность всей системы в целом может привести не только к проблемам доступности всей системы, но и к нарушению корректности работы связанных элементов и даже к выводу их из строя.

Может иметь место, если система спроектирована как сильносвязанная, каскадный эффект, который приведет к полной неработоспособности всей системы.

### **2.7.2.3 Уничтожение физических серверов**

Уничтожение физических серверов, если система не имеет резервов, неизбежно ведет к нарушению целостности.

### **2.7.3 Нарушение конфиденциальности информации**

Рассматриваемая система является веб-приложением, которое доступное в сети Интернет множеству пользователей, что означает, что существует множество аккаунтов пользователей, которые не должны иметь доступ к данным чужих профилей и модифицировать их.

#### **2.7.3.1 SQL-инъекции в базе данных для получения доступа к приватным данным**

Внедрение SQL, в зависимости от типа используемой СУБД и условий внедрения, может дать возможность атакующему выполнить произвольный запрос к базе данных (например, прочитать содержимое любых таблиц, удалить, изменить или добавить данные), получить возможность чтения и/или записи локальных файлов и выполнения произвольных команд на атакуемом сервере.

Атака типа внедрения SQL может быть возможна из-за некорректной обработки входных данных, используемых в SQL-запросах.

#### **2.7.3.2 Внедрение XSS в frontend-приложение**

XSS не так опасна для сервера, но опасна для пользователя. Она работает в браузере пользователя, и поэтому позволяет только украсть его данные. XSS или Cross-Site Scripting работает в JavaScript. Злоумышленник передает специальную строку, в строке содержится JS код, далее браузер думает, что этот код отправлен сайтом и выполняет его, а код может быть любым.

#### **2.7.3.3 Уязвимость междоменного взаимодействия CSRF**

Атака CSRF или Cross-Site Request Forgery позволяет злоумышленнику заставить браузер жертвы отправить определенный HTTP запрос, включая куки, файлы сеанса и любую другую, автоматически включаемую информацию в уязвимое веб-приложение.

#### **2.7.4 Резюме на выявленные угрозы**

Основным источником угрозы является пользователь-злоумышленник (внешняя преднамеренная угроза).

Основными объектами угрозы является аккаунты пользователя и их приватная информация.

Возможные способы несанкционированного доступа – похищение пароля пользователя, перехват сессии, внедрение вредоносного кода в компоненты программной системы.

### **2.8 Управление доступом на основе ролей**

Несмотря на то, что разрабатываемая система не преследует никаких коммерческих целей, что автоматически означает, что в системе не будет осуществляться товарно-денежный оборот, а также в системе не планируется хранение специфичной информации пользователя, которую он мог бы посчитать конфиденциальной, все же есть необходимость в приватности данных пользователей, а также разделение пользователей системы по ролям.

Разделение по ролям позволит наделить часть привилегированных пользователей особыми правами по модификации и просмотру данных, что необходимо в рамках администрирования системы.

Для разрабатываемой системы планируется выделить всего две роли:

- пользователь;
- администратор.

Где пользователь – обычный пользователь системы, а администратор – пользователь с особыми правами. Планируется, что администраторами будут являться сами же разработчики системы или персонал, обеспечивающий работоспособность системы.

Естественно, в отличии от администраторов, пользователи не должны обладать правами просмотра приватных данных друг друга.

### 3 Технологическая часть

#### 3.1 Разработка генетического алгоритма

В классическом представлении генетический алгоритм состоит из 4 этапов, через которые проходит каждое поколение агентов (рис. 30)

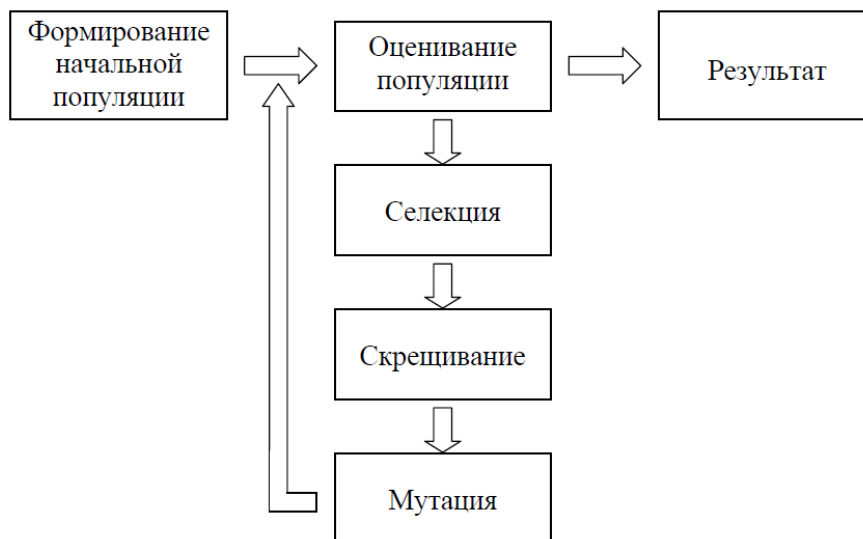


Рисунок 30 - 4 этапа работы генетического алгоритма

Формирование начальной популяции. Данный этап служит для ввода в систему первую группу агентов.

Оценка популяции осуществляется по некоторым критериям и необходима для дальнейшего отбора агентов.

Селекция представляет из себя удаление из системы тех агентов, которые после оценки были признаны негодными.

На этапе скрещивания происходит рекомбинация геномов агентов и генерация новых.

На этапе мутации новое сформировавшееся поколение агентов подвергается мутации, в результате чего их генетических под претерпевает изменение.

На этапе «Результат», может проводиться анализ и оценка как всего процесса работы генетического алгоритма (начиная с начальной популяции), так и конечных результатов

Для разработки генетического алгоритма для моделирования искусственной жизни необходимо выделить соответствия между формальными определениями и конкретной желаемой реализации в разрабатываемой системе моделирования.

Таблица 6 – Этапы работы генетического алгоритма.

<b>Этапы</b>	<b>Формальное представление</b>	<b>Реализация в разрабатываемой системе</b>
Оценка	Оценка особей по заранее определенным критериям. Составление рейтинга агентов по уровню приспособленности или оптимальности решения	Т.к. моделируемая система должна подражать эволюционным процессам, то оценка агента будет проводиться по выживаемости агента, а критерием будет являться уровень энергии, накопленный агентом
Селекция	Отбор особей после оценки. Наиболее эффективные особи остаются для формирования следующего поколения агентов, а менее эффективные отсеиваются	Селекция реализуется в неявном виде при конкуренции агентов за доступные ресурсы
Скращивание	Перекомбинация генов агентов и формирования нового поколения	Может быть реализовано как простое деления (без перекомбинации геномов между особями) или скрещивания двух особей, находящихся в непосредственной близости (при условии, что их геномы не отличаются больше, чем на заранее определенный процент)
Мутация	Случайное модификация одного из генов агента при его создании	Случайное модификация одного из генов агента при его создании



Для спроектированной модели агента используется простой генетический алгоритм. Работа генетического алгоритма заключается в случайном изменении числа в ячейке кода-генома на случайное значение. При этом вероятность мутации в ячейке может быть определена как один из параметров моделирования.

Мутация в одном из генов агента совершается в момент создания агента другим агентом, не зависимо от того, было ли это деление или селекция.

Сам агент появляется в случайной смежной ячейке от реплицирующего его агента.

Процент начальной энергии агента зависит от процента переданной ему энергии родительским агентом. Процент передаваемой энергии к дочернему агенту определяется параметрами моделирования.

### 3.1.1 Реализация модели агента

Модель агента реализуется структурой с набором методов и параметров агента. Совокупность параметров агента определяют его текущее состояние. Параметры агента разделяются на два типа:

- внутренние (уровень энергии, геном, значение УТК и др.);
- внешние (окружающее пространство со всей совокупностью инородных объектов, возможностей по перемещению, других агентов и т.д.)

Внутренние параметры представимы как значение переменных структуры агента, а внешние параметры доступны агенту для считывания и анализа посредством выполнения им соответствующих команд (пример на рисунке 31).

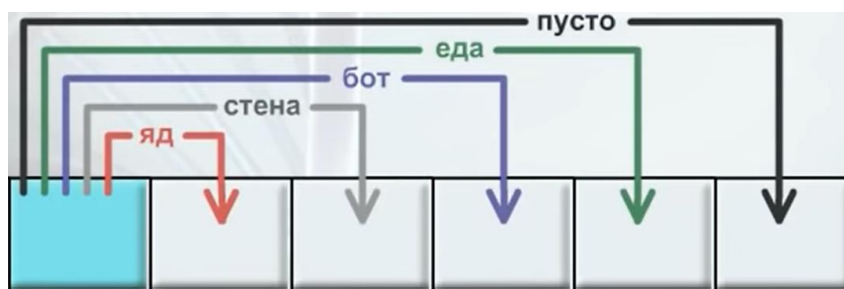


Рисунок 31 – Команда и параметры окружения

Важным аспектом для разрабатываемой системы является то, что она должна представлять возможность гибко настраивать процесс моделирования, а не предоставлять некий ограниченный набор вариантов моделирования, твёрдо заложенный разработчиком в виде программного кода.

Для обеспечения гибкости настройки не только сеанса моделирования, но и самой структуры агента, используются интерфейсы. В зависимости от значений параметров, которые выставил пользователь, интерфейс агента реализуется, а также отдельные его компоненты, реализуются по-разному (рисунок 32).

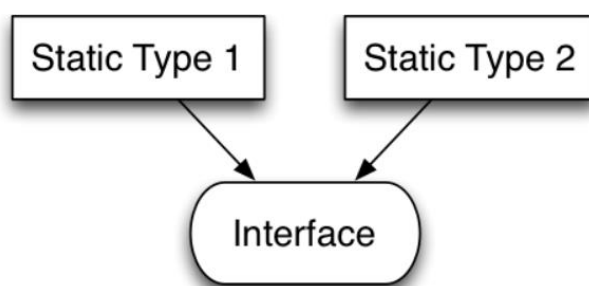


Рисунок 32 – Реализация интерфейса

Данное решение позволяет уйти от конкретной реализации и подменять её под ту, которая подходит для текущих настроек моделирования.

Восприятие пространства агентом может быть реализовано двумя различными способами. Как направленное или ненаправленное.

В случае направленного передвижение агента будет зависеть от его ориентации в пространстве, в случае ненаправленного от команд (рис. 33).

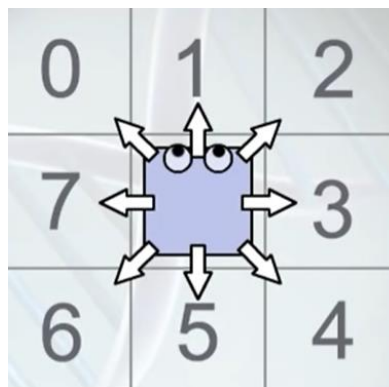


Рисунок 33 – Ненаправленное восприятие пространства агентом

### 3.1.2 Реализация кода-генома и обработки команд агента

Код-геном реализуется классической структурой данных «массив», где в качестве каждого элемента массива представлен код одной из команд.

Обработка команд агента представляет из себя проход по элементам массива, где каждый код команды сопоставляется с набором возможным команд. Сопоставление кода в ячейке генома с набором исполняемых функций происходит посредством оператора switch-case.

### 3.1.3 Реализация пространства моделирования

В зависимости от настроек сеанса пространство моделирования может быть представлено плоским прямоугольным полем, гексагональным полем, или какой-либо другой плоской сеткой, которая может быть представлена ненаправленным графом (пример на рисунке 34).

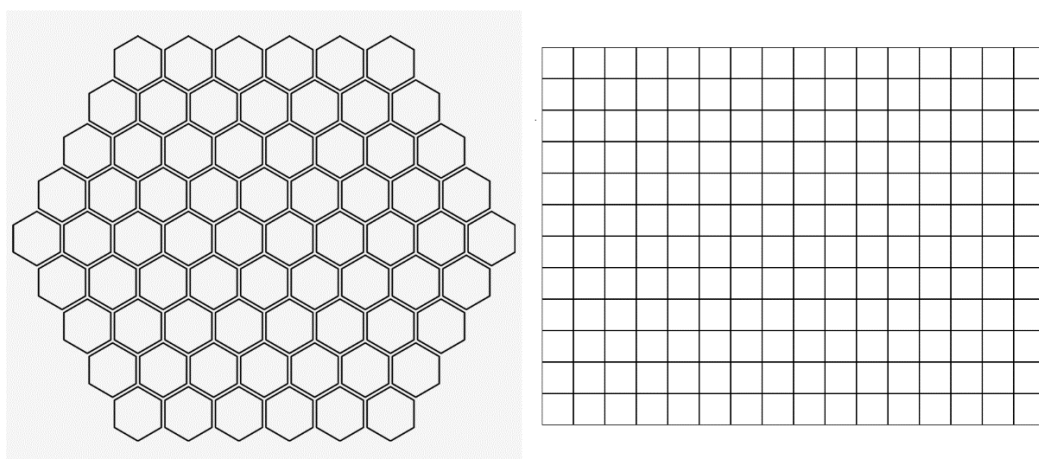


Рисунок 34 – Примеры полей для проведения моделирований

### 3.1.4 Реализация сеанса моделирования

Сеанс моделирования реализуется отдельным асинхронным процессом. При запросе пользователя на актуальную информацию по процессу моделирования backed-server Golang обращается за актуальной информацией к асинхронному процессу, после чего полученные данные отправляет клиенту.

Аналогично происходит изменение параметров моделирование, остановка и возобновление симуляции.

Отправка данных в БД осуществляется самим асинхронным процессом без участия основного backend сервера Golang.

## 3.2 Создание хранилища данных

Для хранения данных используется реляционная БД PostgreSQL. Модель данных представляет из себя совокупность из 10 таблиц (рис. 35)

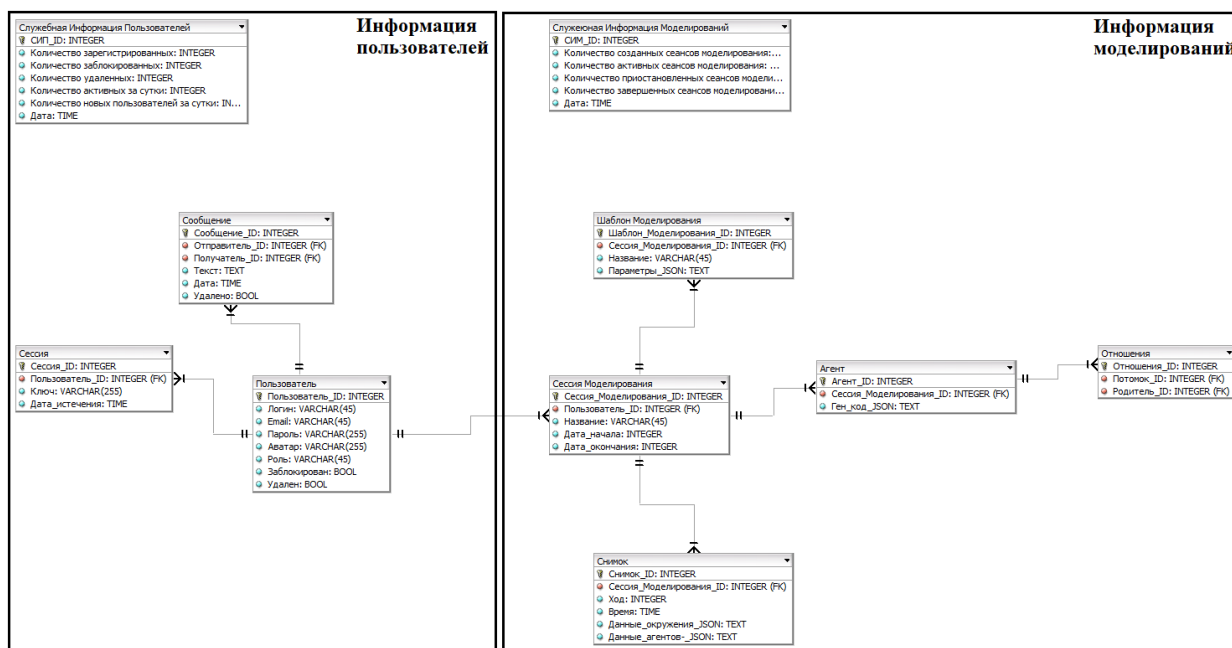


Рисунок 35 – Схема базы данных

На схеме базы данных таблицы можно разделить на две совокупности. Пользовательская информация – таблицы, которые содержат данные о пользователях и их сессиях. Информация моделирований – таблицы, которые содержат информацию о сеансах моделирования.

Также в каждой совокупности таблиц представлена служебная таблица без явных связей с другими таблицами. Они необходимы для хранения ежедневной статистики.

Подробная информация о функциональном назначении каждой таблицы представлена в таблице 7.

Таблица 7

Название таблицы	Тип	Назначение
Пользователь	Обычный	Хранение пользователей и их данных
Сессия	Обычный	Хранение сессий пользователей
Сообщение	Обычный	Хранения переписки пользователей
Сессия Моделирования	Обычный	Связующая информация для сессий моделирования
Шаблон Моделирования	Обычный	Хранения шаблонов для настройки сеансов моделирования
Снимок	Обычный	Хранение информации о моделировании в момент времени
Агент	Обычный	Хранение информации об агента
Отношения	Обычный	Хранение информации о родственных связях между агентами
Служебная Информация Пользователей	Служебный	Хранение служебной информации о пользователях
Служебная Информация Моделирований	Служебный	Хранение служебной информации о сеансах моделирования

### **3.3 Обеспечение информационной безопасности**

#### **3.3.1 Построение модели нарушителя**

Т.к. в ходе выявления потенциальных угроз было выяснено, что большая часть из них относится к воздействию на систему из вне, то было принято решение построить модель для такого нарушителя.

С учётом особенностей разрабатываемой системы, предполагается, что нарушителем может быть одиночка или небольшая группа авантюристов. Модели нарушителя такие, как «организация-конкурент» и «правительственная структура» не будут рассматриваться, т.к. разрабатываемая автоматизированная система не является коммерческим продуктом, а также не имеет под собой никакой политической подоплеки.

Модели нарушителей «Одиночка» и «Группа злоумышленников» представлены в таблице 8.

На основании построенных моделей нарушителей можно предугадать основные векторы атак, на основании чего разработать методы защиты системы от попыток нарушить конфиденциальность информации или целостность самой системы.

Для устранения выявленных уязвимостей предлагается предпринять ряд мер. Защита должна быть обеспечена на всех доступных уровнях в рамках разумного. Основная часть мер должна быть применена на тех элементах программной системы, на которые в первую очередь поступает на обработку пользовательский запрос.

Таблица 8 – Модели нарушителей

<b>Характеристика</b>	<b>Одиночка</b>	<b>Группа злоумышленников</b>
Вычислительная мощность технических средств	Персональный компьютер	ЛВС
Доступ к Интернету, тип каналов доступа	Модем или выделенная линия	Использование выделенных каналов с высокой пропускной способностью
Финансовые возможности	Сильно ограниченные	Ограниченные
Уровень знаний в области ИТ	Невысокий или средний	Средний или высокий. Множество компетенций
Используемые технологии	Готовые программы, самописные скрипты, использование популярных уязвимостей	Исследование атакуемой системы, поиск уязвимостей, изготовление вредоносных программ, внедрение троянов и вирусов
Знание о системе защиты атакуемой системы	Отсутствие знаний об особенностях системы	Могут предпринимать значительные усилия по получения информации о работе системы на протяжении длительного временного периода
Преследуемые цели	Эксперимент, получение реального опыта взлома, хулиганство, особые личные мотивы	Вымогательство, внесение искажений в работу системы, похищение личных данных пользователей
Характер действий	Скрытый	Скрытый
Глубина проникновения	Первое успешное воздействие или до обнаружения	До достижения поставленных целей, до обнаружения

### **3.3.2 Обеспечение защиты от нарушения доступности и целостности системы в единой точке доступа к системе**

На этапе проксирования запроса от пользователя на сервере nginx можно предпринять некоторое количество мер по защите системы.

#### **3.3.2.1 Блокировка DDoS источника по IP-адресу**

Позволяет обеспечить частичную защиту от DDoS спама, однако из-за NAT не может быть использоваться постоянно. Для надежности защита должна обеспечиваться провайдером, т.к. он обладает большими ресурсами и знаниями о конфигурации сети Интернет для блокировки нарушителей.

#### **3.3.2.2 Ограничение количества запросов от пользователей в единицу времени**

Данное ограничение позволяет значительно сэкономить ресурсы в случае, если некоторая группа юзеров совершает аномально большое количество запросов, в котором нет необходимости при штатном использовании сервисов, предоставляемых веб-приложением.

#### **3.3.2.3 Отключение медленных пользователей**

Отключение медленных пользователей позволяет защититься от атак, которые используют медленные соединения для утилизации большого процента ресурсов на сервисе, на который совершается атака.

### **3.3.3 Обеспечение защиты в приложении frontend-сервера**

Для защиты пользователя от некорректного поведения на стороне веб-приложения в браузере необходимо обеспечить защиту от наиболее частых типов атак со стороны внешнего нарушителя.

#### **3.3.3.1 Защита от XSS**

Для защиты от XSS (внедрение вредоносного кода) обеспечивается сериализация данных, которая призвана из вводимой информации удалять все типы символов, которые могут быть восприняты как исполняемый код при интерпретации браузером.



### **3.3.3.2 Защита от CSRF**

Запрос CSRF-токена – один из самых популярных и надежных на данный момент способов защиты от CSRF-атак. Обеспечивает защиту от несанкционированных действий при междоменном взаимодействии.

### **3.3.4 Обеспечение защиты на backend-сервере**

На данном программном компоненте стоит в первую очередь обратить внимание на защиту пользовательских данных, т.к. данный компонент напрямую взаимодействует с базами данных.

#### **3.3.4.1 Защита от XSS**

Использование экранированных входных\выходных данных. Применение встроенных функций для очистки кода от вредоносных скриптов.

#### **3.3.4.2 Использование белых списков для доверенных хостов**

Для взаимодействия со всеми компонентами программной системы должен использоваться подход «белых списков», когда разрешено взаимодействие только с теми хостами, которые были явно прописаны в списке разрешенных. В этом случае какое-либо взаимодействие с «чужими хостами» будет запрещено.

Это также обеспечивает дополнительную защиту, когда проводится попытка нарушения работоспособности системы изнутри.

### **3.3.5 Обеспечение защиты данных в БД**

Осуществляется непосредственно средствами реляционной базы данных PostgreSQL.

#### **3.3.5.1 Вход по паролю**

Для взаимодействия с базой данных требуется пройти идентификацию и аутентификацию.

#### **3.3.5.2 Разграничение прав доступа**

Перед совершением какого-либо действия проводится авторизация пользователя.

### **3.3.5.3 Защита полей таблиц**

Можно явно заблокировать для пользователей БД операции по удалению и модификации данных, без предварительного резервного копирования изменяемых данных.

### **3.3.5.4 Шифрования данных**

Наиболее ценные данные в базе должны храниться в зашифрованном виде. Например, пароли.

## Заключение

В соответствии с техническим заданием была разработана программная система моделирования искусственной жизни с использованием цифровых автоматов в качестве выпускной квалификационной работы.

Разработанная система представляет из себя веб-приложение, доступное широкому кругу пользователей в глобальной сети Интернет. Веб-приложение позволяет моделировать самоорганизующиеся системы, наподобие реальных биологических.

При разработке системы были предусмотрена возможность расширения системы, вследствие чего вся система была спроектирована как совокупность небольших модулей, а горизонтальные связи между однородными компонентами сведены к минимуму.

Для упрощения интеграции текущих и последующих модулей системы для обмена данными между компонентами системы использовались стандартные сетевые протоколы HTTP, TCP и UDP.

Для упрощения администрирования системы и повышения отказоустойчивости была использована система оркестровки контейнерами Kubernetes совместно с технологией контейнеризации Docker.

Разработанная система имеет дальнейшие перспективы для развития и модификации. На полезности системы сильно отразится добавление новых типов моделирования, социального взаимодействия между пользователями системы, возможность импорта данных моделирования в человеко-читальные форматы (XML, YAML), улучшения графической звуковой части сеанса моделирования (дополнительные фильтры, плавное перемещение агентов, звуки оповещения, 3D модели).

## Список использованных источников

1. Гладков Л.А., Курейчик В.В., Курейчик В.М. Генетические алгоритмы / Под ред. В. М. Курейчика. — 2-е изд., исправл. и доп. — М.: ФИЗМАТЛИТ, 2010. — 368 с.
2. Емельянов В.В., Курейчик В.В., Курейчик В.М. Теория и практика эволюционного моделирования. – М.: ФИЗМАТЛИТ, 2003. 432 с.
3. Штуца И.М. Модели и алгоритмы принятия решений на основе генетического поиска // Штуца И.М.; – Москва, 2008. - №1. – 19 с.
4. Гуренко В.В. Введение в теорию автоматов [Электронный ресурс] : электронное учебное издание: учебное пособие по дисциплинам "Теория автоматов", "Прикладная теория цифровых автоматов" / В. В. Гуренко ; Московский гос. технический ун-т им. Н. Э. Баумана, Фак. "Информатика и системы управления", Каф. "Компьютерные системы и сети". - Москва: МГТУ им. Н. Э. Баумана, 2013. – 62 с.
5. Скобцов Ю. А. Основы эволюционных вычислений: учебное пособие / Ю. А. Скобцов ; М-во образования и науки Украины, Донецкий нац. технический ун-т. - Донецк : ДонНТУ, 2008. - 326 с
6. Введение в ГА и Генетическое Программирование [Электронный ресурс] // Алголист: алгоритмы и методы вычислений. URL: <http://www.algolist.ru> (дата обращения 20.10.2019)
7. Атлас простейших клеточных автоматов Стивена Вольфрама [Электронный ресурс] // Атлас Вольфрама. URL: <http://atlas.wolfram.com>
8. Голубин А.В. Определение параметров генетического алгоритма для оптимизации многопараметрических функций // Прогрессивные технологии, конструкции и системы в приборо- и машиностроении. / Сб. статей. - М.: Изд-во МГТУ им. Н.Э. Баумана, 2001. - С. 65-67.
9. Простейшие клеточные автоматы и их применение [Электронный ресурс] // Хабр. URL: <https://habr.com/ru/post/273393/> (дата обращения 11.11.2019)

10. Эволюционирующие клеточные автоматы [Электронный ресурс] // Хабр. URL: <https://habr.com/ru/post/455958/> (дата обращения 28.09.2019)
11. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. — СПб.: Питер, 2018. — 640 с.
12. Марко Лукша. Kubernetes в действии / пер. с англ. А. В. Логунов. — М.: ДМК Пресс, 2019. — 672 с.
13. Сайфан Джиджи. Осваиваем Kubernetes. Оркестрация контейнерных архитектур. — СПб.: Питер, 2019. — 400 с.
14. Раджпут Динеш. Spring. Все паттерны проектирования. — СПб.: Питер, 2019. — 320 с.
15. Модели жизненного цикла программного обеспечения [Электронный ресурс] // Хабр. URL: <https://habr.com/ru/post/111674/> (дата обращения 21.02.2020).
16. Иванова Г.С. Технология программирования. М. : Изд-во КНОРУС, 2011.
17. Тестирование. Фундаментальная теория [Электронный ресурс] // Хабр. URL: <https://habr.com/ru/post/279535/> (дата обращения 22.02.2020).
18. Информационная безопасность: Учебное пособие. Часть 1. — СПб.: Изд-во СПбГУЭФ, 2010. — 96с.

## **Приложение А – Техническое задание**

**Листов 11**

## **Приложение Б – Руководство пользователя**

**Листов ??**

## **Приложение В – Фрагменты исходного кода системы**

**Листов ??**



## **Приложение В – Копии листов графической части**

**Листов ??**

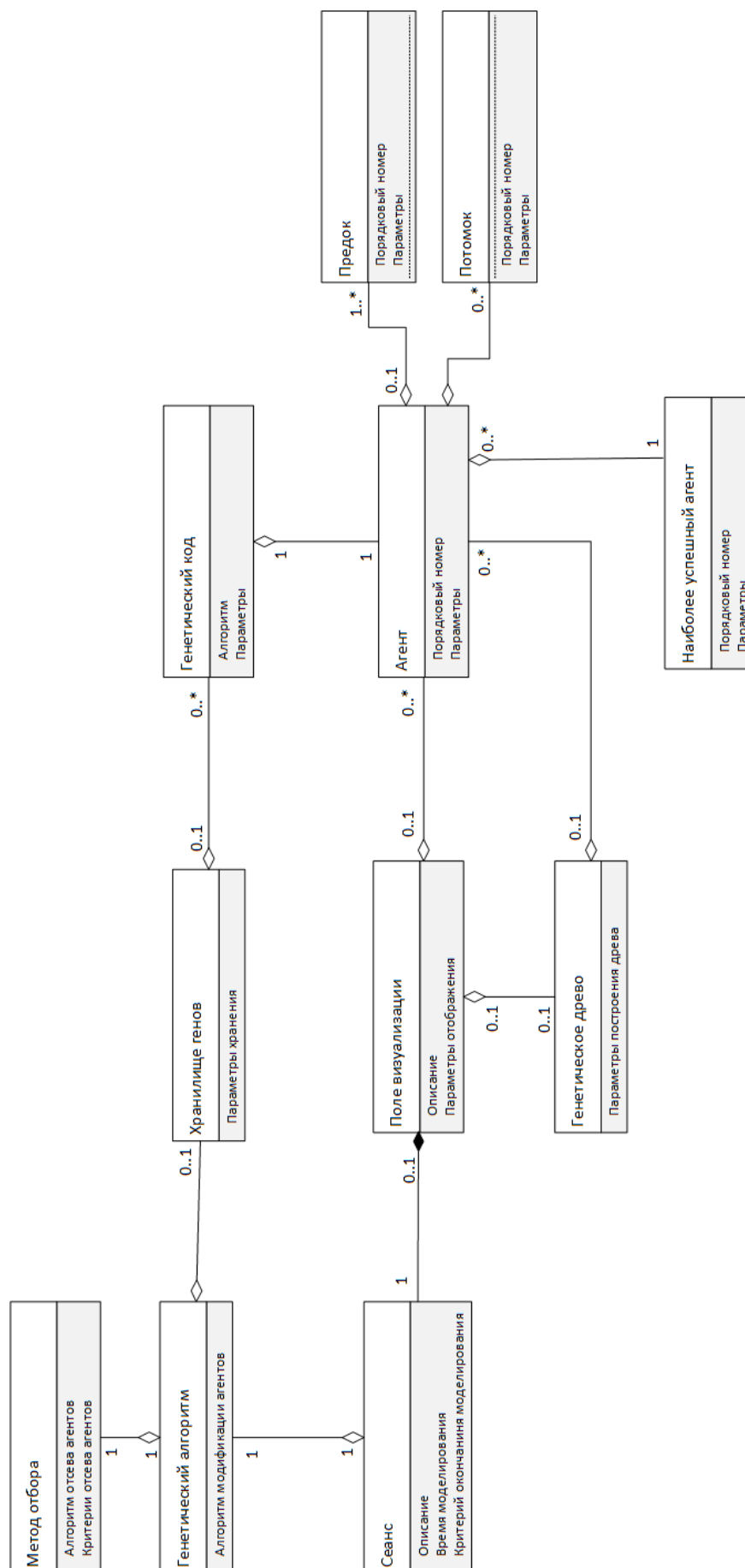


Рисунок 36 – Концептуальная модель предметной области

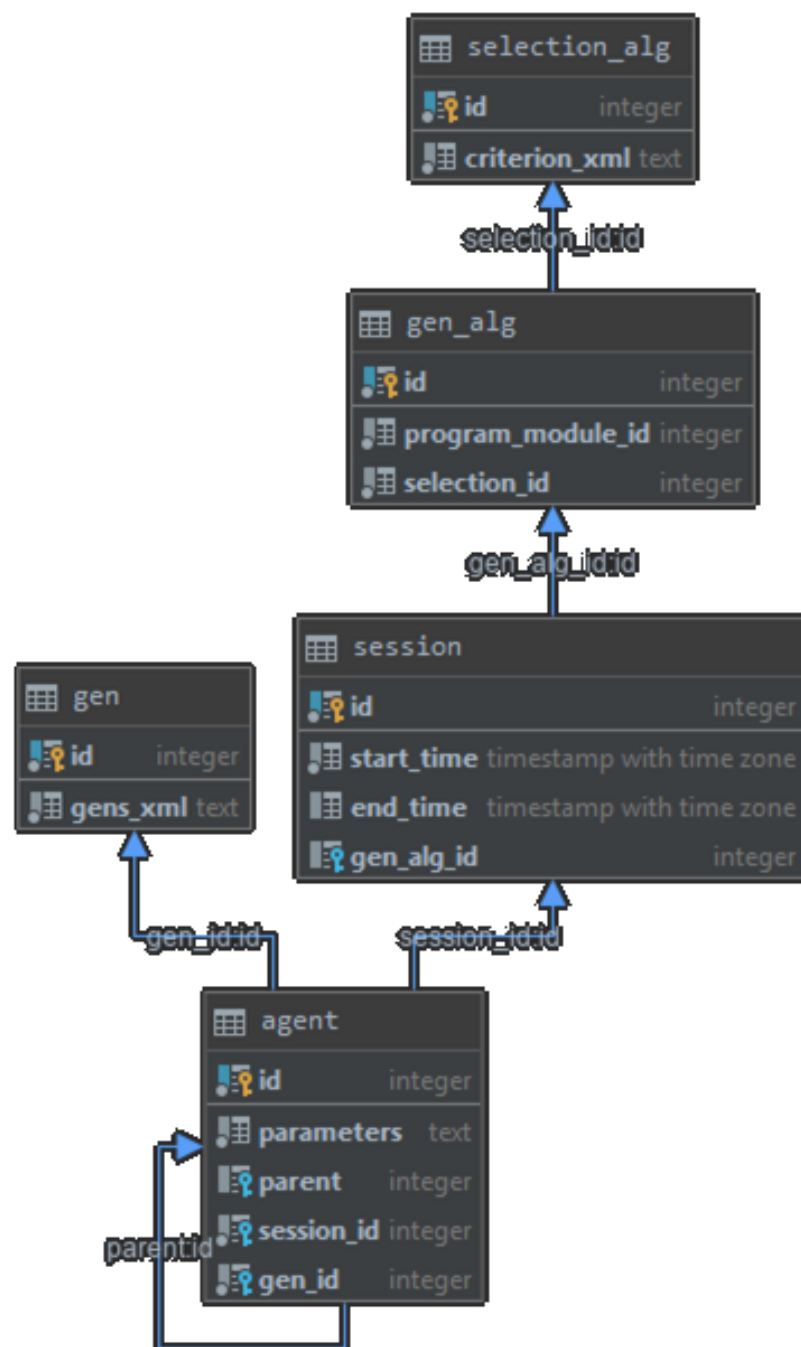


Рисунок 37 – Прототип схемы базы данных

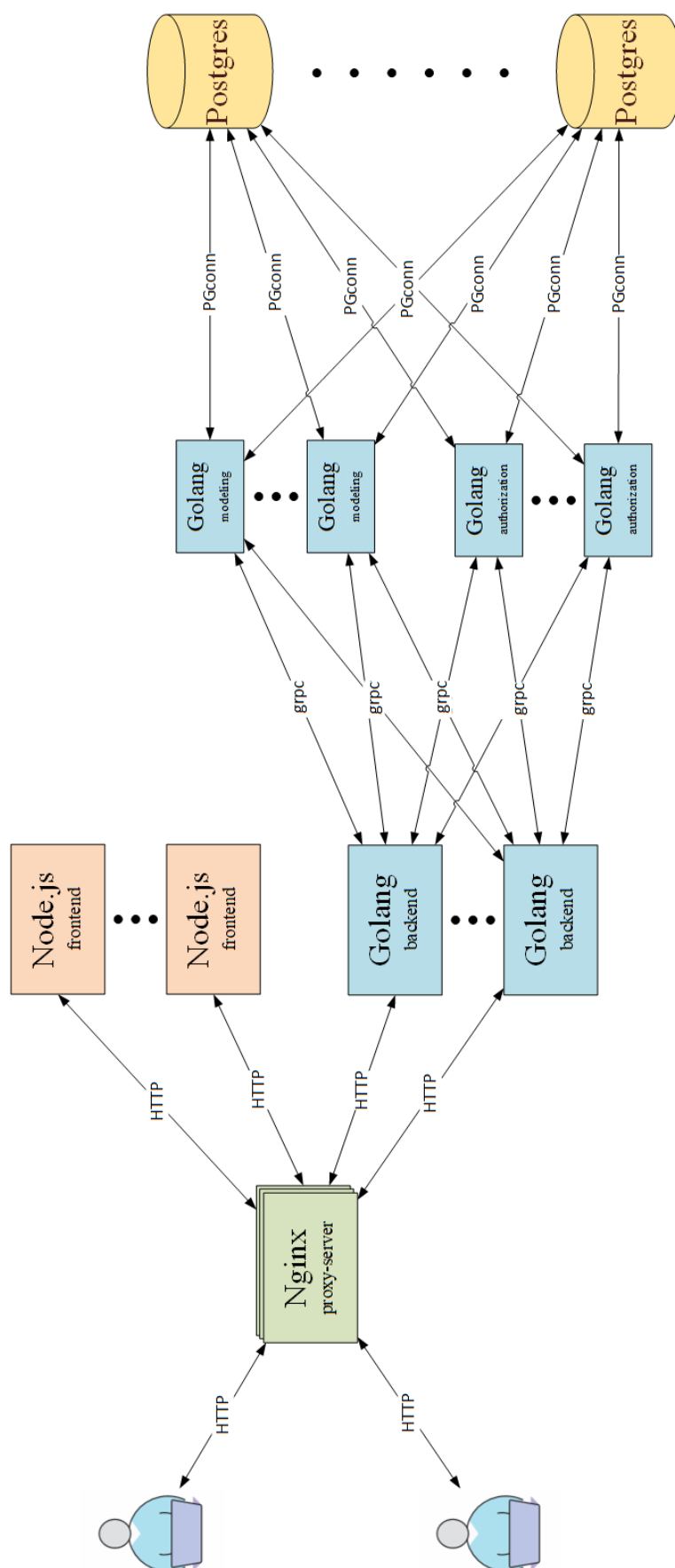


Рисунок 38 – Архитектура программного обеспечения

# Кластер Kubernetes

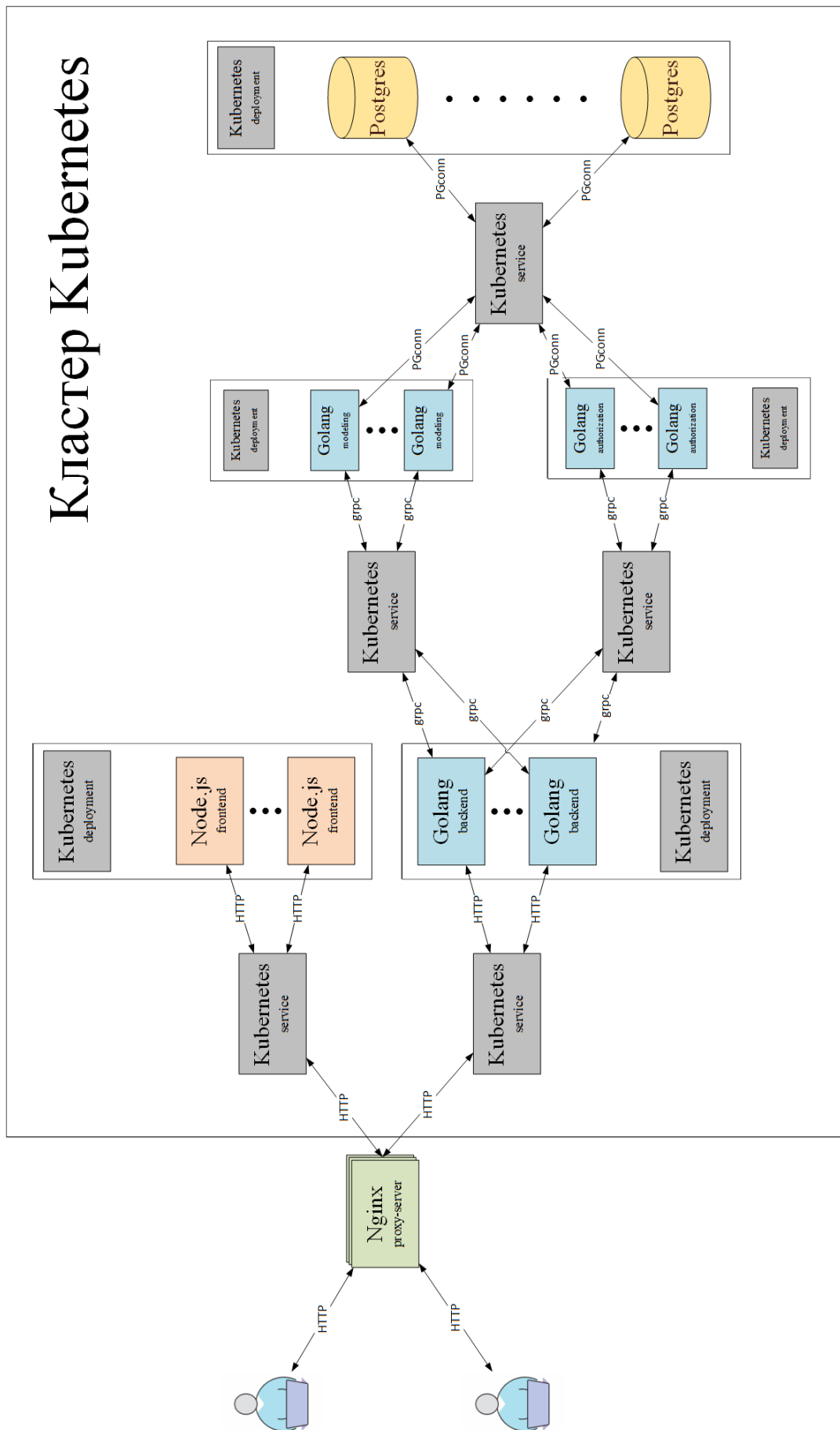


Рисунок 39 – Разрабатываемая программная система в кластере Kubernetes

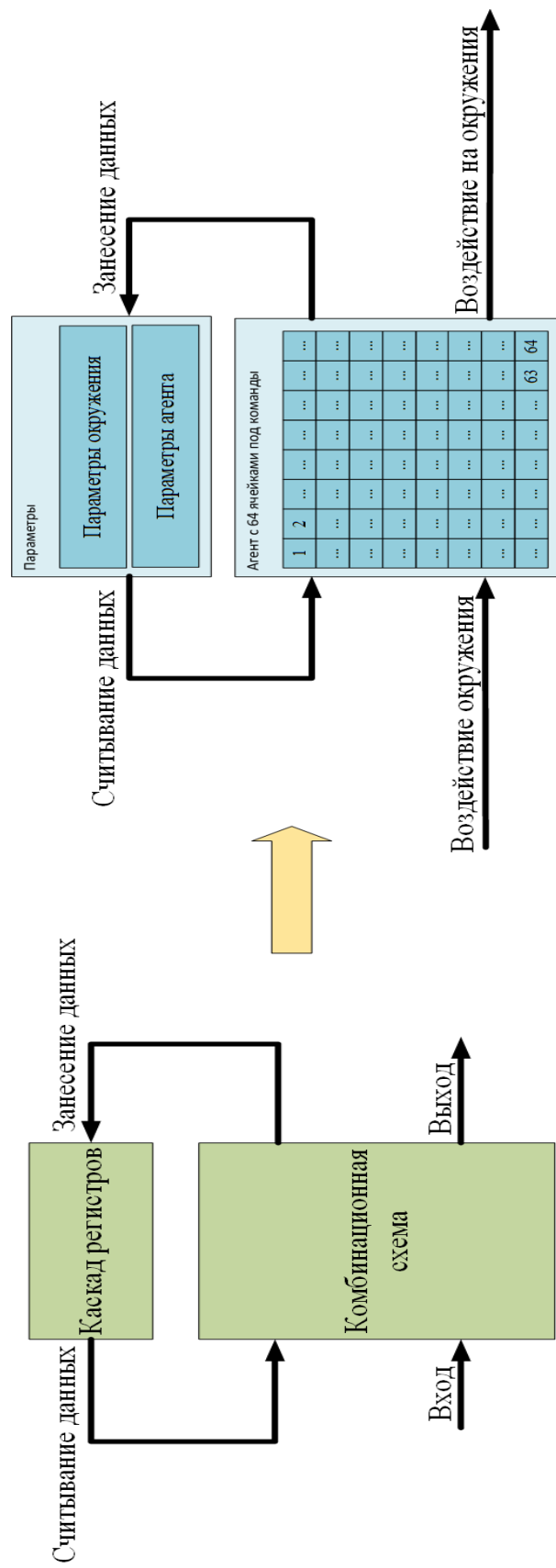


Рисунок 40 – Модель абстрактного автомата и агента

