

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Факультет Кафедра

вычислительной техники

«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»
Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

РЕАЛИЗАЦИЯ АГЕНТНО-БАЗИРОВАННЫХ
МЕТАКОМПЬЮТЕРНЫХ СИСТЕМ С
ИСПОЛЬЗОВАНИЕМ НЕЙРОСЕТЕВЫХ
ТЕХНОЛОГИЙ

Студент

Свищев 27.05.2024
(подпись, дата)

Свищев Данил Сергеевич
(ФИО полностью)

Руководитель

Зинкин 27.05.2024
(подпись, дата)

Зинкин С.А.
(фамилия, инициалы)

Нормоконтролёр

Бычков 27.05.2024
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Рецензент

Мартышкин

Мартышкин А.И., к.т.н., зав. кафедрой Программного обеспечения ПензГУ
(подпись, дата) (фамилия, инициалы) (должность, место работы)

Работа допущена к защите (протокол зас. кафедры от 27.05.24 № 11)

Заведующий кафедрой

М.А. Митрохин
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой Отлично (протокол заседания ГЭК от 06.06.24 № 12)

Секретарь ГЭК

Е.И. Гурин
(подпись)

Е.И. Гурин
(фамилия, инициалы)

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

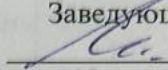
Кафедра

вычислительной техники

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

 М.А. Митрохин

«13» сентября 2022 г.

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
МАГИСТРА**

1. Студент Свищев Данил Сергеевич гр. 22ВВИм1 факультета
вычислительной техники направления 09.04.01.

2. Тема работы «Реализация агентно-базированных метакомпьютерных
систем с использованием нейросетевых технологий»

Тема утверждена приказом ПГУ №765/о от "13" сентября 2022г.

3. Руководитель магистерской диссертации д.т.н., профессор кафедры ВТ
ПГУ Зинкин С.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: реализация метакомпьютерной системы на
основе пиринговой сети и мультиагентной платформы для распараллеливания
обработки данных и выполнения вычислений с использованием искусственного
интеллекта.

Разработать агентно-базированную метакомпьютерную систему с
использованием нейросетевых технологий для выполнения параллельного
процессинга в сети.

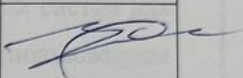
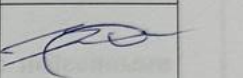
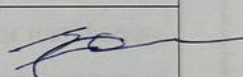
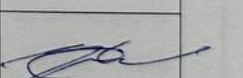
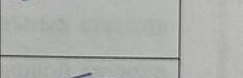
5. Перечень подлежащих разработке вопросов:

1. Анализ предметной области метакомпьютерных систем и
возможности их использования для параллельной обработки

данных.

2. Реализация метакомпьютерной системы, базирующейся на интеллектуальных агентах и архитектуре пиринговой сети.
3. Отладка и тестирование агентно-базированной метакомпьютерной системы.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области метакомпьютерных систем	10%	27.09.2022-15.11.2022	
2.	Анализ предметной области больших языковых моделей	10%	16.11.2022-22.03.2023	
3.	Выбор инструментов разработки	5%	23.03.2023-14.06.2023	
4.	Проектирование структуры разрабатываемой системы	15%	15.06.2023-10.09.2023	
5.	Разработка всех компонентов метакомпьютерной системы	40%	11.09.2023-05.01.2024	
6.	Оптимизация работы системы	10%	06.01.2024-17.04.2024	
7.	Отладка и тестирование агентно-базированной метакомпьютерной системы	10%	18.04.2024-09.05.2024	
8.	Оформление выпускной квалификационной работы	5%	10.05.2024-28.05.2024	

Дата выдачи задания "13" сентября 2022г.

Руководитель магистерской диссертации

13.09.2022 Зинкин С.А.

(подпись, дата)

Задание к исполнению принял студент

13.09.2022 Свищев Д.С.

(подпись, дата)

Работу к защите допустить

Декан факультета



04.06.2024

Фионова Л.Р.

(подпись, дата)

РЕФЕРАТ

Магистерская работа содержит 80 листов, 27 рисунков, 4 источника

МЕТАКОМПЬЮТЕРНЫЕ СИСТЕМЫ, МАСШТАБИРУЕМЫЕ ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ, РАСПРЕДЕЛЕННЫЕ ВЫЧИСЛЕНИЯ, ОБРАБОТКА ДАННЫХ, ДИНАМИЧЕСКАЯ БАЛАНСИРОВКА, НЕЙРОСЕТЕВЫЕ ТЕХНОЛОГИИ, БОЛЬШИЕ ЯЗЫКОВЫЕ МОДЕЛИ, МУЛЬТИАГЕНТНЫЕ СИСТЕМЫ, ПИРИНГОВАЯ СЕТЬ

Целью работы является изучение архитектур и возможностей реализации мультиагентных масштабируемых интеллектуальных систем для выполнения распределенных вычислений в сети с использованием нейросетевых технологий.

Проведен анализ различных фреймворков для работы с нейронными технологиями и мультиагентными системами на языке программирования Java, методов проектирования распределенных интеллектуальных систем, обучения нейронных сетей, а также их подключения к агентам платформы, реализаций параллельных вычислений и параллельной обработки данных на основе агентно-ориентированной платформы, изучены возможности динамического конфигурирования и управления узлами в сети.

В результате проделанной работы, были изучены библиотеки для работы с нейронными сетями, большие языковые модели, возможности интеграции нейросетевых технологий с платформой Java, а также была разработана метакомпьютерная система на основе интеллектуальных агентов и архитектуре пиринговой сети с использованием большой языковой модели для реализации распределенного процессинга в сети.

ПГУ 1.090401.21.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата
Разраб.		Свищев Д.С.		27.05.2024
Провер.		Зинкин С.А.		27.05.2024
Н. Контр		Бычков А.С.		27.05.2024
Утверд.		Митрохин М.А.		27.05.2024

реализация агентно-базированных
метакомпьютерных систем с
использованием нейросетевых
технологий

Лит.	Лист	Листов
	2	80
ФВТ, каф. ВТ		
гр. 22ВВим1		

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	6
ВВЕДЕНИЕ	7
2 Анализ темы	10
2.1 Актуальность выбранной темы	10
2.2 Обзор технологии и инструментов	12
2.2.1 Технологии и инструменты пиринговых сетей	12
2.2.2 Технологии и инструменты мультиагентных систем	14
2.2.3 Технологии и инструменты систем искусственного интеллекта ...	16
2.3 Направление исследования	18
3 Постановка задачи	22
4 Выбор инструментов разработки	24
4.1 Мультиагентная платформа JADE	25
4.2 Библиотеки искусственного интеллекта	29
4.2.1 Neuroph	30
4.2.2 Deeplearning4j	32
5 Описание архитектуры	34
5.1 Архитектура мультиагентной системы	34
5.2 Архитектура нейронной сети	35
5.3 Архитектура метакомпьютера	38
6 Описание алгоритма	46
6.1 Конфигурация агента	47
6.2 Поиск агентов	53
6.3 Балансировка и масштабируемость	56
6.4 Взаимодействие агентов	59
6.5 Взаимодействие с нейронной сетью	64
6.6 Обучение модели	67
ЗАКЛЮЧЕНИЕ	69
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	70
ПРИЛОЖЕНИЕ А Листинг программы	71
ПРИЛОЖЕНИЕ Б Презентация	79

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В данной магистерской диссертации применяются следующие сокращения и обозначения

ИИ - искусственный интеллект

БД - база данных

СУБД - система управления базами данных

AI (artificial intelligence) - искусственный интеллект

API (Application programming interface) - контракт, предоставляемый системой

GPT (Generative Pre-trained Transformer) – генеративный, предобученный трансформер

IDE (Integrated development environment) – интегрированная среда разработки

BERT (Bidirectional Encoder Representations from Transformers) – языковая модель, представленная на архитектуре трансформера и предназначенная для выполнения функций предобучения языковых представлений, целью которых является последующее применение в задачах обработки естественного языка

GPT (Generative Pre-trained Transformer) – это тип нейронных языковых моделей, которые обучаются на больших наборах текстовых данных, чтобы генерировать текст, схожий с человеческим

T5 (Text-to-Text Transfer Transformer) – серия крупных языковых моделей

ВВЕДЕНИЕ

Различные приложения интернета, особое внимание уделяется промышленному интернету вещей, реализуют генерацию большого объема данных, чем когда-либо прежде. Стоит отметить, что ежегодно объем этих данных увеличивается и их количество становится всё больше и больше. В различных промышленных ситуациях с применением устройств с Интернетом вещей, в частности на предприятиях с сильно распределенной системой, регулярная отправка необработанных данных в ЦОД (центр обработки данных) или на основной сервер может оказаться невозможной.

Практика переноса принятия решений в реальное время на местах способствует ряду оптимизаций, таких как: уменьшения задержки, снижения затрат на передачу и хранение данных, снижение нагрузки на сеть предприятия (отсутствует необходимость передачи огромного количества данных на расстоянии). Для этого используются технологии AI – искусственного интеллекта и ML – машинного обучения.

Искусственным интеллектом вещей называю приложения, которые предоставляют мощности ИИ в инфраструктурах IoT. Не смотря на то, что модели ИИ также обучаются, сбор данных и обучение применению логических выводов возможно выполнять непосредственно на предприятии за счет организации окружения и развертывания моделей ИИ на периферийных компьютерах.

Характерной особенностью прикладных задач, возникающих в рамках крупномасштабных проектов, является потребность в использовании высокопроизводительных систем, вычислительных возможностей которых в отдельности не хватает для эффективного решения таких задач.

В настоящее время происходит активная разработка и внедрение распределенных вычислительных систем, основанных на использовании большого числа компьютеров, объединённых в единую сеть. Одним из перспективных направлений развития таких систем являются метакomпьютеры, представляющие собой множество взаимодействующих

между собой компьютеров, объединённых в единую вычислительную систему.

Под реализаций метакомпьютера понимается интеграция высокопроизводительных систем и высокоресурсных машин в единую распределенную вычислительную среду. Данный подход обеспечивает ряд преимуществ перед традиционными централизованными системами, предоставляя возможность неограниченной горизонтальной масштабируемости и отказоустойчивости, возможности параллельной обработки больших объёмов данных и условия для реализации распределенных алгоритмов, решения больших и сложно структурированных задач, за счет совокупной совокупной производительности среды, ранее не доступной пользователям локальных вычислительных систем.

Одной из ключевых технологий, используемых при создании метакомпьютеров, являются пиринговые одноранговые сети. Эти сети позволяют эффективно обмениваться данными между компьютерами, минуя центральные сервера, что делает их удобными для использования в распределенных вычислительных системах.

В настоящее время наблюдается бурный рост интереса к различным системам искусственного интеллекта (ИИ). В связи с этим меняются и сами принципы человеко-машинного взаимодействия. С определенной долей уверенности можно утверждать, что происходит переход на более высокий уровень абстракции — взаимодействие человека и ИИ. Современным решением по реализации и управлению вычислений на основе или с использованием ИИ в пиринговой сети является применение мультиагентных систем (МАС). МАС обеспечивают решения проблем, которые сложно или невозможно, за счет ресурсных ограничений, решить с помощью одного агента или монолитной системы.

Данные системы представляют собой одну из разновидностей систем искусственного интеллекта, образованную несколькими взаимодействующими интеллектуальными агентами. Интеллектуальные

агенты, в свою очередь, — это агенты (чаще всего под агентом подразумевается некая компьютерная программа), которым присущи следующие признаки:

- целеустремленность,
- обучаемость,
- социальность,
- независимость.

Целью данной работы является исследование применения пиринговых одноранговых сетей для создания метакомпьютера на основе мультиагентных интеллектуальных систем с использованием средств искусственного интеллекта.

2 Анализ темы

2.1 Актуальность выбранной темы

Пиринговые сети построенные на базе мультиагентные интеллектуальных систем и имеют широкий спектр применений, что обосновывает их актуальность.

Пиринговые сети представляют собой децентрализованную систему обмена информацией между равноправными участниками. Примерами применения пиринговых сетей является, например, обмен файлами. Пиринговые сети позволяют пользователям обмениваться файлами напрямую друг с другом, минуя центральные серверы. Это позволяет ускорить процесс обмена файлами и снизить нагрузку на серверы. Также можно выделить облачные технологии. Пиринговые сети могут использоваться для создания децентрализованных облачных сервисов, где каждый пользователь может выступать как клиент, так и сервер. Стоит обратить внимание и на блокчейн. Пиринговые сети лежат в основе технологии блокчейн, которая используется для создания безопасных и прозрачных систем учета и передачи информации,

Мультиагентные системы, в свою очередь, представляют собой набор агентов, которые взаимодействуют друг с другом для достижения поставленной цели, причем каждый агент действует независимо, но вместе они образуют коллективный разум, способный решать сложные задачи. Примерами применения мультиагентных систем являются:

- управление роботами. МАС применяются для координации действий группы роботов, которые выполняющих общую задачу, примером выступает управление группой дронов, которые осуществляют поиск и спасение людей в аварийных ситуациях,

- оптимизация маршрутов. МАС предоставляют возможности для оптимизации маршрутов транспорта, например, для доставки товаров или пассажирских перевозок,

- биотехнологии. МАС используются для моделирования биологических процессов, в которых относят рост клеток или реакцию иммунной системы организма,

- комплексные распределенные вычислительные вычисления. МАС обеспечивают реализации систем, которые являются неограниченными по ресурсам и используются для выполнения вычислений, распределенных по сети.

Немаловажную роль при создании метакomпьютерных систем играет искусственный интеллект. Это объясняется тем, что системы, основанные на принципах работы с искусственным интеллектом, способны обучаться и адаптироваться к изменяющимся условиям, они могут использовать алгоритмы машинного обучения для принятия решений и оптимизации своих действий, например автономные транспортные средства. Системы, которые оснащены ИИ, используются для управления автономными автомобилями, они осуществляют обработку информации от различных датчиков и принимают решения о дальнейших действиях, например о направлении движения, скорости или маневрировании. Выделяют также робототехнику. МАС с ИИ используются для управления объединениями роботов, которые могут координировать действия и оптимизировать их маршруты. Немаловажной составляющей является интернет вещей. МАС с ИИ реализуются для управления устройствами интернета вещей, в частности интернета промышленных вещей такие системы собирают данные от различных устройств, анализируют их и формируют выводы для принятия решений.

Искусственный интеллект играет ключевую роль в развитии мультиагентных систем, делая их более интеллектуальными, адаптивными и эффективными, а совместное использование пиринговых сетей и мультиагентных систем, построенных на основе интеллектуальных агентов, представляет собой уникальный подход к организации метакomпьютерной системы, позволяющей реализовать любую по сложности и уникальности

функцию, которая невозможна для выполнения на одном компьютере, что обеспечивает высокую степень масштабируемости и гибкости за счет распределения вычислительных ресурсов между множеством узлов сети. Данный подход повышает надежность системы, так как отказ одного узла не приводит к остановке всей системы, а также делает систему интеллектуальной, адаптивной и эффективной.

2.2 Обзор технологии и инструментов

2.2.1 Технологии и инструменты пиринговых сетей

Для реализации пиринговых сетей используются различные языки программирования, технологии и архитектуры. Основными языками для реализации данной технологии являются:

- C++. Язык программирования, который используется для создания высокопроизводительных приложений, требующих высокой скорости работы и обработки больших объемов данных. К достоинствам данного языка относят высокую производительность, хорошую поддержку многопоточности, а к недостаткам сложность в разработке и поддержке и потенциальные проблемы с безопасностью,
- Python. Данный язык используется для быстрой разработки прототипов и скриптовых приложений, что обеспечивает простоту изучения и использования, а также большое количество библиотек и развитое сообщество, но данный язык поддерживает относительно низкую производительность по сравнению с C++,
- Java. Один из популярных языков, который зарекомендовался себя для создания кросс-платформенных приложений благодаря легкости развертывания, хорошей поддержке сообщества, надежности и мощности,
- Go. Язык, который используется для создания высокопроизводительных приложений с акцентом на простоту и безопасность, включает встроенную многопоточность, но ограничивается экосистемой библиотек по сравнению с другими языками.

Пиринговые же сети в своей основе используют ряд технологий, описанных ниже.

BitTorrent – это протокол для пиринговых сетей, позволяющий участникам скачивать файлы с помощью распределенной сети за счет высокой скорости загрузки и снижение нагрузки на источник файла, но содержит потенциальные проблемы с безопасностью и возможное нарушение авторских прав.

DHT представляет собой технологию, позволяющая участникам находить и обмениваться данными без необходимости знать IP-адреса друг друга. К достоинствам данной технологии относится децентрализация, что представляет собой отсутствие необходимости в централизованных серверах. Недостатками же являются возможные проблемы с безопасностью, а также повышенная сложность разработки и последующей поддержки.

P2P-прокси представляет собой технологию, которая скрывает действительные IP-адреса участников и обеспечивает возможности анонимности и безопасности, а также защиту личных данных и предотвращение отслеживания, но также существуют потенциальные проблемы с безопасностью и возможное ограничение функциональности.

Непосредственно архитектура пиринговой сети может быть представлена в виде одноранговой сети или пирингового хостинга, где под одноранговой сетью понимается архитектура, в которой каждый участник одновременно выступает и клиентом, и сервером. Плюсами данной архитектуры выступает децентрализация и распределение нагрузки, но могут быть потенциальные проблемы с безопасностью и сложность координации действий участников. Пиринговый же хостинг представляет собой архитектуру, использующую одноранговую пиринговую сеть для публикации и доступа к веб-страницам. В данном случае в качестве плюсов также выделяют децентрализацию и отсутствие необходимости в централизованных серверах, но определяют потенциальные проблемы с безопасностью и возможное ограничение функциональности.

Выбор конкретного языка программирования, технологии и архитектуры зависит от требований проекта, предпочтений разработчика и специфики пиринговой сети.

2.2.2 Технологии и инструменты мультиагентных систем

Агентные платформы представляют собой среду разработки и существования мультиагентных систем. Существует большое количество разработанных реализаций агентных платформ, характеризующиеся уникальными возможностями, а также достоинствами и недостатками. На рисунке 1 представлен список из доступных агентских платформ (Тд1/Тд2 – время доставки сообщения в мс между агентами на одном хосте/на двух хостах).

Название	Лицензия	Стандарты	Поддержка моб. у-в	Тд1/Тд2**	Поддержка разраб.	Реализация/ Особенности
JADE	LGPL	FIPA2 KQML	JME до MIDP1.0	7,5 / 68,8	Есть	Java;.NET/ plug-ins
A-globe	LGPL	Частично FIPA	Нет	2,8 / 13,9	Есть	Java
IBM ABLE	Comm-eval	Планируется FIPA	Нет	???	Есть	Java
JaCK	Comm	FIPA	JME	21,7 / 17,6	Есть	Java/среда разработки
AgentBuilder	Comm	KQML	Нет	???	Есть	Java/методология и среда разработки
FIPA-OS	LGPL	FIPA1	Нет	607,1 / 96,2	нет с 2001г.	Java
ZEUS	LGPL	FIPA1	Нет	224,8 / 107,6	нет с 2001г.	Java/среда разработки

Рисунок 1 – Агентные платформы

Мультиагентные платформы (МАП) – это программные системы, которые позволяют создавать распределенные приложения, основанные на взаимодействии автономных агентов. Они могут использоваться в различных областях, включая искусственный интеллект, робототехнику, автоматизацию производства и управление сетями.

Для языка программирования Java существует несколько популярных МАП, каждая со своими особенностями и возможностями.

JADE представляет собой одну из самых известных, популярных и применяемых МАП для языке программирования Java. Она предоставляет

удобный интерфейс для создания, управления и взаимодействия между агентами. JADE поддерживает различные протоколы связи и может работать в различных сетевых средах. К её достоинствам относятся мощная поддержка сообщества разработчиков, наличие множества примеров и документации, а также встроенная поддержка многих языков программирования. Среди недостатков можно отметить сложность настройки и конфигурации, особенно для новых пользователей.

Jason – это мультиагентная платформа, которая использует ориентированный на сообщения подход к разработке агентов. Она предоставляет простой и гибкий API для создания агентов и обеспечивает поддержку различных стратегий поведения. Jason хорошо подходит для обучения и исследований в области мультиагентных систем. Его преимущества включают простоту использования и лёгкость интеграции с другими системами, но может быть менее мощным и гибким по сравнению с некоторыми другими МАП.

Repast Symphony – мультиагентная платформа, предназначенная для моделирования и симуляции сложных систем. Она предлагает визуальный интерфейс для создания и запуска моделей, а также включает в себя инструменты для анализа данных и визуализации результатов. Repast Symphony обладает мощными возможностями для параллельных вычислений и может быть полезен при работе с большими и сложными моделями. Данная платформа требует значительных усилий для освоения и является более сложной для новичков.

MASON представляет ещё одну мультиагентную платформу, специализирующуюся на моделировании и симуляции. Она предоставляет широкий набор инструментов для создания и тестирования моделей, включая графический интерфейс и библиотеку агентов. MASON хорошо подходит для исследования социальных и экономических систем. Богатый набор функций предоставляется возможности для работы с большими моделями.

Каждая из этих платформ имеет свои уникальные особенности и предназначена для разных целей. Выбор конкретной МАП зависит от требований проекта, опыта разработчика и предпочтений в стиле программирования. В целом, JADE считается одной из наиболее популярных и широко используемых мультиагентных платформ для Java благодаря своей гибкости, поддержке сообщества и широкому спектру возможностей.

2.2.3 Технологии и инструменты систем искусственного интеллекта

Архитектуры и технологии искусственного интеллекта значительно различаются и применяются в зависимости от конкретных задач и условий, рассмотрим некоторые из них.

Архитектура трансформеров представляет собой архитектуру, основанную на механизме внимания, которая обрабатывает последовательности данных без необходимости предварительного обучения представления входных данных. Примерами трансформеров являются BERT, GPT и T5, эти модели используются для задач NLP, таких как классификация текста, машинный перевод и генерации текста. Трансформеры определяют способность обрабатывать длинные последовательности и эффективны в задачах NLP но предъявляют высокие требования к памяти и вычислительным ресурсам.

Конволюционные нейронные сети – архитектура, которая часто используется для обработки изображений и распознавания образов. CNN обучаются на больших наборах данных, чтобы обнаруживать и классифицировать объекты. имеют способность обрабатывать большие объемы данных и эффективны в задачах компьютерного зрения, но также предъявляют высокие требования к вычислительным ресурсам и сложны в обучении.

Рекуррентные нейронные сети применяются для обработки последовательностей данных, таких как тексты или временные ряды. RNN могут запоминать информацию из предыдущих шагов и использовать ее для принятия решений, обрабатывать последовательности данных и эффективны

в задачах, где важна память, но не решают проблемы с масштабируемостью и трудности в обучении.

Генеративно-состязательные нейросети – архитектура, состоящая из двух конкурирующих сетей: генеративной и дискриминативной. Первая пытается создать новые данные, а вторая — определить, являются ли они реальными или сгенерированными. GAN используются для создания синтетических изображений, видео и аудио, генерации высококачественных синтетических данных и эффективны в задачах, связанных с созданием контента, но не обеспечивают контроль качества сгенерированных данных.

К инструментам и языкам программирования, которые можно использовать для создания искусственного интеллекта относят:

- Python. Самый популярный язык программирования для разработки AI-приложений. Он имеет множество библиотек, таких как TensorFlow, Keras и PyTorch, которые упрощают разработку и обучение нейронных сетей,
- C++. Язык программирования, применяемый для создания высокопроизводительных приложений, разработки и проектирования решений в области искусственного интеллекта, особенно когда требуется высокая производительность,
- Java. Язык программирования, который используется для создания кроссплатформенных приложений. Он также применяется в разработке AI-решений, особенно когда требуется масштабируемость и надежность,
- Rust. Относительно новый язык программирования, который набирает популярность среди разработчиков AI-приложений благодаря своей безопасности и высокой производительности.

2.3 Направление исследования

Уникальной особенностью данного направления является исследование возможности распределенного обучения модели с использованием интеллектуальных агентов. Целью такого распределенного обучения является:

- увеличение скорости обучения модели за счет горизонтального масштабирования узлов в системе,
- уменьшения ресурсных требований к машинам, на которых выполняется код, обучающий модель,
- снижения материальных затрат на дорогостоящее не масштабируемое для обучения оборудование.

Для обеспечения распределенного обучения, реализуются алгоритмы обучения моделей на основе коммуникации агентов и загрузки пользовательских данных. Ввиду того, что модель должна быть доступна для коммуникации в пользовательском интерфейсе, осуществляется фоновое обучение модели, при завершении которого, производится подмена модели «на лету» или в режиме реального времени. Данная особенность позволяет обучать модель и обеспечивать ее доступность с максимальным показателем за счет моментальной подгрузки данных с использованием нативных методов вызова.

Коммуникация объектов представляет собой своего рода репликацию данных для обучения модели конкретного объекта или обученного состояния модели. Данное общение агентов может осуществляться несколькими способами - автоматическим и ручным режимами, постоянный процесс обмена и по нажатию кнопки в пользовательском интерфейсе соответственно.

Обучение на основе данных, предоставляемых пользователем представляет наиболее простой и менее надежный для обеспечения качества модели процесс. Он достигается возможностью подгрузить датасеты для обучения, которые представляют из себя структуру ключ-значение, где в качестве ключа выступает промт, а в качестве значения используется наиболее подходящий, с точки зрения пользователя, ответ. Для обеспечения качества предоставляемых ответов, обучаемые данные необходимо пропускать через специальные системы фильтрации. Эти системы включают в себя фильтр NFSW, эвристический фильтр, семантический дедублированный подход и текстовый классификатор.

Одной из возможностей упрощенного обучения моделей является так называемое дообучение с эффективной обработкой параметров (Parameter-Efficient Fine-Tuning, PEFT). Соответствующие методики реализованы, например, в библиотеке `peft`, которая может проводить обучение низкоразмерных адаптеров (Low-Rank Adaptation, LoRA) на модели, загруженной в 8-битном формате (рисунок 2).

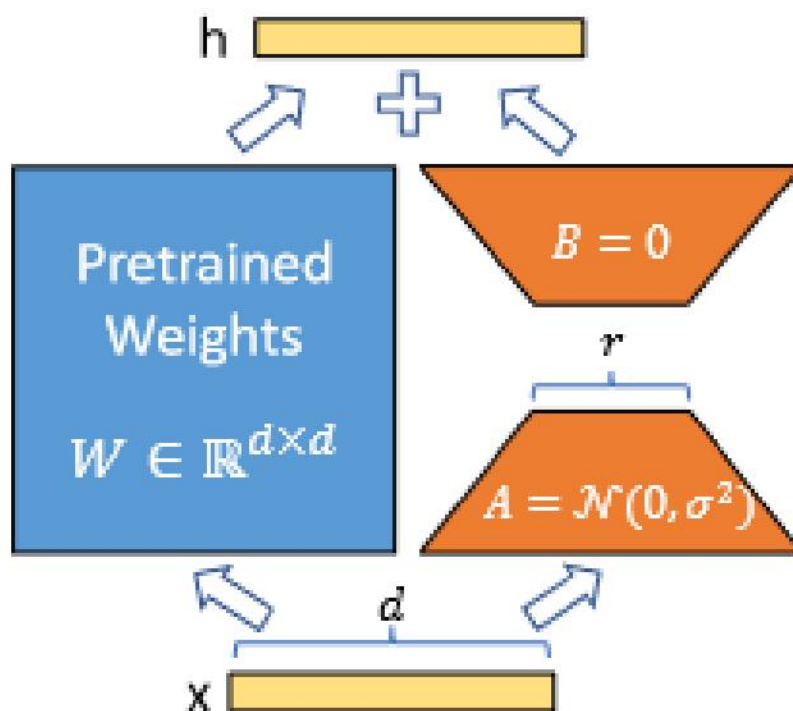


Рисунок 2 – LoRA для линейных слоёв

Загрузка модели в 8-битном формате весьма заметно снижает потребность в памяти, так как на один параметр для веса нужен лишь один байт (например, для модели с 7 миллиардами параметров, понадобится 7 Гб памяти). Вместо того, чтобы напрямую подстраивать в ходе обучения исходные веса, при использовании LoRa поверх некоторых существующих слоёв (обычно — это слои внутреннего внимания) добавляются маленькие слои адаптеров. В результате оказывается, что количество обучаемых параметров очень сильно снижается.

При использовании такого сценария применяют следующее практическое правило: выделять примерно 1,2-1,4 Гб памяти на миллиард

параметров (в зависимости от размера пакета и длины последовательности), чтобы в этот объём поместилось бы всё, что нужно для дообучения модели.

Эти подходы позволяют осуществлять дообучение больших моделей на потребительских устройствах. Это обеспечивает возможность размещать на одном GPU очень большие модели, но их обучение при этом может оказаться весьма медленным процессом. При таком сценарии развития событий самой простой стратегией будет применение параллелизма данных. А именно — на других GPU создают копии среды обучения, после чего передают разным устройствам различные пакеты данных. Это позволяет распараллелить обработку прямых и обратных проходов в модели и масштабировать окружение обучения за счёт количества применяемых GPU.

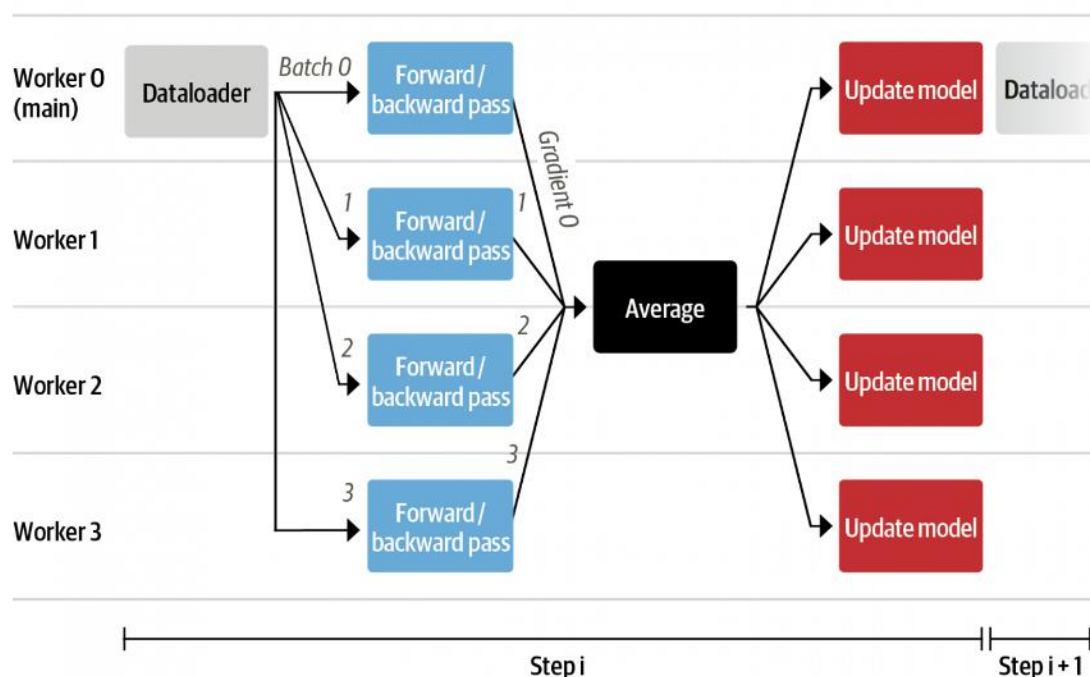


Рисунок 3 - Распараллеливание обучения модели

Современным и технологичным подходом к ускорению обучения моделей

является распараллеливание обучения не только на уровне GPU, но и на уровне вычислительных машин в рамках локальной вычислительной сети. Данный подход представляет возможность практически бесконечного масштабирования, так как оно достигается не увеличением ресурса конкретной вычислительной машины (это ограниченная возможность), а

увеличением количества вычислительных машин для обучения и распределение данных по ним.

3 Постановка задачи

Требуется разработать пиринговую сеть на основе мультиагентной интеллектуальной системы, реализующую графический интерфейс для взаимодействия пользователя с системой и программный интерфейс, включающий в себя совокупность узлов, каждый из которых состоит из интеллектуальных агентов.

Система должна реализовывать следующие функции:

- отправка сообщения пользователем,
- оценка ответа сообщения,
- отправка широковещательного запроса в сети,
- отправка сообщения на определенный узел сети,
- обучение модели на основе датасета,
- создание диалоговых комнат,
- просмотр диалогов.

Функциональные требования:

- пользователь регистрируется в пиринговой сети,
- для пользователя создаются минимально необходимые для работы узлы: агент интерфейса, агент подключения к нейронной сети, агент для работы с базой данных,

- пользователь создает диалоговые комнаты,
- пользователь отправляет сообщения своему узлу,
- пользователь отправляет сообщения другому узлу пиринговой сети,
- пользователь отправляет широковещательный запрос в сеть,
- пользователь оценивает результат работы интеллектуальных агентов,
- агенты обмениваются данными/обученными моделями по процессу системы,

- агенты обмениваются данными/обученными моделями по запросу пользователя,
- агенты обучают модель на основе загружаемых пользователем данных.

Нефункциональные требования:

- минимальная задержка отправки данных до своего узла,
- минимальная задержка отправки данных до другого узла,
- минимальная задержка отправки данных до по сети,
- отказоустойчивость системы,
- балансировка узлов пиринговой сети,
- service-discovery для поиска узлов в сети,
- доступность на запись.

Общие требования:

- операционные системы Microsoft Windows, Linux, Mac OS,
- системы программирования IntelliJ IDEA, Xcode,
- языки программирования Java, C++,
- фреймворки и библиотеки: JaxaFx, JADE, JNI,
- реляционная СУБД PostgreSQL/нереляционная СУБД MongoDB.

4 Выбор инструментов разработки

Современные компьютерные ресурсы обладают большими мощностями, совершенно несравнимыми с возможностями представителей прошлого века. Но даже в век информационного технологического прогресса необходимо правильно и разумно использовать вычислительные мощности при выполнении различных объемных и тяжелых операций.

Одним из весомых аспектов построения систем, выполняющих критически важные операции, является автономность компонентов приложения и разделение операций на атомарные и транзакционные единицы внутри них.

Вышеописанные требования успешно покрываются при проектировании архитектуры на основе паттернов распределенных и интеллектуальных агентно-базированных систем, которые обеспечивают отказоустойчивое и надежное функционирование сервисов в виде агентов в сети, а также реализуют высокую доступность и предоставляют возможности динамической балансировки агентов, управление которой может осуществляться в ручном режиме посредством администрирования системы, так и в процессе ее жизнедеятельности - автоматической регулировкой нагрузки системы за счет специализированных агентов. Технология реализации пиринговой сети в агентно-ориентированной метакomпьютерной или облачно-сетевой грид среде основана на поддержке динамического взаимодействия, перемещения, добавления и удаления агентов, что позволяет масштабировать вычислительные ресурсы за счет добавления новых агентов в группу, экземпляры которой параллельно и независимо друг от друга выполняют заданные операции, а использование концептуальных моделей искусственного интеллекта обеспечивает интероперабельность интеллектуальных систем на уровнях внешнего и внутреннего проектирования.

Интеллектуальность данных систем достигается не только путем обучения интеллектуальных агентов работы с конкретными данными, программированием их поведения и балансировки нагрузки, но и обучением алгоритмов и нейросетей, встраиваемых непосредственно в работу агентов.

В данной работе рассматривается проектирование распределенной масштабируемой интеллектуальной системы, поддерживающей динамическую балансировку и реализующей пиринговую сеть.

4.1 Мультиагентная платформа JADE

Наиболее распространенной, популярной и широко используемой агентной платформой является платформа JADE, проект которой разрабатывается компанией Telecom Italia Lab.

Java Agent Development Framework представляет собой программную среду, которая используется для разработки мультиагентных систем и приложений и поддерживает FIPA-стандарты для интеллектуальных агентов, включая в себя:

- среду выполнения агентов (агенты регистрируются и работают под управлением среды),
- библиотеку классов, которые используются для разработки агентных систем,
- набор графических утилит, обеспечивающих возможность администрирования платформы, наблюдения и управления за жизненным циклом агентов.

JADE – это программное обеспечение промежуточного слоя (рисунок 4), разработанное компанией TILAB, предназначенное для создания распределенных мультиагентных приложений на основе транспортной архитектуры «точка-точка». Все ключевые элементы, такие как интеллект, инициатива, информация, ресурсы, контроль могут быть равномерно распределены между мобильными терминалами и компьютерами выделенной сети. Эта среда способна динамически взаимодействовать с узлами, которые в терминологии JADE называются агентами. Данные агенты способны

появляться и исчезать в системе в зависимости от требований программной среды.

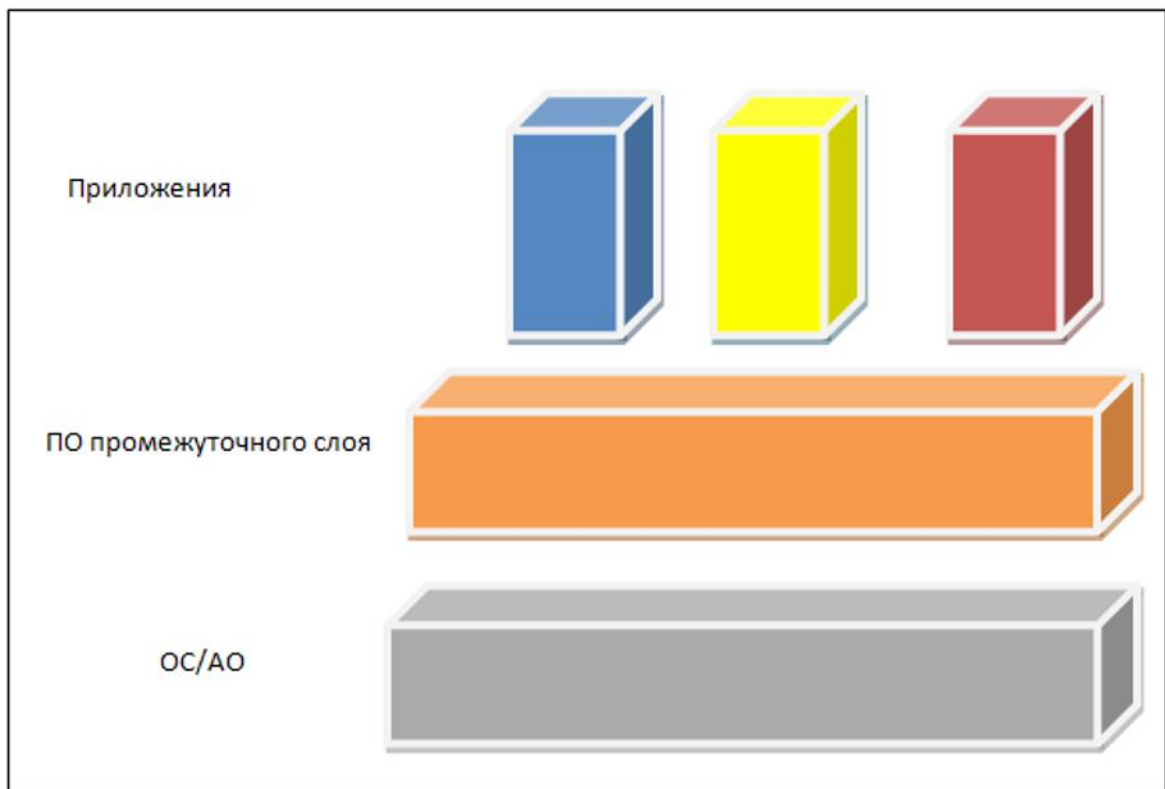


Рисунок 4 – Программное обеспечение промежуточного слоя

JADE предоставляет инженеру-разработчику агентных систем определенный набор средств:

- FIPA-compliant Agent Platform. Платформа на основе стандарта FIPA, включающая обязательные системные агенты определенных типов, которые автоматически запускаются при старте платформы,
- distributed Agent Platform. Распределенная агентная платформа, которая способна использовать несколько компьютеров в качестве узлов, на каждом из которых работает только одна Java Virtual Machine. Агенты выполняются как Java-потoki, а для обмена сообщениями между агентами, в зависимости от их расположения, используется механизм транспортировки под названием множественная поддержка доменов. Данный механизм представляет собой группу агентов, основанных на спецификациях FIPA, которые могут объединяться в федерацию, тем самым реализуя мультидоменную агентную среду,

- multithreaded execution environment with two-level scheduling. Многопоточное окружения для исполнения обеспечивает каждому JADE-агенту собственный поток управления и возможность работать в многопоточном режиме. JVM осуществляет планирование задач, которые исполняются агентами или одним из них,

- object-oriented programming environment. Большинство концепций, присущих FIPA-спецификации, представлены в виде Java-классов, которые формируют пользовательский интерфейс,

- library of interaction protocols. Стандартные интерактивные протоколы fipa-request и fipa-contract-net используются для создания агента, способного действовать в соответствии с правилами данных протоколов. Разработчикам прикладных программ необходимо лишь реализовать специфические доменные действия, тогда как вся независимая от прикладной программы логика будет реализовываться платформой JADE,

- administration GUI. Различного уровня сложности операции по управлению платформой могут исполняться через встроенный пользовательский интерфейс, который отображает активных агентов и контейнеры агентов. Используя графический интерфейс, администраторы мультиагентной системы имеют возможность выполнять базовые функции, такие как создание, уничтожение, прерывание и возобновление действия агентов, создание иерархии доменов и мультиагентных федераций.

Платформа JADE реализована на языке программирования Java и ее в основу заложены следующие принципы:

- функциональная совместимость. Платформа JADE разработана в соответствии со спецификациями FIPA (рисунок 5), следствием чего агенты платформы имеют возможность взаимодействовать со сторонними агентами, которые поддерживающими этот стандарт,

- портируемость и единообразие. Продукт JADE предоставляет собой однородный набор прикладных программных интерфейсов (API), которые не являются зависимыми от внешних условий, таких как базовое

устройство сети или версии платформы Java. В процессе своей работы данное программное обеспечение предоставляет одинаковое API для различных окружений, например J2EE, J2SE, J2ME. При развертывании приложений инженерам необходимо определить тип среды исполнения Java,

- простота использования. Обеспечивает контракт, состоящий из набора простых и интуитивно-понятных интерфейсов, скрывая сложную логику реализации программного обеспечения промежуточного слоя от пользователя,

- принцип «разрабатывать по средствам». Инженерам достаточно использовать только необходимые возможности, которые предоставляет ПО промежуточного слоя, то есть не требуется знать что-либо об неиспользуемых функциях платформы. Недействующих функций системы не создают дополнительные ресурсные или вычислительные расходы.

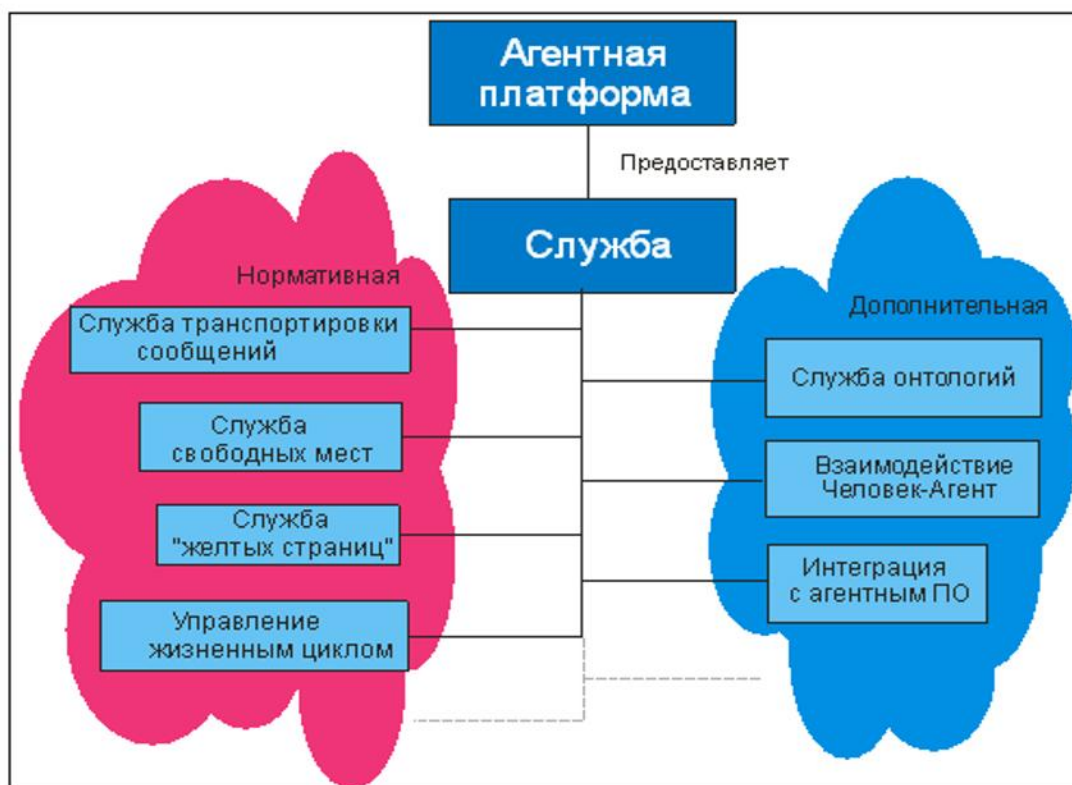


Рисунок 5 – Службы, предоставляемые агентной платформой по стандарту FIPA

4.2 Библиотеки искусственного интеллекта

В сфере искусственного интеллекта используется большое количество языков программирования, одним из которых является широко известный и

твёрдо утвердившийся язык Java. Java применяют для различных целей, например от создания решений для машинного обучения и нейронных сетей до алгоритмов поиска, генетического программирования и мульти-робототехнических систем.

Такие свойства, как кодирование различных типов алгоритмов, объектно ориентированный подход и возможность гибкого масштабирования, обязательны для проектов в сфере ИИ. Java обеспечивает указанные выше функции и подходит как нельзя лучше. Сегодня технологии искусственного интеллекта активно применяются в различных сферах, в частности в бизнесе, где очень востребованы возможности Java, которые позволяют реализовать единую версию приложения, работающую на нескольких платформах.

К сильным сторонам и преимуществам Java также относят простоту создания и отладки кода, обеспечение упрощенной работы с крупномасштабными проектами и визуализацией, а также наличие Swing и SWT, что позволяет создавать сложную графику и пользовательские интерфейсы.

Следует также отметить, что ИИ представляет обширную область, и в рамках данного реферата затрагивается одна из наиболее популярных тем ИИ – разработка и применение нейронных сетей.

Наиболее популярными библиотеками языка программирования Java в сфере искусственного интеллекта являются:

- `neurph`. Библиотека с открытым исходным кодом, используемая для создания нейронных сетей и применения машинного обучения,
- `deeplearning4j`. Библиотека глубокого обучения, которая предоставляет API для создания нейронных сетей.

4.2.1 Neuroph

Neuroph – это облегченная среда нейронной сети Java, которую можно использовать для разработки общих архитектур нейронных сетей. Небольшое количество базовых классов, которые соответствуют основным концепциям

NN, и редактор графического интерфейса, упрощают изучение и использование (рисунок 6).

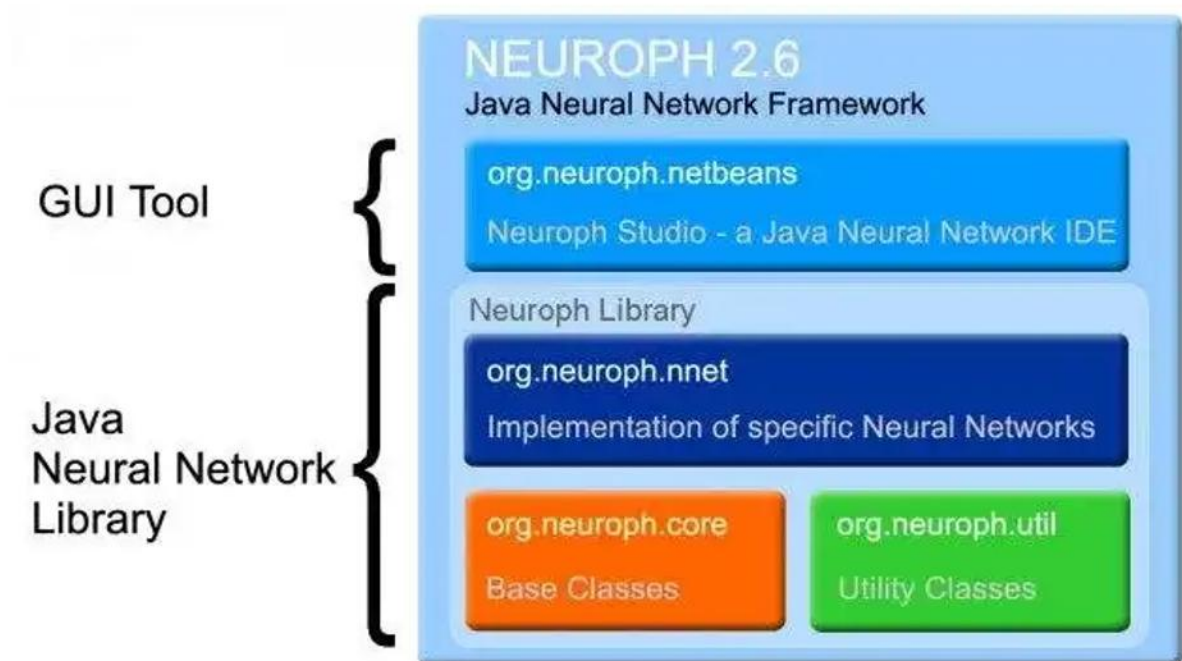


Рисунок 6 – Структура библиотеки Neuroph

Выделяют следующие особенности библиотеки:

- поддерживает многослойные персептроны с обратным распространением,
- поддержка распознавания изображений,
- простой в использовании графический интерфейс для создания и экспериментов с NN,
- поддерживает различные другие нейронные сети и правила обучения,
- простой в использовании API.

На рисунке 7 демонстрируется диаграмма классов библиотеки Neuroph.

Neuroph Framework 2.3 Core Class Diagram

Java Neural Network Framework
<http://neuroph.sourceforge.net>

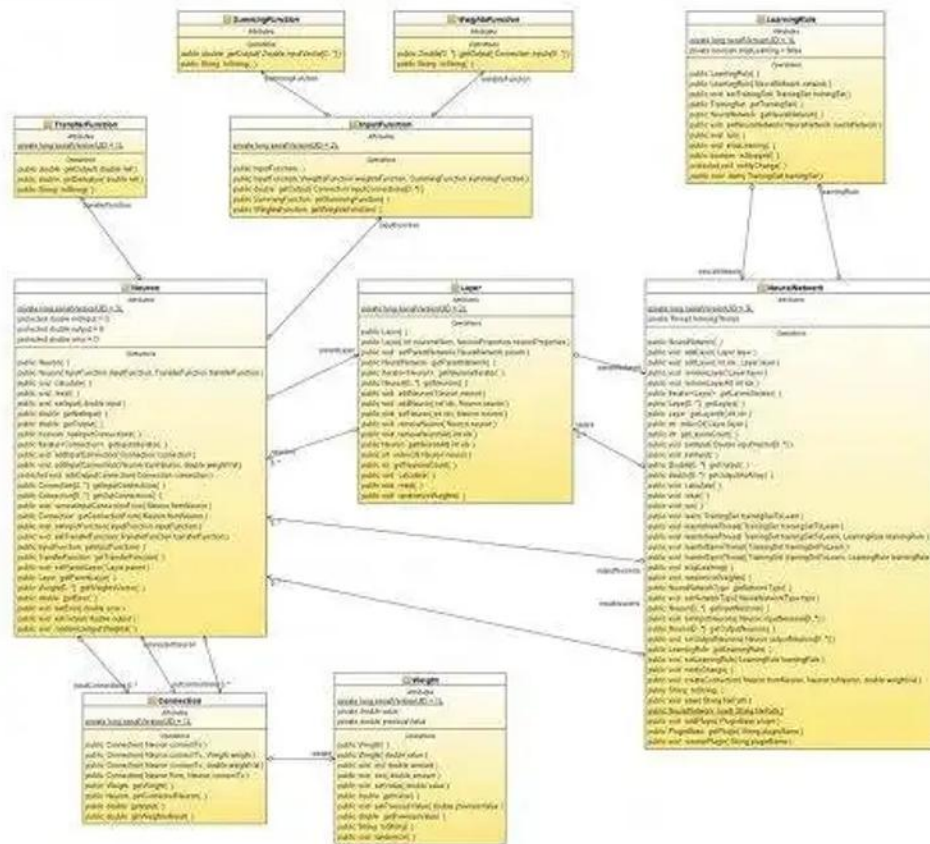


Рисунок 7 – Диаграмма классов библиотеки Neuroph

Neuroph предоставляет несколько вариантов взаимодействия пользователя с библиотекой, используя:

- инструменты на основе графического интерфейса,
- библиотеку в Java-коде.

Оба подхода основаны на базовой иерархии классов, которая строит искусственные нейронные сети из слоев нейронов .

В рамках данного реферата основное внимание будет уделено программной стороне, но также будут приведены примеры нескольких общих классов из подхода Neuroph на основе графического интерфейса.

Непосредственное внедрение нейросетевых технологий в распределенную агентно-ориентированную систему (рисунок 4), реализующую принятие решений агентов на основе обученных нейросетей, является основной целью для данной научно-исследовательской работы.

4.2.2 Deeplearning4j

Deeplearning4j – это библиотека на языке Java, которая используется как совокупность программа, представленная в виде фреймворка для глубокого обучения. В реализацию данной библиотеки включены такие алгоритмы, как:

- ограниченная машина Больцмана,
- рекурсивная тензорная нейронная сеть,
- стековый автокодировщик с фильтрацией шума,
- глубокие сети доверия,
- глубокий автокодировщик,
- word2vec,
- doc2vec,
- GloVe.

Данная библиотека является открытым программным обеспечением, которая распространяется под лицензией Apache 2.0, основными разработчики которой выступает группа машинного обучения в Сан-Франциско во главе с Адамом Гибсоном, а непосредственно коммерческие внедрения поддерживает стартап Skymind.

Deeplearning4j написана на языке программирования Java и выполняется в этой же среде, но несмотря на это обеспечивает совместимость с Clojure и включает интерфейс для языка Scala.

Фреймворк обеспечивает возможности различного комбинирования компонентов, за счет объединения нейронных сетей с машинами Больцмана, сверточными нейронными сетями, автокодировщиками и рекуррентными сетями в одну систему. Также реализована поддержка расширенных средств, которые используются для визуализации. В зависимости от потребностей обучения, оно может проводится либо с помощью обычных многослойных нейронных сетей, либо же со сложными сетями, у которых определён граф вычислений.

Deeplearning4j осуществляет обучение через кластеры, в таком случае нейронные сети обучаются параллельно по итерациям. Библиотека

предоставляет интеграцию с ядром архитектуры CUDA, чтобы обеспечить возможность чистых операций с GPU и способствовать распределению операций на графических процессорах.

Deeplearning4j включает в себя функции, собранные в единый класс, для работы с многомерными массивами данных в библиотеке ND4J, что способствует облегчению вычислений для научных задач на языках Java и Scala, но при этом функциональность соответствует языку NumPy для языка программирования Python. Использование этих средств позволяет эффективно решать задачи линейной алгебры и матричных преобразований в практических приложениях.

Библиотека DataVec, которая включена в фреймворк, осуществляет векторизацию файлов различных входных форматов и обеспечивает возможность генерации различных выходных форматов методом, подобным MapReduce, непосредственные данные трансформируются в векторную форму.

В библиотеку включены реализации, которые оптимизированы на Java:

- частотной инверсии,
- глубокое обучение,
- алгоритм Миколова,
- word2vec,
- doc2vec,
- GloVe,
- принцип стохастического встраивания соседей с распределением

Стюдента для реализации облака слов.

Мощности параллельных графических процессоров фреймворка используют инструментарий векторного и тематического моделирования для обработки больших объемов текстов.

5 Описание архитектуры

5.1 Архитектура мультиагентной системы

Платформа JADE включает в себя:

- встроенные библиотеки, представляющие собой наборы Java-классов, которые требуются для разработки интеллектуальных агентов,
- среду исполнения, включающую в себя базовые службы и обеспечивающая активность на устройстве во времени исполнения агента.

Каждый экземпляр платформы JADE во время своего исполнения называется контейнером, реализуя физическое разделение и логическое объединение агентов.. Совокупность контейнеров называется платформой (рисунок 8) и формирует однородный слой, который прячет от агентов сложность и распределенный характер механизмов, которые располагаются на более низком уровне, например:

- аппаратное обеспечение,
- операционные системы,
- типы сетей,
- JVM.

Под контейнером понимается динамическая среда, используемая для исполнения мультиагентных приложений и содержащая совокупность агентов. Набор таких активных контейнеров формируется в совокупность, которая называется платформой. В совокупности контейнеров всегда определяется главный (Main container), который может быть только одним, а все остальные коммуницируют с ним и осуществляют свою регистрацию в момент запуска. Данный принцип объясняет причину, почему первым контейнером при старте платформы должен быть именно главный контейнер, а все остальные должны быть «подчиненными» контейнерами и заранее определить главный контейнер, на котором на котором будет осуществлена регистрация. Другой главный контейнер, который может быть запущен в сети, представляет собой совокупность других контейнеров.

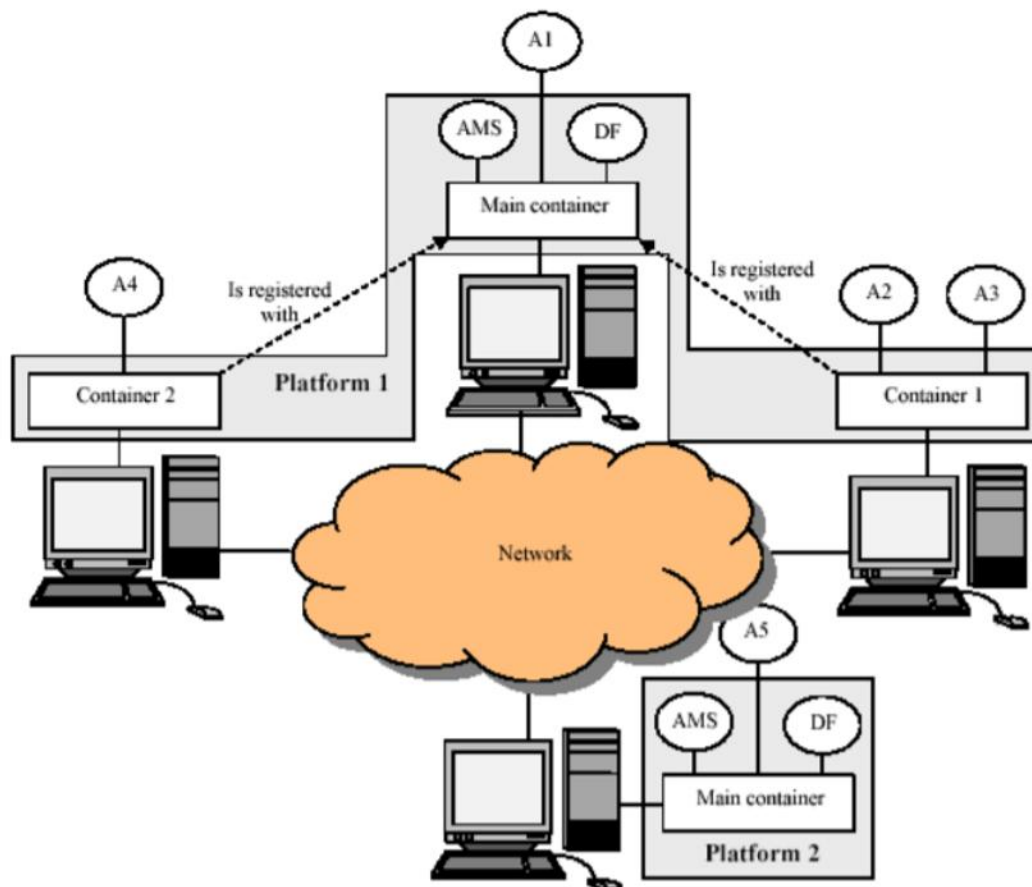


Рисунок 8 – Платформа JADE

Рисунок 8 иллюстрирует концепцию двух JADE-платформ, которые состоят из трех и одного контейнера соответственно. JADE-агенты определяются уникальными именами и при наличии данных имен друг у друга, они могут общаться, независимо от их фактического местонахождения: на разных платформах, в разных контейнерах на одной платформе или в общем контейнере.

5.2 Архитектура нейронной сети

В качестве архитектуры нейронной сети выбран тип трансформер. Архитектура трансформера искусственного интеллекта представляет собой модель машинного обучения, которая использует механизм внимания для обработки последовательностей. Эта архитектура была впервые представлена в статье "Attention Is All You Need" (внимание - это все, что вам нужно) в 2017 году.

Основная идея заключается в том, чтобы заменить рекуррентные нейронные сети (RNN), такие как LSTM и GRU, на механизм внимания,

который может фокусироваться на важных частях входной последовательности. Это позволяет модели обрабатывать длинные последовательности без потери информации.

Архитектура трансформера состоит из нескольких компонентов, рассмотрим каждый из них. Encoder – блок отвечает за кодирование входной последовательности. Он содержит несколько слоев, каждый из которых состоит из двух подблоков, а именно мультиплексора внимания и многонаправленного сверточного слоя. Мультиплексор внимания используется для определения важности каждого элемента в последовательности, а многонаправленный сверточный слой применяется для обработки соседних элементов. Decoder отвечает за декодирование выходной последовательности. Он аналогично encoder содержит несколько слоев, однако, в отличие от него, decoder также имеет дополнительный мультиплексор внимания, который называется "выходным мультиплексором внимания", который используется для получения информации о предыдущих выходах модели.

Механизм внимания представляет собой ключевой компонент архитектуры трансформера. Механизм внимания используется для определения важности каждого элемента в последовательности. Он работает путем создания вектора внимания для каждого элемента, который затем используется для взвешивания соответствующих элементов.

Подготовительные слои используются для преобразования входных данных перед их подачей в encoder. Они могут включать в себя слои нормализации, линейные слои и слои активации. Постпроцессинговые слои используются для преобразования выходных данных после их получения от decoder. Они могут включать в себя слои нормализации, линейные слои и слои активации.

Архитектура трансформера показала высокую эффективность в различных задачах обработки естественного языка, таких как машинный перевод, распознавание именованных сущностей и чтение comprehension.

Архитектура трансформера, основанная на механизме внимания, стала основой для создания многих моделей больших языковых моделей (LLM), рассмотрим некоторые из них. BERT – одна из самых известных моделей LLM, разработанная Google AI Language Team. BERT обучена на большом объеме текстовых данных и способна понимать контекст слов в предложении и используется для решения множества задач в области обработки естественного языка, включая классификацию текста, семантический анализ и машинный перевод. GPT и его последующие версии, включая GPT-2 и GPT-3, разработаны OpenAI. GPT-3 является одной из самых мощных моделей LLM на сегодняшний день, обеспечивая возможность генерировать человекоподобный текст, отвечать на вопросы и даже создавать свои собственные рассказы и стихи. T5 – модель, разработанная Google AI, которая обучается на огромном количестве текстовых данных, способна выполнять широкий спектр задач обработки естественного языка, включая классификацию, генерацию текста и исправление ошибок, и использует одну и ту же архитектуру для всех задач, что упрощает ее использование. XLNet – модель, разработанная исследователями из Google AI и Columbia University, которая сочетает в себе преимущества моделей с рекуррентными нейронными сетями и трансформерами и показывает высокие результаты в задачах понимания естественного языка.

Архитектура выбранной модели определяет нацеленность на эффективность и производительность в задачах обработки естественного языка. Использование фреймворка Transformer обеспечивает эффективность вычислений, за счет реализации только на архитектуре декодера.

Модель генерирует выходные данные, исходя из знаний исключительно предыдущего контексте и не требует кодирования входные данные. Это позволяет повысить производительность модели относительно генерации данных. На рисунке 9 продемонстрирована архитектура выбранной LLM-модели.

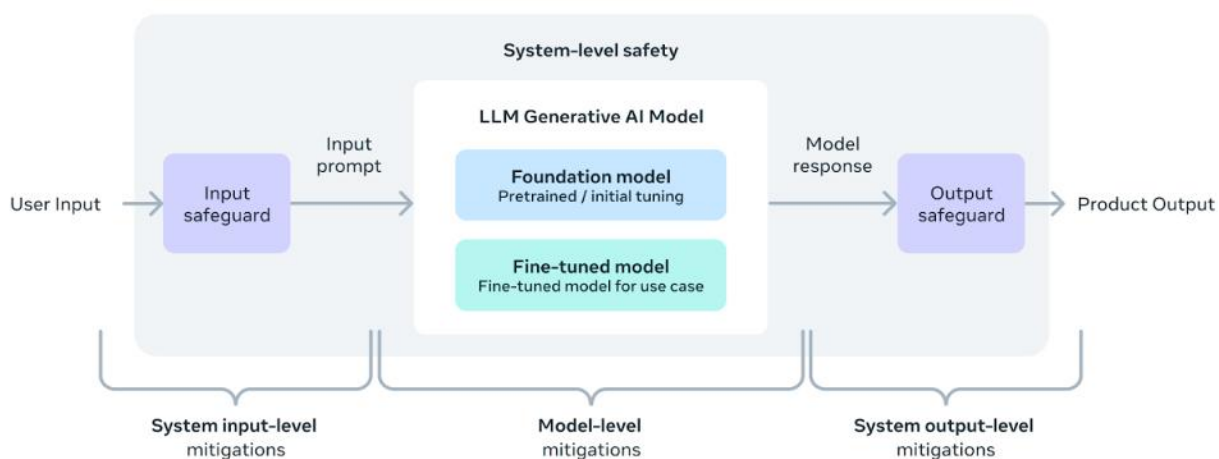


Рисунок 9 – Архитектура LLM модели

5.3 Архитектура метакomпьютера

Метакomпьютер представляет собой совокупность взаимосвязанных узлов, обменивающихся данными и предназначены для выполнения распределенных вычислений, расчетов, обучения.

В данной работе рассматривается архитектура метакomпьютера, построенная на основе технологии пиринговой сети, в рамках которой существует N-ое количество узлов системе, каждый из которых является равноправным ее участником и может выполнять функции и клиента, и сервера одновременно. Рисунок 10 представляет собой обобщенную схему архитектуры метакomпьютера, построенного на основе пиринговой сети.

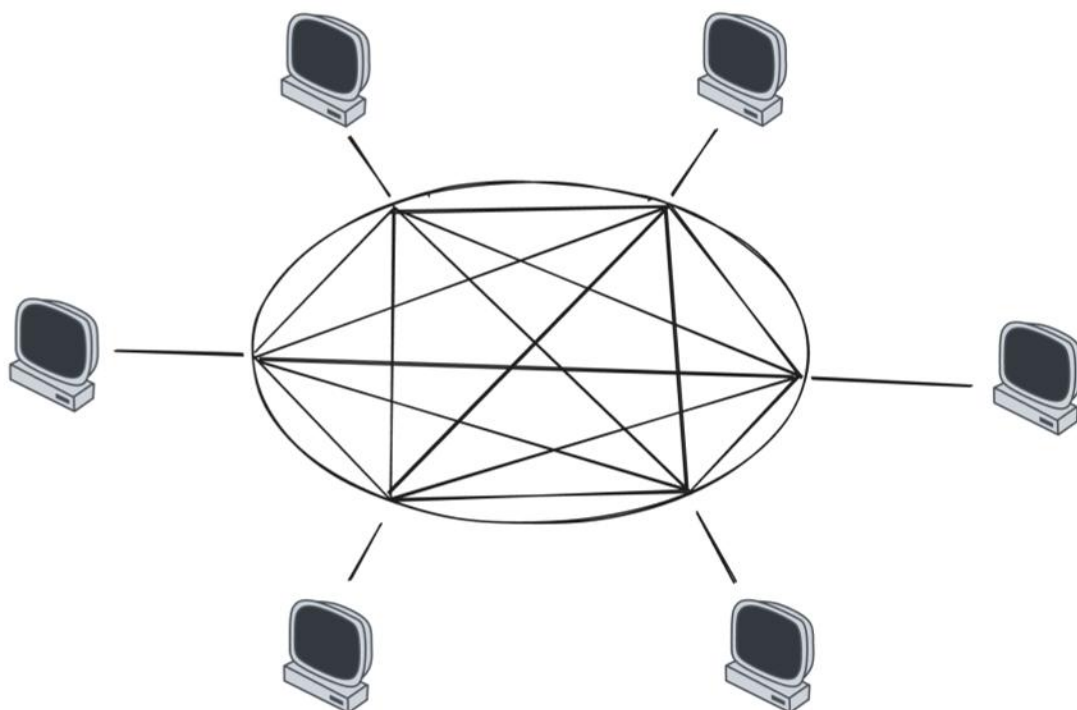


Рисунок 10 – Обобщенная схема пиринговой сети

Реализация данной пиринговой сети основывается на фреймворке JADE, который представляет собой мультиагентную систему. Системы такого типа состоят из агентов. Под агентом понимается программный объект, которые умеет:

- воспринимать происходящую ситуацию,
- принимать решения,
- осуществлять коммуникации с другими объектами,
- устанавливать динамические связи.

Под мультиагентной системой понимается совокупность интеллектуальных запрограммированных агентов, которые организованы в один или несколько контейнеров и предназначены для решения поставленной задачи. В данном случае интеллектуальные объекты объединены в так называемые сообщества, которые представляют собой узлы системы. Целью каждого узла в данном случае выступают функции обработки данных или запросов, которые приходят либо из сети, либо от пользователя через интерфейс, тем самым узел реализует и клиентскую и серверную части одновременно. Расширенную схему пиринговой сети с минимально необходимым набором агентов каждого узла можно рассмотреть на рисунке 11.

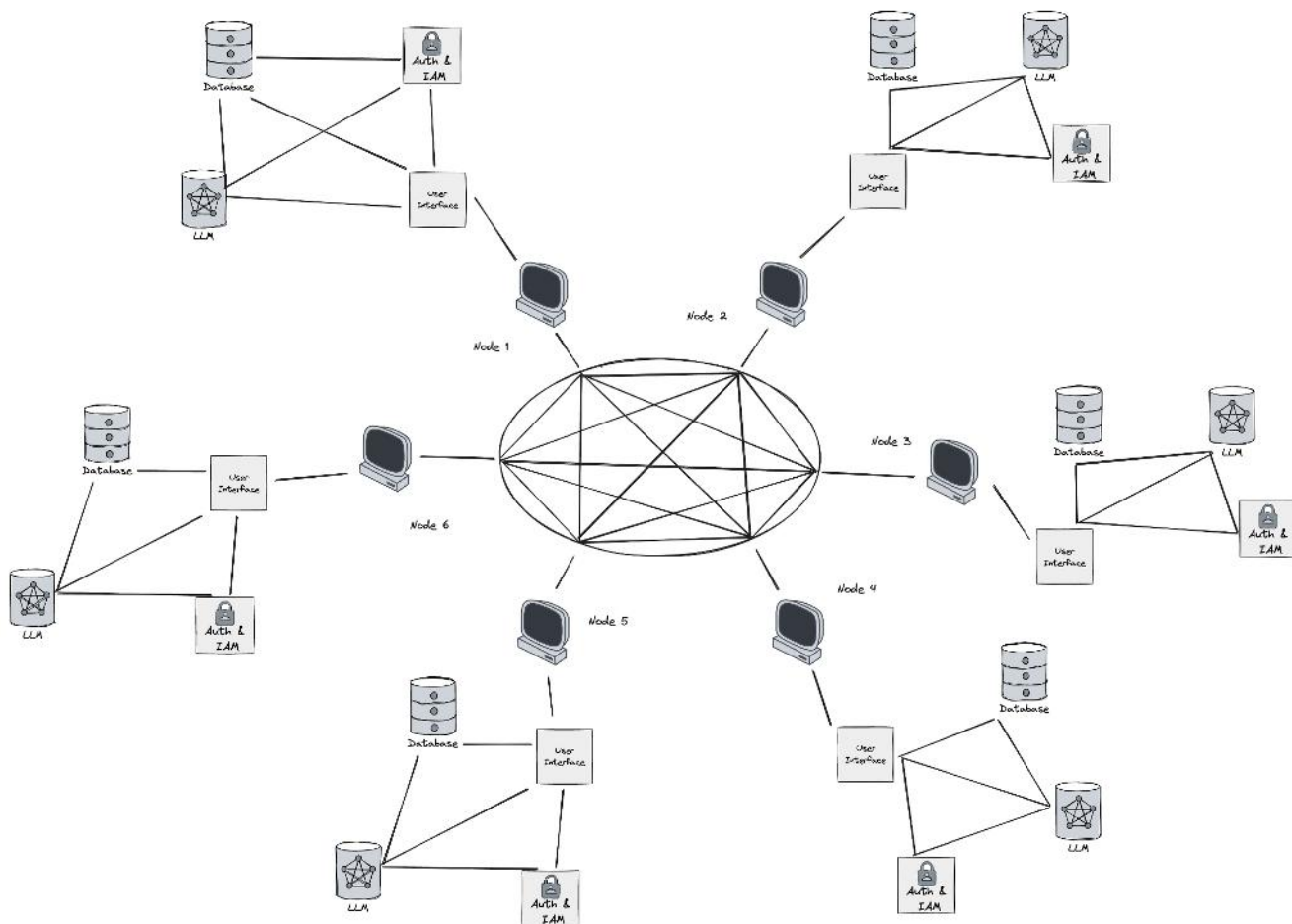


Рисунок 11 – Расширенная схема пиринговой сети

Сложность реализации архитектуры метакomпьютера заключается в проектировании и написании масштабируемого и динамического приложения. Причем, масштабируемость данного решения должна достигаться его динамическим конфигурированием: встроенными средствами администрирования или настройкой окружения «на лету» без изменения программного кода.

Динамическая реализация в рамках данной работы обусловлена проектированием балансировки нагрузки узлов и простой возможностью их добавления или удаления, активации или деактивации в сервисе желтых страниц, а также за счет автоматического циклического поиска и публикации агентов с определенными параметрами однозначно характеризующими агентов в глобальной сети по выполняемой функции и принадлежности к конкретной группе.

Схема, представленная на рисунке 9, формирует состав пиринговой сети, в рамках которой определен ряд узлов. Каждый узел состоит как минимум из 4 интеллектуальных агентов JADE:

- агент, реализующий пользовательский интерфейс,
- агент, реализующий взаимодействие с базой данных,
- агент, реализующий взаимодействие с искусственный интеллект,
- агент, реализующий авторизацию и регистрацию пользователя в

сети и проверку корректность авторизационных данных однонаправленных и широковещательных запросов по сети.

Обобщенная диаграмма классов одного узла агента продемонстрирована на рисунке 12 и отражает иерархическую структуру использования подхода объектно-ориентированного программирования.

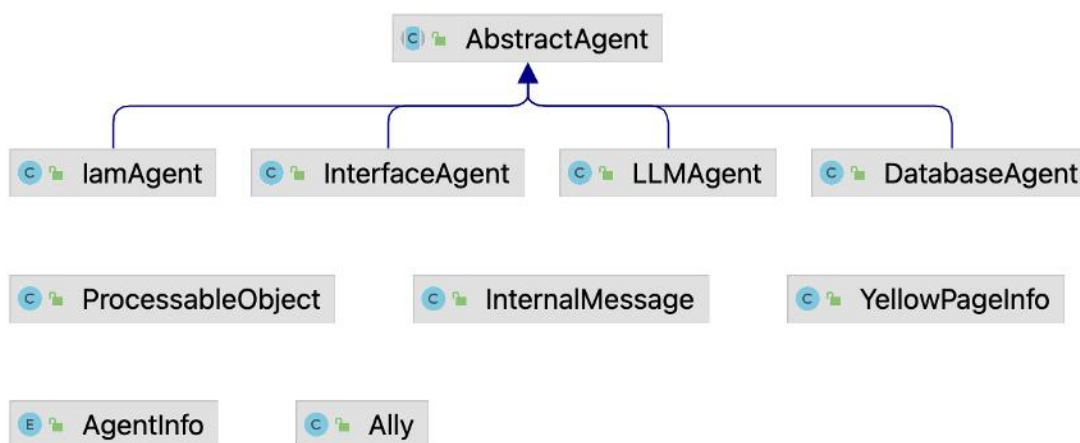


Рисунок 12 – Обобщенная диаграмма классов

Каждый из агентов имеет свою кодовую базу и реализует описанные выше им функции, совокупно выполняя общую задачу функционирования узла в системе. Каждый агента узла, находясь в “выделенной группе” осуществляет коммуникацию внутри между агентами для реализации внутренних функций или обрабатывая события пользовательского интерфейса. Также, агент работает и в сети в целом от имени контейнера, который является частью пиринговой сети и предназначенный для логического объединения JADE-агентов. Рисунок 13 схематичного

отображает внутренние и внешние взаимодействия агентов узла с агентами сети.

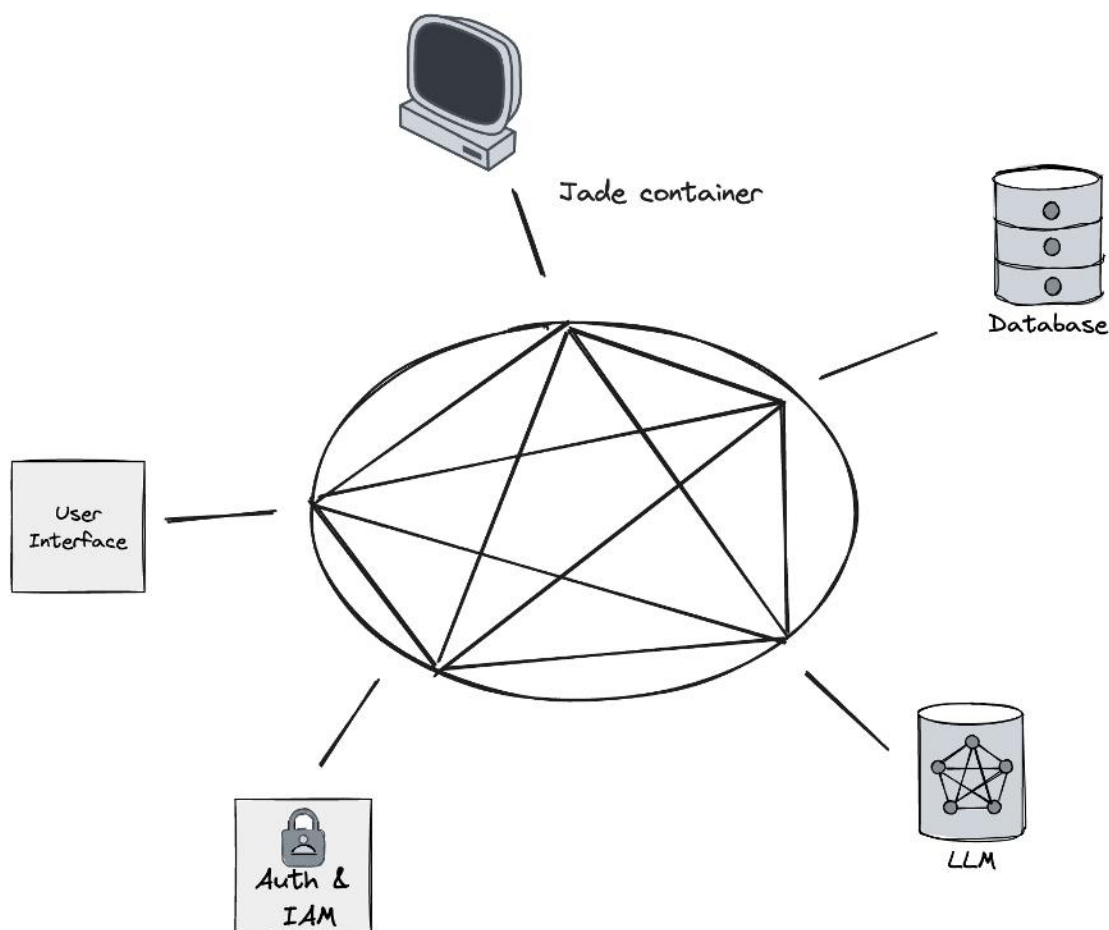


Рисунок 13 – Схема узла пиринговой сети

Отдельно реализованный объект представляет собой самостоятельную единицу, встроенную в коммуникацию определенного узла сети. В зависимости от функциональности и реализации объект может представлять собой отдельный класс или же целый модуль, который разработан для выполнения функций агента. Так, например, для агента взаимодействия с искусственным интеллектом реализован отдельный модуль, который включает в себя следующие основные составляющие (рисунок 14):

- загрузчик нативной библиотеки,
- API и модели данных для взаимодействия с моделью,
- вспомогательные классы, такие как классы для определения операционной системы исполняемой вычислительной машины.

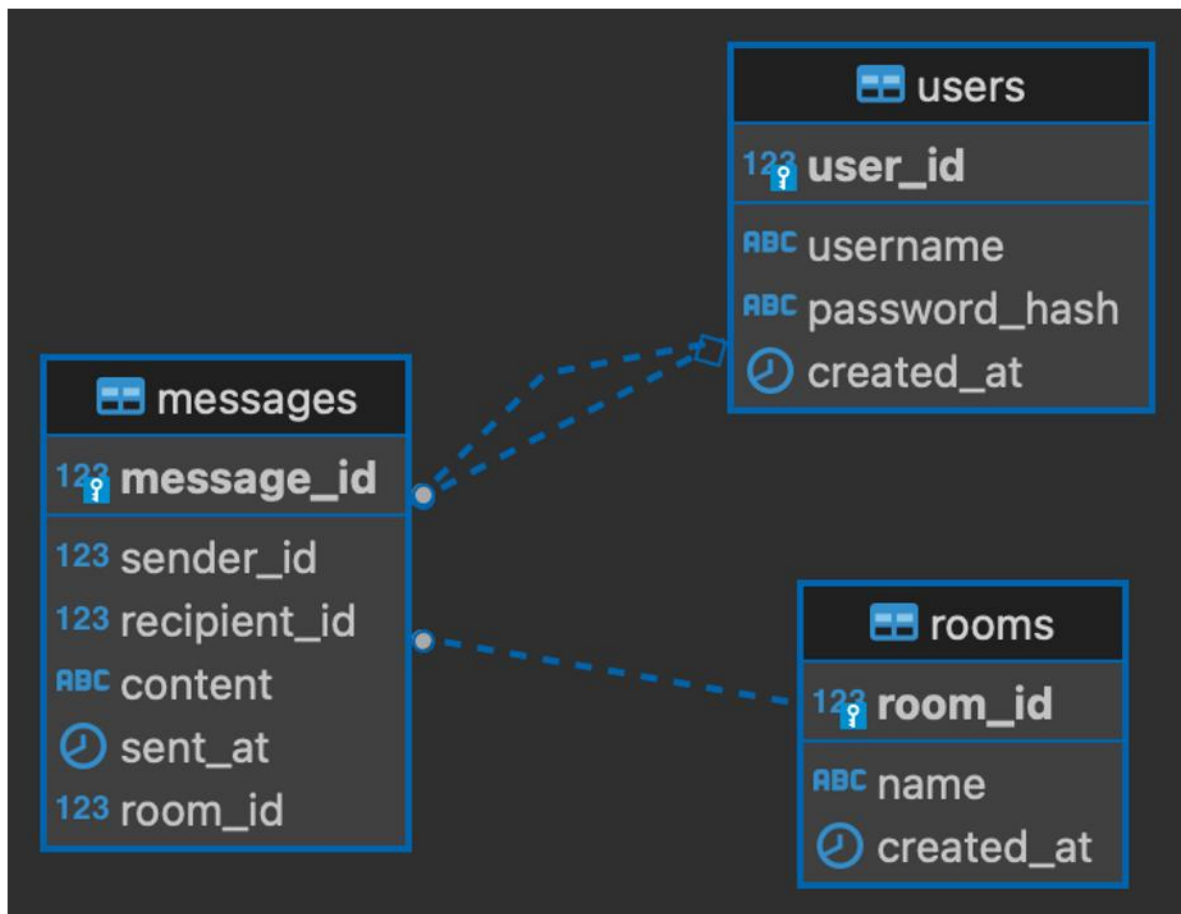


Рисунок 15 – Схема базы данных

При создании базы данных чата учтены следующие сущности: сообщения, пользователи, комнаты. Вот примерная структура таблиц:

Таблица users:

- **user_id** (INT, PRIMARY KEY) - уникальный идентификатор пользователя,
- **username** (VARCHAR) - имя пользователя,
- **password_hash** (CHAR) - хэш пароля пользователя,
- **created_at** (DATETIME) - дата и время регистрации пользователя.

Таблица rooms:

- **room_id** (INT, PRIMARY KEY) - уникальный идентификатор комнаты,
- **name** (VARCHAR) - название комнаты,
- **created_at** (DATETIME) - дата и время создания комнаты.

Таблица messages:

- message_id (INT, PRIMARY KEY) - уникальный идентификатор сообщения,
- sender_id (INT, FOREIGN KEY) - идентификатор отправителя сообщения,
- recipient_id (INT, FOREIGN KEY) - идентификатор получателя сообщения (может быть NULL, если сообщение отправлено в общую комнату),
- content (TEXT) - содержимое сообщения,
- sent_at (DATETIME) - дата и время отправки сообщения,
- room_id (INT, FOREIGN KEY) - идентификатор комнаты, в которой было отправлено сообщение.

6 Описание алгоритма

Программа представляет собой проект, написанный на языке программирования Java с использованием фреймворков JADE (агентная платформа), Maven (фреймворк для автоматизации сборки проектов), DeepLearning4j (фреймворк для обучения нейронных сетей).

Проект разработан в соответствии с концепциями агентно-ориентированного программирования, обеспечивающего описание агентов системы на основе средств, предоставляемых платформой JADE, и объектно-ориентированного программирования, благодаря которой, программный продукт представлен в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы составляют иерархию наследования.

В данном проекте каждый агент представлен в виде описания определенного класса, вычислительная модель которого является многозадачной, то есть позволяет реализовывать несколько функций одновременно. Функциональные возможности, предоставляемые агентом и запрограммированные как исполняемые задачи, реализованы как одно или несколько поведений агента, при этом, внутренний планировщик, автоматически управляет планированием и исполнением поведений.

По-умолчанию в системе каждому агенту платформы соответствует один поток, но данный подход обеспечивает возможность конфигурации политики выделения потоков агентам:

- использование одного потока на всех агентов,
- жесткое выделение одного потока на одного агента,
- создание фиксированного пул рабочих потоков, каждый из которых распределяется по агентам в разных соотношениях.

На рисунке 16 описано визуальное представление каждого экземпляра-агента программы.

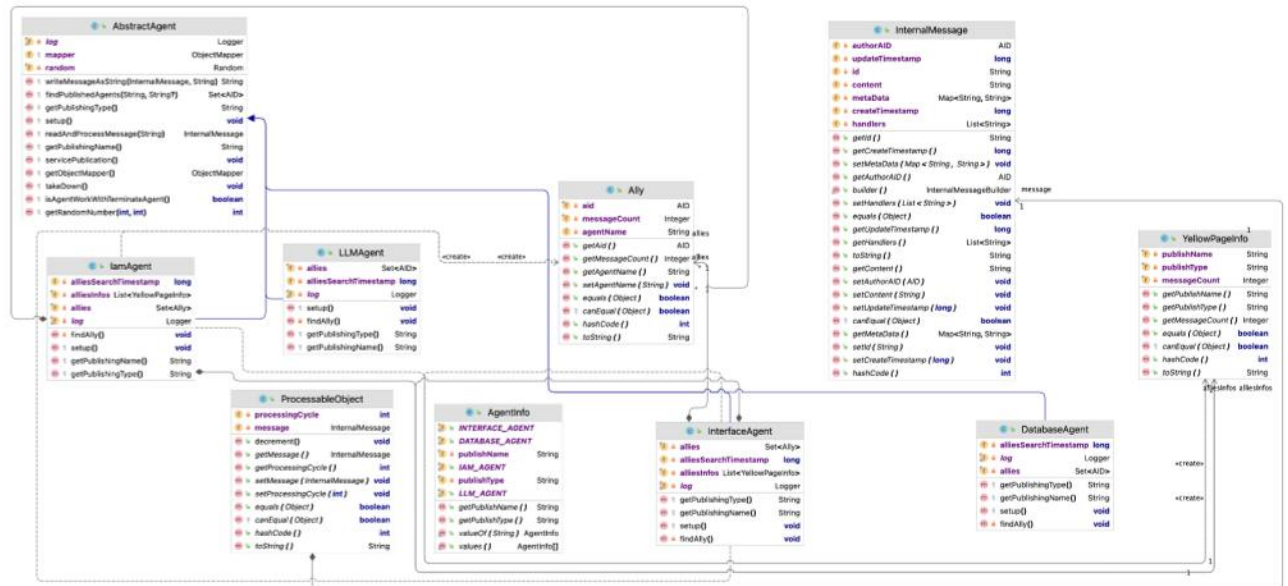


Рисунок 16 – Расширенная диаграмма классов мультиагентной системы

6.1 Конфигурация агента

Инициализацией распределенного алгоритма является создание агентов и их регистрация в главном контейнере платформы. Последовательность создания экземпляров агентов описана в конфигурационном файле проекта, представленном на рисунке 17.

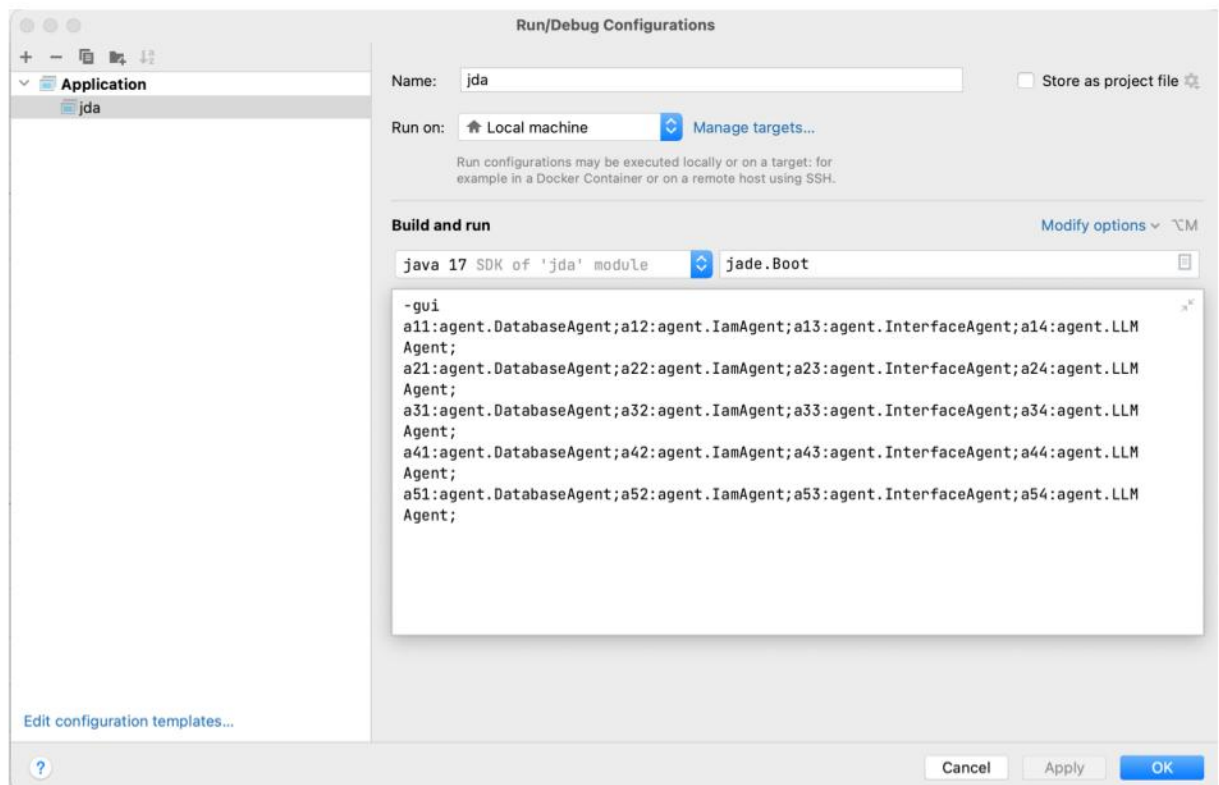


Рисунок 17 – Конфигурационный файл развертывания системы

После создания агентов, каждый экземпляр регистрируется в сервисе желтых страниц, который необходим для динамического поиска агентов по заданным критериям. При публикации агента неотъемлемыми атрибутами являются идентификатор агента (AID) и тип публикуемого агента, в зависимости от роли и звена, указанных на концептуальном графе.

При успешной регистрации каждый агент осуществляет поиск агента-адресата сообщения. Данный поиск осуществляется через службу желтых страниц по заданным критериям, а именно по имени агента (не обязательно) и тип опубликованного экземпляра (обязательно).

В соответствии с возможностями, предоставляемыми платформой JADE, каждый агент может обладать совокупностью поведений, каждое из которых описывает действие агента в определенный момент времени. Причем, важной особенностью является то, что данные поведения выполняются в одном потоке, что позволяет избежать дополнительной логики для реализации многопоточной обработки данных.

Таким образом, каждый агент регистрирует поведение, тип которого зависит от функций, выполняемых агентом. Основным поведением каждого агента является циклическое поведение, описывающее поиск и обработку найденных сообщений, включая их отправку в отдельных случаях. Стоит отметить, что отправка сообщений возможна только в том случае, если агент-адресат был найден.

В соответствии с объектно-ориентированной концепцией программирования, в разработанном приложении применяется такое свойство систем как наследование, которое позволяет описать класс новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью.

Также используется такое свойство как абстракция. Абстрагирование – это способ выделить набор общих характеристик объекта, исключая из рассмотрения частные и незначимые. Соответственно, абстракция – это набор всех таких характеристик. Данное свойство реализовано выделение

наиболее общих данных и методов для всех агентов абстрактный класс `AbstractAgent`.

В данном классе представлены абстрактные методы, каждый из которых реализуется в производных классах:

- `String getPublishingType` - установка типа агента для публикации в сервисе желтых страниц,
- `String getPublishingName` - установка имени агента для публикации в сервисе желтых страниц,

Также в данном классе описаны наиболее общие методы, используемые всеми агентами приложения:

- `void setup()` - инициализация агента в системе,
- `void takeDown()` - деинициализация агента в системе и отмена регистрации опубликованного сервиса,
- `void servicePublication(@Nonnull String type)` - публикация в сервисе желтых страниц,
- `Set<AID> findPublishedAgents(@Nonnull String agentType, @Nullable String agentName)` - поиск агентов в сервисе желтых страниц.

Агент `Directory Facilitator` – это менеджер директорий, который представляет собой так называемую службу желтых страниц. В данной службе агенты имеют возможность публиковать информацию о себе и предоставляемых ими сервисах. С помощью данного менеджера директорий агент может осуществлять поиск других агентов, предоставляющих необходимые ему сервисы, и коммуникацию с ними. Для каждой платформы может существовать несколько менеджеров директорий. Пример использования менеджера директорий показан на рисунке 18.

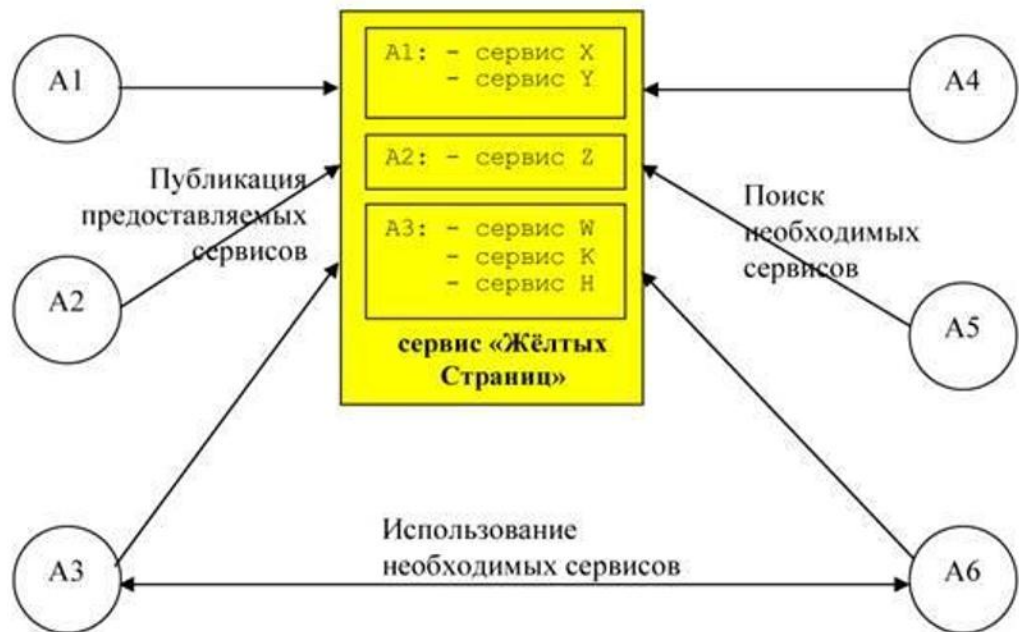


Рисунок 18 – Сервис желтых страниц

В рамках данной работы, агент наследуется от абстрактного класса и дополняет свое описание полями и методами, расширяющими базовый класс и придающими уникальность производному.

Агент InterfaceAgent:

Предоставляет пользовательский интерфейс для взаимодействия клиента с системой.

Поля:

- Logger logger – логгер для протоколирования запросов,
- InterfaceForm form – пользовательский интерфейс,
- Set<AID> allies – идентификаторы агентов-адресатов сообщения,
- String AGENT_PUBLISHED_NAME – имя публикации агента.

Методы:

- String getPublishingType() - установка типа агента для публикации в сервисе желтых страниц,
- String getPublishingName() - установка имени агента для публикации в сервисе желтых страниц,
- void setup() - инициализация агента в системе,
- void sendMessage(String message) - отправка сообщения агентам узла или другим агентам в сети,

- void findAllies() - поиск агентов узла.

Внутренние классы:

- class MessageService extends CyclicBehaviour - обработчик входящих сообщений. Осуществляется получение сообщения по заданному шаблону и отправку агентам узла в зависимости от его типа.

Агент DatabaseAgent:

Предоставляет интерфейс для взаимодействия узла с базой данных, осуществляет базовые операции и представляет собой CRUD-репозиторий.

Поля:

- Logger logger – логгер для протоколирования запросов,
- Set<AID> allies – идентификаторы агентов-адресатов сообщения,
- String AGENT_PUBLISHED_NAME – имя публикации агента.

Методы:

- String getPublishingType() - установка типа агента для публикации в сервисе желтых страниц,
- String getPublishingName() - установка имени агента для публикации в сервисе желтых страниц,
- void setup() - инициализация агента в системе,
- void sendMessage(String message) - отправка сообщения агентам узла,
- void findAllies() - поиск агентов узла.

Внутренние классы:

- class MessageService extends CyclicBehaviour - обработчик входящих сообщений. Осуществляется получение сообщения по заданному шаблону и отправку агентам узла в зависимости от его типа.

Агент IamAgent:

Предоставляет интерфейс для реализации регистрации/авторизации пользователя в системе и обеспечивает возможности для взаимодействия с пользователями.

Поля:

- `Logger logger` – логгер для протоколирования запросов,
- `Set<AID> allies` – идентификаторы агентов-адресатов сообщения,
- `String AGENT_PUBLISHED_NAME` – имя публикации агента.

Методы:

- `String getPublishingType()` - установка типа агента для публикации в сервисе желтых страниц,
- `String getPublishingName()` - установка имени агента для публикации в сервисе желтых страниц,
- `void setup()` - инициализация агента в системе,
- `void sendMessage(String message)` - отправка сообщения агентам узла и агентам сети,
- `void findAllies()` - поиск агентов узла.

Внутренние классы:

- `class MessageService extends CyclicBehaviour` - обработчик входящих сообщений. Осуществляется получение сообщение по заданному шаблону и отправка агентам узла в зависимости от его типа.

Агент `LLMAgent`:

Предоставляет интерфейс для взаимодействия с нейронной сетью.

Поля:

- `Logger logger` – логгер для протоколирования запросов,
- `Set<AID> allies` – идентификаторы агентов-адресатов сообщения,
- `String AGENT_PUBLISHED_NAME` – имя публикации агента.

Методы:

- `String getPublishingType()` - установка типа агента для публикации в сервисе желтых страниц,
- `String getPublishingName()` - установка имени агента для публикации в сервисе желтых страниц,
- `void setup()` - инициализация агента в системе,
- `void sendMessage(String message)` - отправка сообщения агентам узла и агентам сети,

- void findAllies() - поиск агентов узла.

Внутренние классы:

- class MessageService extends CyclicBehaviour - обработчик входящих сообщений. Осуществляется получение сообщение по заданному шаблону и отправку агентам узла в зависимости от его типа.

6.2 Поиск агентов

Динамическая инициализация и публикация агентов предоставляет возможности для реализации динамического поиска агентов по ранее описанному справочнику классификации и группировки агентов.

Агент, находясь в сервисе желтых страниц, доступен для его поиска по заданным параметрам, к которым относят обязательный – тип агента и необязательный – наименование агента. По заданным критериям составляется шаблон поиска узлов и формируется запрос выборки. Реализация метода поиска агентов в сервисе желтых страниц представлена ниже.

```
/**
 * Поиск агентов в сервисе желтых страниц
 *
 * @param agentType тип агента
 * @param agentName наименование агента
 * @return множество идентификаторов найденных объектов
 */
protected Set<AID> findPublishedAgents(@Nonnull String
agentType, @Nullable String agentName) {
    log.info("Try to find agents in yellow pages service by
name: {}", agentName);

    DFAgentDescription template = new DFAgentDescription();
    ServiceDescription serviceDescription = new
ServiceDescription();

    if (agentName != null) {
        serviceDescription.setName(agentName);
```

```

    }
    serviceDescription.setType(agentType);
    template.addServices(serviceDescription);

    try {
        return Arrays.stream(DFService.search(this,
template))
                        .map(DFAgentDescription::getName)
                        .collect(Collectors.toSet());
    } catch (FIPAException ex) {
        log.error("An error occurred while searching for a
 {}: {}", agentName, ex.getMessage());
    }

    return Collections.emptySet();
}

```

Вызов данного метода осуществляется каждым агентом при его инициализации и по расписанию в соответствии с циклически планируемым режимом.

Для определения уникальных идентификаторов агентов формируется ряд взаимодействующих агентов, который представлен в виде списка объектов содержащих наименование и тип искомого агента, а также количество отправляемых ему сообщений.

```

private final List<YellowPageInfo> alliesInfos =
Arrays.asList(
    new YellowPageInfo(INTERFACE_AGENT.getPublishName(),
INTERFACE_AGENT.getPublishType(), 1),
    new YellowPageInfo(DATABASE_AGENT.getPublishName(),
DATABASE_AGENT.getPublishType(), 2),
    new YellowPageInfo(IAM_AGENT.getPublishName(),
IAM_AGENT.getPublishType(), 1),
    new YellowPageInfo(LLM_AGENT.getPublishName(),
LLM_AGENT.getPublishType(), 1)
);

```

На основе сформированных данных, для каждой указанной записи списка выполняется поиск агента следующим методом:

```
private void findAlly() {
    alliesInfos.forEach(i ->
findPublishedAgents(i.getPublishType(), i.getPublishName())
        .stream()
        .findFirst()
        .ifPresent(aid ->
allies.add(new Ally(aid, i.getMessageCount()))
    );

    this.alliesSearchTimestamp = System.currentTimeMillis();
    log.info("Find allies result: {} for agent: {}",
allies.stream().map(Ally::getAid).map(AID::getName).collect(Collectors.toList()), getName());
}
```

Результатом выполнения итерации по каждой записи является формирование множества агентов - получателей сообщений, которые в дальнейшем используются в рамках циклического режима работы, осуществляющего обработку данных.

Помимо поиска узлов для отправки сообщений, алгоритм также поддерживает возможности динамического поиска агентов-отправителей сообщений. Аналогичным образом задается структура отправителей в виде списка с указанием наименования, типа и количества ожидаемых от узла сообщений.

```
private final List<YellowPageInfo> sendersInfos =
Arrays.asList(
    new YellowPageInfo(LLM_AGENT.getPublishName(),
LLM_AGENT.getPublishType(), 1),
    new YellowPageInfo(DATABASE_AGENT.getPublishName(),
DATABASE_AGENT.getPublishType(), 1)
);
```

Аналогичным образом по данной структуре осуществляется поиск отправителей и формируется список агентов, от которых ожидается получение определенного количества сообщений.

```
private void findSenders() {  
    sendersInfos.forEach(i ->  
senders.addAll(findPublishedAgents(i.getPublishType(),  
i.getPublishName())));  
  
    this.sendersSearchTimestamp =  
System.currentTimeMillis();  
    log.info("Find senders result: {} for agent: {}",  
senders.stream().map(AID::getName).collect(Collectors.toList()),  
getAID().getName());  
}
```

6.3 Балансировка и масштабируемость

Каждый узел метакомпьютера, представленного на рисунке 9, осуществляет процессинговую обработку данных передаваемых как от пользователя к агенту, так и от агента к агенту. Причем, в процессе выполнения алгоритма может возникнуть необходимость увеличить количество узлов на конкретном шаге выполнения или же наоборот, уменьшить их.

Для возможности балансировки и контроля нагрузки реализован динамический поиск агентов в сервисе желтых страниц, а также автоматическая публикация каждого агента в данном сервисе, после инициализации базовых свойств. Публикация узлов осуществляется на основе разработанного описания и классификации агентов распределенного алгоритма относительно выполняемой функции и местонахождения в распределенном алгоритме. Далее представлен фрагмент кода, описывающий классификацию и группировку узлов.

```
public enum AgentInfo {  
    INTERFACE_AGENT("interface_agent", "task-creator"),  
    DATABASE_AGENT("database_agent", "task-enter"),  
    IAM_AGENT("iam_agent", "task-enter"),
```

```
LLM_AGENT("llm_agent", "task-enter");
```

```
private final String publishName;  
private final String publishType;  
}
```

Каждый узел переопределяет абстрактные методы, позволяющие получить тип и наименование агента в сервисе желтых страниц.

```
/**  
 * Установка типа агента для публикации в сервисе желтых  
страниц  
 *  
 * @return тип агента при публикации в сервисе желтых  
страниц  
 */  
protected abstract String getPublishingType();  
  
/**  
 * Установка имени агента для публикации в сервисе желтых  
страниц  
 *  
 * @return имя агента при публикации в сервисе желтых  
страниц  
 */  
protected abstract String getPublishingName();
```

Примером реализации может служить агент пользовательского интерфейса, реализация абстрактных методов которого представлена ниже:

```
@Override  
protected String getPublishingType() {  
    return INTERFACE_AGENT.getPublishType();  
}  
  
@Override  
protected String getPublishingName() {  
    return INTERFACE_AGENT.getPublishName();  
}
```

Определив наименования и тип, осуществив инициализацию агента в `super`-методе, производится непосредственная публикация агента в сервисе желтых страниц.

```
/**
 * Инициализация агента в системе
 */
@Override
protected void setup() {
    log.info("Agent {} start working", getName());

    this.mapper = getObjectMapper();

    servicePublication();
}
```

Такие составляющие как уникальный идентификатор, тип и наименование определяют описание сервиса, с помощью которого осуществляется динамический поиск агентов в системе.

```
/**
 * Публикация агента в сервисе желтых страниц
 */
protected void servicePublication() {
    log.info("Try to publish agent {} in yellow pages
service", getName());

    DFAgentDescription dfAgentDescription = new
    DFAgentDescription();
    ServiceDescription serviceDescription = new
    ServiceDescription();

    dfAgentDescription.setName(getAID());
    serviceDescription.setType(getPublishingType());
    serviceDescription.setName(getPublishingName());
    dfAgentDescription.addServices(serviceDescription);
}
```

```

try {
    DFService.register(this, dfAgentDescription);
    log.info("Agent {} successfully published in yellow
pages", getName());
} catch (FIPAException e) {
    log.error("Agent {} publication in yellow pages
unavailable", getName());
    takeDown();
}
}

```

Определив возможности динамической регистрации агентов, осуществляется их динамическое управление и конфигурация. Возможностями платформы предоставляется несколько вариантов управления, а именно:

- ручное администрирование - создание, удаление, активация и деактивация ранее описанных в программе агентов. На рисунке 19 приводится пример создания агента в уже запущенном алгоритме,
- конфигурация окружения для запуска распределенного алгоритма. На рисунке 17 представлен конфигурационный файл, описывающий динамическое создание и публикацию узлов с заданными наименованиями и аргументами.

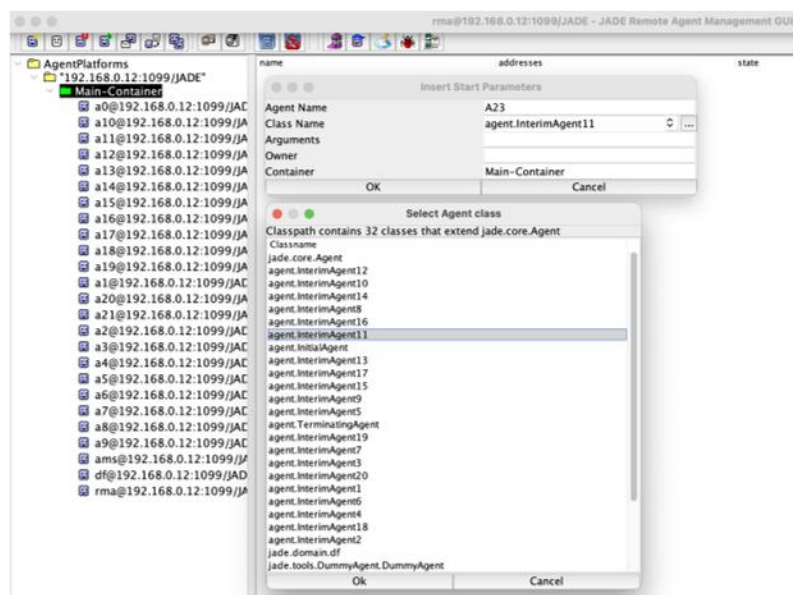


Рисунок 19 – Администрирование агентов

А также текстовое описание конфигурации:

```
-gui  
a11:agent.DatabaseAgent;a12:agent.IamAgent;a13:agent.InterfaceAgent;a14:agent.LLMAgent;  
a21:agent.DatabaseAgent;a22:agent.IamAgent;a23:agent.InterfaceAgent;a24:agent.LLMAgent;  
a31:agent.DatabaseAgent;a32:agent.IamAgent;a33:agent.InterfaceAgent;a34:agent.LLMAgent;  
a41:agent.DatabaseAgent;a42:agent.IamAgent;a43:agent.InterfaceAgent;a44:agent.LLMAgent;  
a51:agent.DatabaseAgent;a52:agent.IamAgent;a53:agent.InterfaceAgent;a54:agent.LLMAgent;
```

6.4 Взаимодействие агентов

В соответствии поставленным заданием реализована пиринговая сеть, выполняемая в сети ЭВМ на основе парадигмы передачи сообщений между интеллектуальными агентами. На рисунке 20 представлена расширенная пиринговая сеть, описывающая декларативные и процедурные знания о распределенном приложении, выполняемом в сетевой среде.

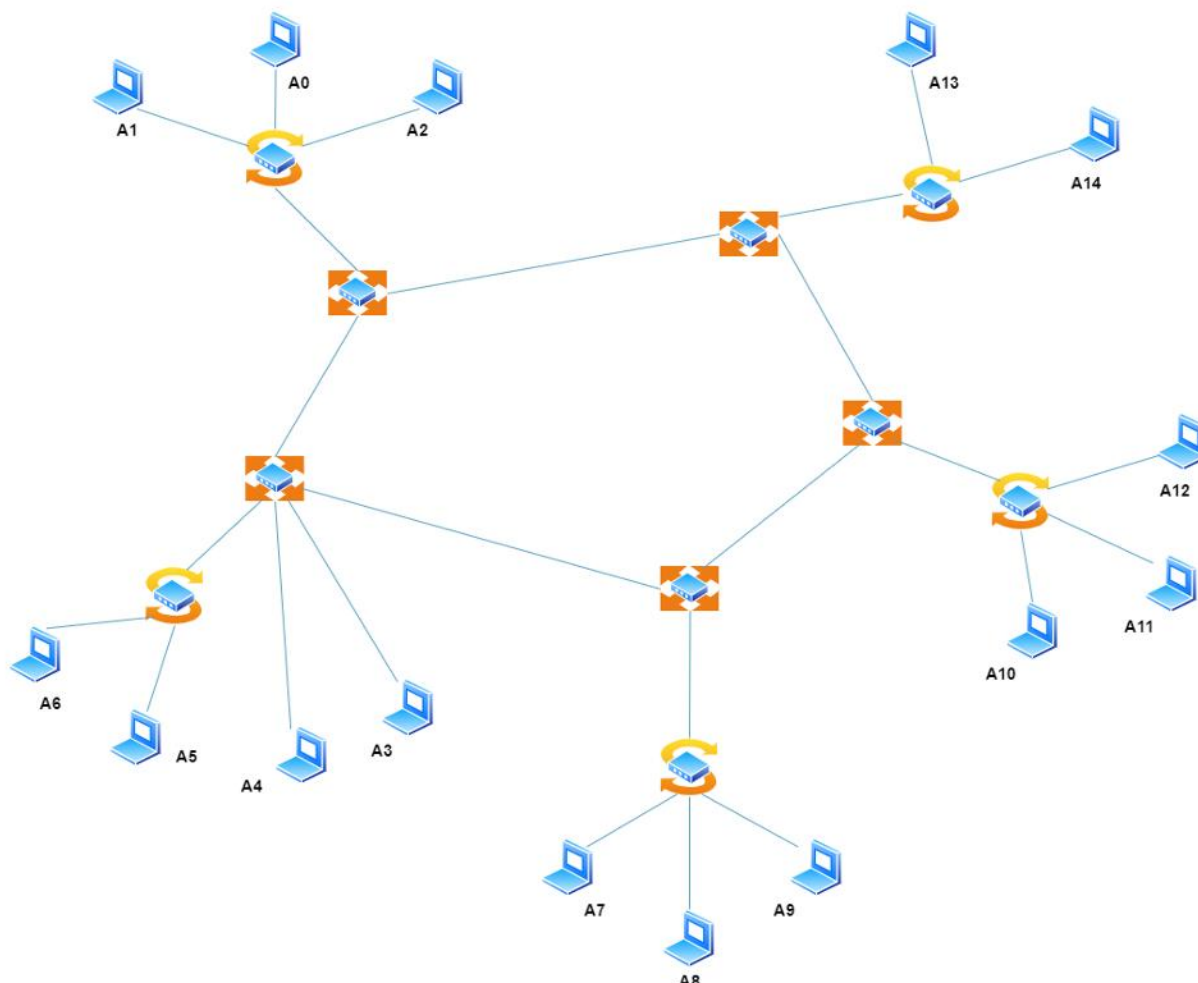


Рисунок 20 – Диаграмма распределенной обработки данных в пиринговой сети

При помощи сообщений в распределенной мультиагентной системе передается управление от одного оператора другому. Сообщения могут содержать результаты вычислений, запросы к базам данных и результаты выполнения данных запросов.

Представленная на рисунке 20 сеть является сложной моделью, состоящей из отдельных простых фрагментов, таких как:

- последовательность. На рисунке 21 дано графическое представление фразы “оператор A_7 , размещенный на узле Y_4 , передает сообщение оператору A_8 , размещенному на узле Y_4 ”,

- выбор передачи управления (“switch-or”). Описывает передачу сообщения по одной из ветвей распределенного алгоритма по значению переменной k . Возможен вариант реализации, при котором каждое результирующее сообщение передается различным операторам (рисунок 22),
- параллельное выполнение (“fork-join”). Данный фрагмент продемонстрирован на рисунке 23 и описывает передачу сообщений по параллельным ветвям распределенного алгоритма.

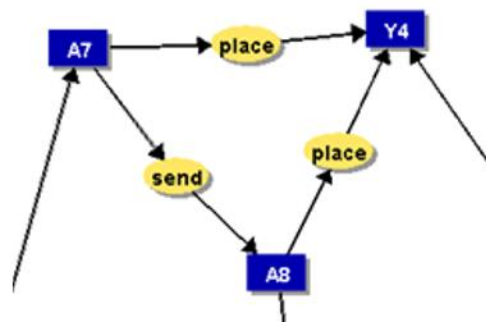


Рисунок 21 – Фрагмент последовательность

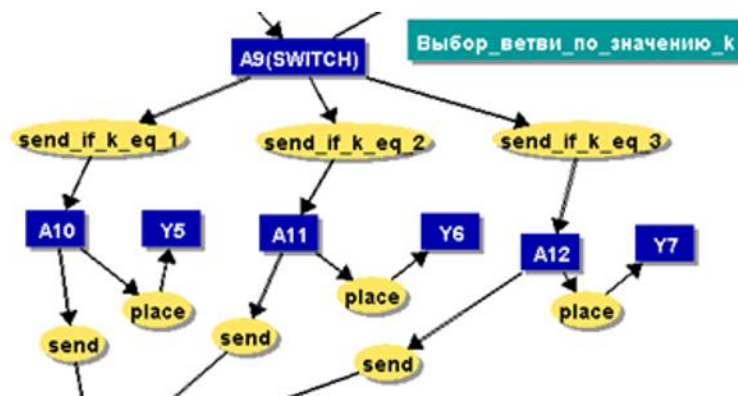


Рисунок 22 – Фрагмент "switch-or"

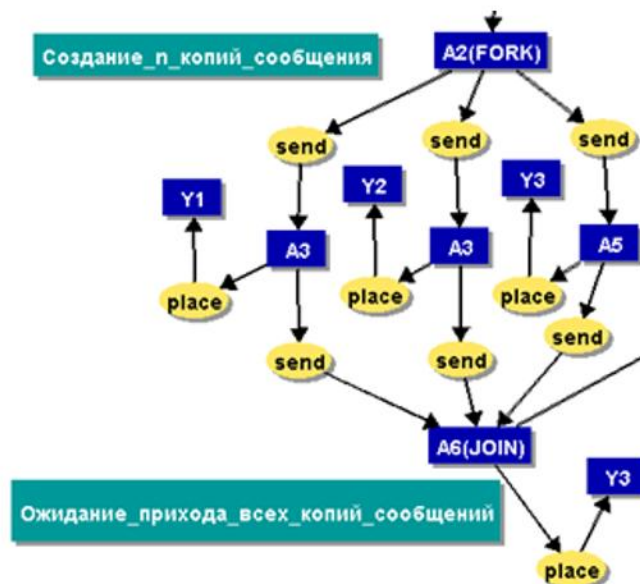


Рисунок 23 – Фрагмент "fork-join"

В соответствии с рисунком 19, взаимодействие между агентами обеспечивается созданием стартового сообщения с помощью пользовательского интерфейса агента InterfaceAgent. По нажатию кнопки выбранное сообщение отправляется в тот узел, в рамках которого существует данный агент. Важным условием является тот факт, что агенты-получатели данного сообщения зарегистрированы в сервисе желтых страниц и найдены как адресаты агентом InterfaceAgent.

В зависимости от выбранного пользователем действия осуществляется коммуникация между агентами внутри узла или агентами всей сети. Внутри узла и по сети сообщения передаются с помощью встроенного механизма передачи данных между интеллектуальными агентами - ACL Message. Внутри узла, в зависимости от выбранной функции, сообщение может быть отправлено:

- агенту DatabaseAgent для создания чата, просмотра списка чатов, просмотра истории чатов и сообщений в каждом из них. В свою очередь агент работы с базой данных публикует в общую шину событие публикации нового сообщения для обеспечения возможности подписаться и среагировать на данное действие,

- агенту IamAgent для регистрации и авторизации пользователя в системе, после чего узел становится доступным для взаимодействия внутри

системы, для обеспечения безопасности при отправке широковебательных запросов по сети,

– агенту LLMAgent для формирования промта, на основе которого будет отправлен запрос в нейросеть, для получения ответа от нейросети, для оценки качества ответа, которое в дальнейшем может использовать для реализации механизма обучения с учителем на основе оценок пользователей, для обучения модели на основе предоставляемых данных.

Для визуального представления публикации и взаимодействия агентов на рисунке 24 представлена диаграмма развертывания метакompьютера.

Пиринговая сеть на основе
мультисетевой системы

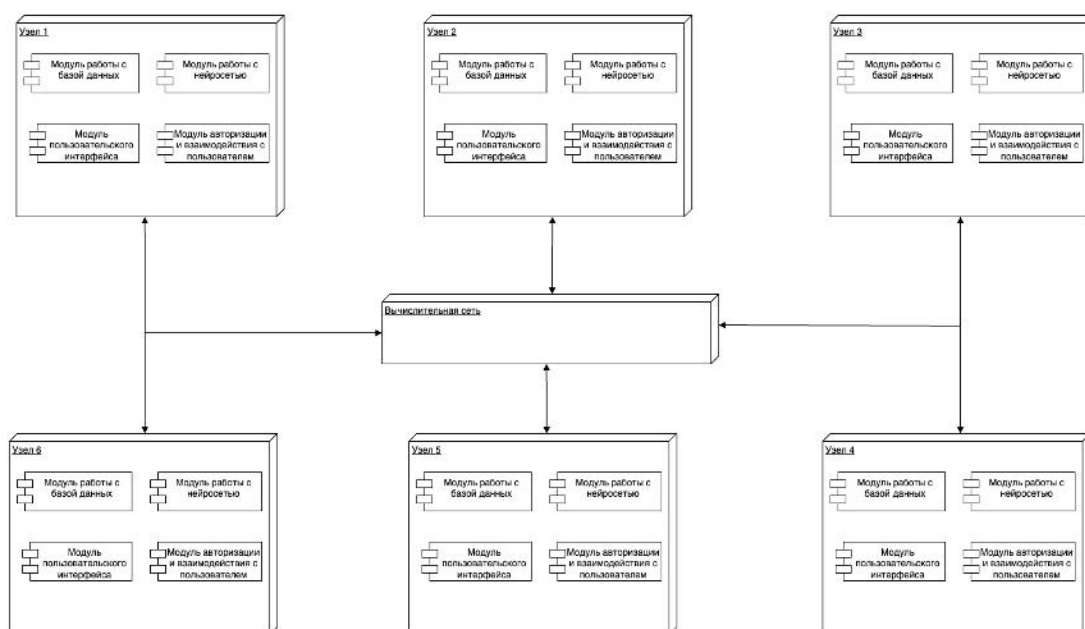


Рисунок 24 – Диаграмма развертывания метакompьютера

6.5 Взаимодействие с нейронной сетью

Для взаимодействия с нейронной сетью был разработан API, позволяющий формировать промты - короткие формулировки, которые предоставляют информацию нейросети о том, что именно требуется от нее, и получать на основе запроса сгенерированный текст - результат ответа, который будет отправляться агенту интерфейса InterfaceAgent.

API представляет собой совокупность классов - загрузчика нативной библиотеки, модели, которые являются основой передачи данных и вспомогательные классы, обеспечивающие возможность выполнять утилитные методы, необходимые для работы программы. На рисунке 25 представлена обобщенная диаграмма классов API для работы с моделью.

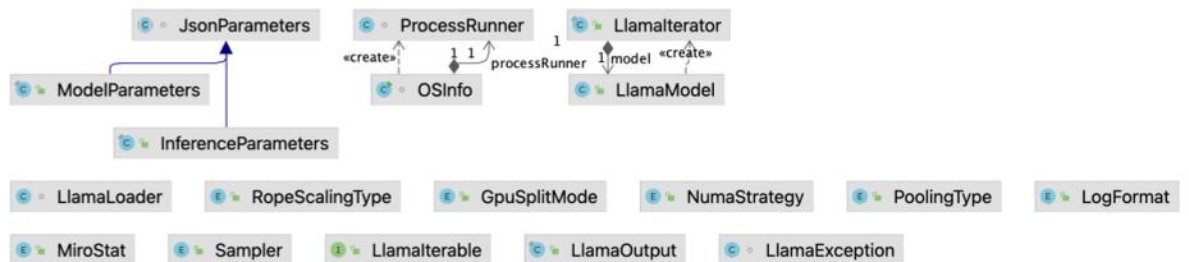


Рисунок 25 – Обобщенная диаграмма классов API для работы с моделью

Основным компонентом среди данных классов является класс-загрузчик, который осуществляет подгрузку скомпилированного нативного файла в JVM. Нативный файл LLM может быть скомпилирован под различные архитектуры в зависимости от исполняемой машины. В данном случае, была произведена компиляция под архитектуру arm64 операционной системы MacOS и для исполнения на процессоре Apple M1 (рисунок 26).



Рисунок 26 – скомпилированные нативные файлы под архитектуру arm64

Компиляция нативных файлов осуществляется следующей последовательностью шагов:

- сборка проекта с использованием фреймворка maven

```
mvn clean compile
```

– компиляция нативных файлов под архитектуру исполняемой машины с необходимыми опциями для операционной системы

```
cmake -DLLAMA_METAL=1 -DLLAMA_CCACHE=OFF
```

– и последнее - билд полученного результата для дальнейшего использования с помощью утилиты командной строки cmake

```
cmake --build . --config Release
```

Для дальнейшего взаимодействия с LLM определяется предобученная модель, с которой в дальнейшем будет осуществляться взаимодействие. Для данной работы была выбрана и подгружена модель `mistral-7b-instruct-v0.2.Q2_K.gguf`. Подгрузка модели осуществляется за счет реализованного API и настраивается с помощью указания пути до файла и конфигурационных настроек модели, например:

```
ModelParameters modelParams = new ModelParameters()  
.setModelFilePath("models/mistral-7b-instruct-v0.2.Q2_K.gguf")  
  
.setNGpuLayers(43)  
.setDisableLog(true);
```

Затем определяется стартовый промт для взаимодействия с моделью, чтобы задать контекст диалога:

```
String systemPrompt = "Это соглашение между пользователем и  
искусственным интеллектом, который является дружелюбным,  
полезным, умным, добрым и честным, хорошо отвечающим на вопросы  
с высоким быстродействием, точностью и качеством ответов,  
никогда не прерывающим и не отказывающимся от ответов на  
вопросы"
```

Дальнейшим шагом является лишь описание циклической модели взаимодействия пользователя (пользовательского интерфейса) и модели, которое может выглядеть следующим образом:

```
BufferedReader reader = new BufferedReader(new  
InputStreamReader(System.in, StandardCharsets.UTF_8));  
  
try (LlamaModel model = new LlamaModel(modelParams)) {  
    System.out.print(system);  
    String prompt = system;
```

```

while (true) {
    prompt += "\nUser: ";
    System.out.print("\nUser: ");
    String input = reader.readLine();
    prompt += input;
    System.out.print("Model: ");
    prompt += "\nModel: ";

    InferenceParameters inferParams = new
InferenceParameters(prompt)
        .setTemperature(0.7f)
        .setPenalizeNl(true)
        .setMiroStat(MiroStat.V2)
        .setStopStrings("User:");

    for (LlamaOutput output :
model.generate(inferParams)) {
        System.out.print(output);
        prompt += output;
    }
}
}

```

Результат исполнения представленного выше кода и взаимодействие агента с моделью искусственного интеллекта, представлен на рисунке 27.

```

User: Расскажи, что такое искусственный интеллект?
Model: {"tid":"0x16e2b3980","timestart":1716536772,"level":"INFO","function":"launch_slot_with_task","line":1133,"msg":"slot is processing task","id_slot":3,"id_task":0}
{"tid":"0x16e2b3980","timestart":1716536772,"level":"INFO","function":"update_slots","line":2265,"msg":"kv cache re [p0, end)","id_slot":0,"id_task":0,"p0":0}
Artificial Intelligence (AI) refers to the development of computer systems that can perform tasks that normally require human intelligence, such as understanding natural language, recognizing patterns, and making logical decisions.

{"tid":"0x16e2b3980","timestart":1716536784,"level":"INFO","function":"print_timings","line":326,"msg":"prompt eval time      = 9852.08 ns /   91 tokens ( 99.47 ns per token,   10.05 tokens per second)","id_slot":3,"id_task":0,"t_prompt_processing":9852.085,"n_prompt_tokens_processed":91,"t_token":99.47258241758242,"n_tokens_second":13.053321402446676}
{"tid":"0x16e2b3980","timestart":1716536784,"level":"INFO","function":"print_timings","line":336,"msg":"generation eval time = 3429.63 ns /   42 runs ( 81.44 ns per token,   12.28 tokens per second)","id_slot":3,"id_task":0,"t_token_generation":3429.633,"n_decoded":42,"t_token":81.44364285714285,"n_tokens_second":12.278429167934707}
{"tid":"0x16e2b3980","timestart":1716536784,"level":"INFO","function":"print_timings","line":346,"msg":"total time = 12472.64 ns","id_slot":0,"id_task":0,"t_prompt_processing":9852.085,"t_token_generation":3429.633,"t_total":12472.679999999999}
{"tid":"0x16e2b3980","timestart":1716536784,"level":"INFO","function":"update_slots","line":1935,"msg":"slot released","id_slot":0,"id_task":0,"n_ctx":512,"n_past":132,"n_system_tokens":0,"n_cache_tokens":0,"truncated":false}
{"tid":"0x16e2b3980","timestart":1716536784,"level":"INFO","function":"update_slots","line":1956,"msg":"all slots are idle"}

User: Расскажи на русском языке что такое искусственный интеллект?
Model: {"tid":"0x16e2b3980","timestart":1716536807,"level":"INFO","function":"launch_slot_with_task","line":1133,"msg":"slot is processing task","id_slot":3,"id_task":43}
{"tid":"0x16e2b3980","timestart":1716536807,"level":"INFO","function":"update_slots","line":2265,"msg":"kv cache re [p0, end)","id_slot":0,"id_task":43,"p0":0}
Искусственный интеллект (ИИ) - это разработка компьютерных систем, которые могут выполнять задачи, которые обычно требуют человеческого интеллекта, такие как понимание естественного языка, распознавание изображений и принятие рациональные решения.

{"tid":"0x16e2b3980","timestart":1716536824,"level":"INFO","function":"print_timings","line":326,"msg":"prompt eval time      = 6697.16 ns /  159 tokens ( 42.12 ns per token,   23.74 tokens per second)","id_slot":3,"id_task":43,"t_prompt_processing":6697.159,"n_prompt_tokens_processed":159,"t_token":42.12049685534891,"n_tokens_second":23.74541035036422}
{"tid":"0x16e2b3980","timestart":1716536824,"level":"INFO","function":"print_timings","line":336,"msg":"generation eval time = 10003.55 ns /   89 runs ( 112.40 ns per token,    8.93 tokens per second)","id_slot":3,"id_task":43,"t_token_generation":10003.553,"n_decoded":89,"t_token":112.39947191811235,"n_tokens_second":8.89683895319955}
{"tid":"0x16e2b3980","timestart":1716536824,"level":"INFO","function":"print_timings","line":346,"msg":"total time = 16700.71 ns","id_slot":0,"id_task":43,"t_prompt_processing":6697.159,"t_token_generation":10003.553,"t_total":16700.712}
{"tid":"0x16e2b3980","timestart":1716536824,"level":"INFO","function":"update_slots","line":1935,"msg":"slot released","id_slot":0,"id_task":43,"n_ctx":512,"n_past":247,"n_system_tokens":0,"n_cache_tokens":0,"truncated":false}
{"tid":"0x16e2b3980","timestart":1716536824,"level":"INFO","function":"update_slots","line":1956,"msg":"all slots are idle"}

```

Рисунок 27 – Результат взаимодействия пользователя с моделью искусственного интеллекта

6.6 Обучение модели

В данной работе применяется обучение большой языковой модели (LLM) на основе весов - это процесс, при котором модель обучается на наборе данных, используя предварительно обученные веса другой модели. Этот подход часто используется для ускорения процесса обучения и улучшения производительности модели.

Вот общий процесс обучения LLM на основе весов:

- выбор исходной модели: была выбрана модель, используемая в качестве основы для новой модели. Эта модель представляет собой модель, которая уже обучена на большом наборе данных и демонстрирует хорошие результаты в нужной задаче.
- загрузка весов: были выбраны и скачены веса модели под конкретного агента. Скачанные веса представляют собой файлы в формате HDF5.
- обучение: используется метод переноса обучения, такой как fine-tuning, для обучения новой модели на новом наборе данных. Fine-tuning включает в себя обучение модели на новом наборе данных, но с использованием предварительно обученных весов. Это помогает модели быстрее адаптироваться к новому набору данных и улучшить ее производительность.
- тестирование и оценка: после обучения проведено тестирование и оценка модели, чтобы убедиться, что она работает должным образом.

ЗАКЛЮЧЕНИЕ

Результатом выполнения данной квалификационной работы является успешно реализованный в соответствии с поставленной задачей проект метакомпьютерных систем на основе мультиагентной платформы, архитектуре пиринговых сетей, с применением искусственного интеллекта.

В работе предложен подход к реализации распределенного метакомпьютерного приложения в среде компьютерной сети на основе комплексного использования предметно-ориентированной мультиагентной системы и концепции агентно-ориентированного программирования в соответствии с архитектурой пиринговой сети, описывающей обмен сообщениями и обработку данных программными агентами. Использована динамично развивающаяся открытая агентная платформа, обеспечивающая широкий спектр функциональных возможностей при работе с агентами. Эта платформа является эффективной средой разработки и управления агентными приложениями в различных предметных областях, в том числе в области метакомпьютинга. В систему внедрена большая языковая модель, использующаяся каждым узлом системы и обеспечивающая генерацию токенов в соответствии с поступающими запросами. В агентно-базированной среде эффективно реализованы межузловые взаимодействия внутри пиринговой сети, организация параллельной обработки данных одним или парой агентов в рамках узла. Предлагаемые методы решения задач позволяют оперативно создавать распределенные приложения, работающие в облачно-сетевой среде; время разработки и отладки при этом сокращается примерно на порядок.

Дальнейшее развитие метакомпьютерной системы подразумевает параллельное обучение большой языковой модели для обеспечения качества генерируемых ответов, а также оптимизация работы алгоритма распределенной обработки данных в сети.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Java Agent Development Environment (JADE). – URL: <http://jade.tilab.com/>. [Electronic resource]. Blank from the screen. Access is free (date of access: 1.09.2022).
2. Волчихин В. И., Карамышева Н. С., Зинкин С. А., Гурин Е. И. Алгоритмика, логика и моделирование агентно-базированных метакомпьютерных систем с повышенным уровнем параллельности // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2022. – № 2. – С. 5–25. – DOI: 10.21685/2072-3059-2022-2-1.
3. Волчихин В. И., Зинкин С. А., Карамышева Н. С. Организация функционирования облачно-сетевых распределенных вычислительных систем с архитектурой «агенты как сервисы» // Известия высших учебных заведений. Поволжский регион. Технические науки. – 2019. – № 4. – С. 27–50. – DOI: 10.21685/2072-3059-2019-4-3.
4. Карамышева Н. С., Свищев Д. С., Попов К. В., Зинкин С. А. Реализация агентно-базированных метакомпьютерных систем и приложений // Известия Юго-Западного государственного университета. – 2022. – № 26(1). – С. 148-171. DOI: <https://doi.org/10.21869/2223-1560-2022-26-1-148-171>.

ПРИЛОЖЕНИЕ А

(обязательное)

Реализация агентно-базированных метакомпьютерных систем с использованием нейросетевых технологий

Листинг программы

```
package agent;

import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;
import jade.core.AID;
import jade.core.Agent;
import jade.domain.DFService;
import jade.domain.FIPAAgentManagement.DFAgentDescription;
import jade.domain.FIPAAgentManagement.ServiceDescription;
import jade.domain.FIPAException;
import lombok.extern.slf4j.Slf4j;
import model.InternalMessage;

import javax.annotation.Nonnull;
import javax.annotation.Nullable;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import java.util.Optional;
import java.util.Random;
import java.util.Set;
import java.util.stream.Collectors;

/**
 * Абстрактный класс для инициализации объекта-агента в системе
 * распределенного алгоритма
 */
```

```

@Slf4j
public abstract class AbstractAgent extends Agent {
    private final Random random = new Random();
    /**
     * Маппер для сериализации/десериализации объектов в JSON
     */
    protected ObjectMapper mapper;

    /**
     * Установка типа агента для публикации в сервисе желтых
страниц
     *
     * @return тип агента при публикации в сервисе желтых страниц
     */
    protected abstract String getPublishingType();

    /**
     * Установка имени агента для публикации в сервисе желтых
страниц
     *
     * @return имя агента при публикации в сервисе желтых страниц
     */
    protected abstract String getPublishingName();

    /**
     * Инициализация агента в системе
     */
    @Override
    protected void setup() {
        log.info("Agent {} start working", getName());

        this.mapper = getObjectMapper();

        servicePublication();
    }
}

```

```

/**
 * Деинициализация агента в системе
 */
@Override
protected void takeDown() {
    try {
        DFService.deregister(this);
    } catch (FIPAException e) {
        log.error("An error occurred while deactivation agent
{} service", getName());
    }
    log.warn("Agent {} stop working", getName());
}

/**
 * Публикация агента в сервисе желтых страниц
 */
protected void servicePublication() {
    log.info("Try to publish agent {} in yellow pages service",
getName());

    DFAgentDescription dfAgentDescription = new
DFAgentDescription();
    ServiceDescription serviceDescription = new
ServiceDescription();

    dfAgentDescription.setName(getAID());
    serviceDescription.setType(getPublishingType());
    serviceDescription.setName(getPublishingName());
    dfAgentDescription.addServices(serviceDescription);

    try {
        DFService.register(this, dfAgentDescription);
        log.info("Agent {} successfully published in yellow
pages", getName());
    } catch (FIPAException e) {

```

```

        log.error("Agent {} publication in yellow pages
unavailable", getName());
        takeDown();
    }
}

/**
 * Поиск агентов в сервисе желтых страниц
 *
 * @param agentType тип агента
 * @param agentName наименование агента
 * @return множество идентификаторов найденных объектов
 */
protected Set<AID> findPublishedAgents(@Nonnull String
agentType, @Nullable String agentName) {
    log.info("Try to find agents in yellow pages service by
name: {}", agentName);

    DFAgentDescription template = new DFAgentDescription();
    ServiceDescription serviceDescription = new
ServiceDescription();

    if (agentName != null) {
        serviceDescription.setName(agentName);
    }
    serviceDescription.setType(agentType);
    template.addServices(serviceDescription);

    try {
        return Arrays.stream(DFService.search(this, template))
            .map(DFAgentDescription::getName)
            .collect(Collectors.toSet());
    } catch (FIPAException ex) {
        log.error("An error occurred while searching for a {}:
{}", agentName, ex.getMessage());
    }
}

```

```

        return Collections.emptySet();
    }

    /**
     * Чтение и первичная обработка полученного сообщения
     *
     * @param receiveMessage полученное сообщение
     * @return обработанное внутреннее сообщение
     */
    protected InternalMessage readAndProcessMessage(String
receiveMessage) {
        try {
            InternalMessage internalMessage =
mapper.readValue(receiveMessage, InternalMessage.class);

internalMessage.setUpdateTimestamp(System.currentTimeMillis());
            internalMessage.getHandlers().add(getAID().getName());

            return internalMessage;
        } catch (JsonProcessingException e) {
            log.error("Couldn't deserialize message content: {}",
receiveMessage);
            throw new IllegalArgumentException(e.getMessage());
        }
    }

    /**
     * Десериализация сообщения
     *
     * @param message объект внутреннего сообщения
     * @param defaultMessage сообщение по умолчанию
     * @return сообщение в формате JSON
     */

```

```

    protected String writeMessageAsString(InternalMessage message,
String defaultMessage) {
    try {
        return mapper.writeValueAsString(message);
    } catch (JsonProcessingException e) {
        log.warn("Couldn't serialize internal message, send
default text");
        return defaultMessage;
    }
}

```

```

protected boolean isAgentWorkWithTerminateAgent() {
    return Optional.ofNullable(getArguments())
        .map(a -> a[0])
        .map(String::valueOf)
        .map(Boolean::valueOf)
        .map(Boolean.TRUE::equals)
        .orElse(false);
}

```

```

protected int getRandomNumber(int min, int max) {
    return random.nextInt(max - min) + min;
}

```

```

/**

```

```

 * Конфигурирование маппера

```

```

 *

```

```

 * @return сконфигурированный маппер

```

```

 */

```

```

protected ObjectMapper getObjectMapper() {
    ObjectMapper objectMapper = new ObjectMapper();

```

```

objectMapper.setSerializationInclusion(JsonInclude.Include.NON_EMPTY);

```

```

    objectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);

    return objectMapper;
}

package enums;

import lombok.Getter;
import lombok.RequiredArgsConstructor;

@Getter
@RequiredArgsConstructor
public enum AgentInfo {
    INTERFACE_AGENT("interface_agent", "task-creator"),
    DATABASE_AGENT("database_agent", "task-enter"),
    IAM_AGENT("iam_agent", "task-enter"),
    LLM_AGENT("llm_agent", "task-enter");

    private final String publishName;
    private final String publishType;
}

package model;

import jade.core.AID;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

```

```

import java.util.Map;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class InternalMessage {
    /**
     * Идентификатор сообщения
     */
    private String id;
    /**
     * Дата и время создания сообщения в миллисекундах
     */
    private long createTimeStamp;
    /**
     * Идентификатор создателя сообщения
     */
    private AID authorAID;
    /**
     * Дата и время обновления сообщения в миллисекундах
     */
    private long updateTimeStamp;
    /**
     * Идентификаторы агентов, обработавших сообщение
     */
    private List<String> handlers = new ArrayList<>();
    /**
     * Содержание сообщения
     */
    private String content;
    /**
     * Метаданные сообщения
     */
    private Map<String, String> metaData = new HashMap();
}

```

```
package model;

import lombok.Data;

@Data
public class YellowPageInfo {
    private final String publishName;
    private final String publishType;
    private final Integer messageCount;
}
```

ПРИЛОЖЕНИЕ Б

Реализация агентно-базированных метакомпьютерных систем с использованием нейросетевых технологий

Презентация

Тема магистерской
диссертации: «Реализация
агентно-базированных
метакомпьютерных систем с
использованием
нейросетевых технологий»

Автор работы: магистрант группы 22ВВиМ1 Свищев Д.С.
Научный руководитель: д.т.н., профессор каф. ВТ ПГУ
Зинкин С.А.

Пенза 2024

Содержание

- > 1. Цели и задачи работы
- 2. Актуальность работы
- 3. Обзор аналогов
- 4. Постановка задачи
- 5. Инструменты разработки
- 6. Проектирование и разработка
- 7. Результаты работы
- 8. Подведение итогов

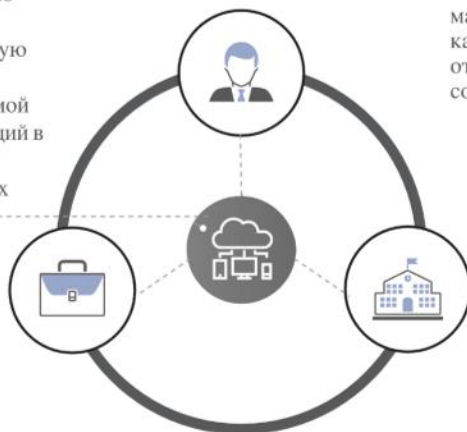
Цели и задачи

Метакомпьютерная система

Требуется разработать метакомпьютерную интеллектуальную систему на основе пиринговой сети и мультиагентной системы, реализующую графический интерфейс для взаимодействия пользователя с системой и программный интерфейс, включающий в себя совокупность узлов, каждый из которых состоит из интеллектуальных агентов

Бизнес

Реализация неограниченных по мощности, горизонтально масштабируемых интеллектуальных систем реального времени на основе вычислительных машин любой мощности



Потребитель

Обеспечение отказоустойчивых, масштабируемых, качественных систем, открытых к обучению для совершенствования результата

Государство

Обеспечение возможности проведения высокоточных и высокопроизводительных вычислений на основе платформы

Актуальность работы

1

Развитие искусственного интеллекта вещей

Увеличение
ежегодного
объема данных

В различных промышленных ситуациях с применением устройств с Интернетом вещей, в частности на предприятиях с сильно распределенной системой, регулярная отправка необработанных данных в ЦОД (центр обработки данных) или на основной сервер может оказаться невозможной.

Необходимость
принятия
решения в
реальном
времени

Увеличение
ежегодного
объема данных

Развитие
метакомпьютерных систем

Интеграция
высокопроизводительных
систем и
высокоресурсных машин

Активная
разработка и
внедрение
распределительных
вычислительных
систем

2

Потребность в высокопроизводительных системах

Характерной особенностью прикладных задач, возникающих в рамках крупномасштабных проектов, является потребность в использовании высокопроизводительных систем, вычислительных возможностей которых в отдельности не хватает для эффективного решения таких задач.

Содержание



1. Цели и задачи работы
2. Актуальность работы
3. Обзор аналогов
4. Постановка задачи
5. Инструменты разработки
6. Проектирование и разработка
7. Результаты работы
8. Подведение итогов

Постановка задачи

Требуется разработать метакомпьютерную систему, реализующую:

- 1 Распределенную обработку данных в сети
- 2 Параллельное выполнения запросов/задач
- 3 Реализацию интеллектуальных агентов и подключение нейронной сети к каждому узлу
- 4 Сохранение данных и их обмен между узлами
- 5 Возможность неограниченного горизонтального масштабирования за счет разных вычислительных мощностей

